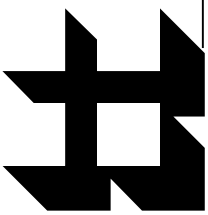

ГЛАВА 5

Элементы управления

Форма является основным звеном программы: ведь именно ее пользователь видит на экране все то время, которое посвящает работе с приложением. Но общение пользователя с программой происходит не просто через форму, сама по себе форма малоинформативна, а через те объекты, которые располагаются на этой форме.

При помощи этих объектов пользователь по мере возможности управляет работой программы, поэтому их и назвали элементами управления.

Элементами управления называются графические объекты, размещаемые на форме и используемые для отображения данных или выполнения каких-либо управляющих действий.

Существует условное деление элементов управления на визуальные и невизуальные. *Визуальными* элементами управления называют те, которые в принципе могут иметь визуальное отображение на форме, соответственно, другая часть элементов управления вообще не имеет специального экранного элемента, ассоциированного с ним, и поэтому называется *невизуальными* элементами управления. Иногда элементы управления называют **Control** по их английскому наименованию. К визуальным элементам управления относятся, например, `Label` (Этикетка), `TextBox` (Текстовое поле), `EditBox` (Редактируемое поле) и др. Невизуальными объектами являются `Custom` (Специальный пользовательский элемент), `HyperLink` (Ссылка).

Условность такой квалификации заключается в том, что некоторые визуальные элементы могут никак не отображаться на форме, если предпринять специальные меры для того, чтобы скрыть эти объекты. Например, если у объекта `PageFrame` скрыть вкладки и рамку, то внешне будет казаться, что никакого `PageFrame` и нет. Однако именно такой прием часто используется при создании пошаговых инструкций типа **Wizard**. Выполняя определенные инструкции и нажимая затем кнопку **Вперед**, вы переходите на следующую вкладку **PageFrame**.

Порядок размещения элементов управления на форме

В Visual FoxPro существует несколько способов размещения объектов на форме. Все они основываются на технологии "Drag-and-Drop" (Перенести-и-оставить). Объект захватывается мышью и переносится на нужное место. "Захватить" объект можно в нескольких местах:

- в специальном **Toolbar** с именем `Form Controls Toolbar`;
- в соответствующей библиотеке классов;
- в объекте `DataEnvironment`.

Создание элементов управления с помощью *Form Controls Toolbar*

Большую часть элементов управления можно взять из **Form Controls Toolbar**. Как правило, он открывается автоматически при открытии формы или класса на редактирование. Если у вас этого не произошло, то воспользуйтесь пунктом главного меню **View | Toolbar | Form Controls Toolbar**, чтобы отобразить его.

Чтобы "захватить" нужный элемент управления, выберите его пиктограмму на панели **Form Controls**, один раз нажмите левую кнопку мыши, затем установите курсор на место предполагаемого размещения элемента. Если вы сразу же отпустите кнопку мыши, объект будет создан с размерами по умолчанию. Если же вы, не отпуская левой кнопки мыши, нарисуете ею прямоугольник, то сможете сразу же установить нужный размер элемента. Впрочем, изменить размер и место расположения элемента вы сможете и после его размещения на форме. Для этого существуют несколько способов.

- Используйте мышь. Если у вас установлена настройка **Grid Lines** (главное меню **View | Grid Lines**), то при перемещении объекта с помощью мыши его положение и размер будут автоматически привязываться к ближайшей линии координатной сетки на форме. Если использовать клавишу <Ctrl>, то привязки не произойдет, и управление положением и размером объекта будет определяться в пикселах с точностью до одного пиксела.
- После выделения объекта мышью его положение и размер можно изменять с помощью клавиатуры, используя клавиши со стрелками. При нажатии клавиши <Shift> использование клавиш со стрелками приведет к изменению размера объекта.
- Размер и положение объекта можно задать явно с помощью задания значений свойств в окне **Properties**.

Еще одной особенностью **Form Controls** является возможность установить на форме несколько однотипных элементов сразу. Для этого используется кнопка **Button Lock**.

Если нажать эту кнопку (на ней изображен замочек), а потом выбрать пиктограмму какого-либо элемента, то кнопка фиксируется в нажатом состоянии, и щелчок мышью на форме приводит к созданию еще одного объекта. Чтобы отменить этот режим, достаточно еще раз нажать на кнопку **Button Lock** или на кнопку с изображением стрелки.

Создание элементов управления через библиотеки классов

Еще один способ создания элементов управления заключается в перетаскивании элемента управления из нужной библиотеки классов на форму. Предположим, у нас уже есть собственная библиотека классов, включенная в проект. В процессе редактирования формы переключитесь на окно проекта, перейдите в нем на вкладку **Classes**, раскройте нужную библиотеку, установите курсор на нужный класс, и, не отпуская кнопки мыши, перетащите его на форму. После того, как кнопка мыши будет отпущена, на форме будет создан элемент управления на базе вашего класса.

Этот механизм действует во всех библиотеках классов, каким бы способом вы его не открывали. Вместе с Visual FoxPro поставляется довольно большое количество готовых библиотек классов. Их можно открыть через дополнительные утилиты, доступные в пункте меню **Tools**: **Class Browser**, **Component Gallery**, **Toolbox**. Классы из этих библиотек можно поместить на форму так же, как и из библиотеки, включенной в проект. Но есть один нюанс: эти библиотеки поставляются с установленной кодовой страницей 1252 (Win Eng). Использовать их напрямую, без изменения кодовой страницы, нельзя. Изменить их кодовую страницу можно при помощи специальной программы изменения кодовых страниц CPZERO.PRG, которая поставляется вместе с Visual FoxPro.

Вы также можете включить свою библиотеку классов в **From Controls Toolbar**. Для этого установите курсор на пиктограмму с изображением книжек в **Form Controls Toolbar**, в появившемся окне выберите пункт **Add** и укажите файл своей библиотеки классов. То же самое можно сделать (а также удалить библиотеку классов в **Toolbar**) в настройках среды Visual FoxPro: **Tools | Options | Controls**. Здесь можно добавить свои классы, как и базовые классы VFP.

Создание элементов управления через *DataEnvironment*

Есть еще один способ создания элементов управления на форме. В режиме редактирования формы откройте окно **DataEnvironment**. В этом окне отображаются таблицы, которые являются источниками данных для элементов управления формы.

Найдите нужное поле в нужной таблице, отображенной в **DataEnvironment**. Установите на это поле курсор мыши, и, не отпуская кнопки, перетащите поле в область формы. Как только вы отпустите кнопку мыши, на форме будет создан объект, который ассоциируется с тем типом данных, который существует у выбранного поля таб-

лицы. Более того, у этого объекта будет выполнен ряд настроек, увязывающих этот элемент с указанным полем таблицы. Если на форму тащить не поле, а всю таблицу за ее заголовок, то будет создан элемент управления `Grid`, у которого в качестве источника данных будет указана эта таблица.

Ассоциация типа данных для каждого элемента выполняется в настройках Visual FoxPro: **Tool | Options | Field Mapping**, либо в дизайнере таблиц для конкретного поля в его свойствах **Map field type to classes**.

Венгерская нотация

В *главе 1* кратко упоминалось о "венгерской нотации". Это некий свод правил по именованию переменных, который также был распространен на именование объектов. Венгерскую нотацию изобрел программист Microsoft Чарльз Симони, венгр по национальности. Но названа она так не только по национальности изобретателя, но и по количеству согласных в наименовании, которое характерно для венгерского языка. Цель этой нотации — лаконичное и эффективное представление информации об объекте в его имени. Имя объекта в соответствии с данной нотацией имеет следующий формат:

```
[Префикс][ОсновноеИмя]
```

Префикс представляет собой сочетание из строчных согласных букв, характеризующих объект, например — `lbl` — этикетка или `frm` — форма.

За префиксом следует собственно имя, отображающее суть объекта (табл. 5.1).

Таблица 5.1. *Венгерская нотация для элементов управления*

Наименование объекта	Префикс	Пример
Checkbox	Chk	chkGotovo
ComboBox	Cbo	cboFirms
Command Button	Cmd	cmdOK
Form	Frm	frmInput
Image	Img	imgIzdel
Label	Lbl	lblHelpMessage
Line	Lin	linLine
ListBox	Lst	lstTractors
MDIForm	Mdi	mdiFormNew
Menu	Mnu	mnuMain
OLE	Ole	oleMyNewOLE
Option Group	Opt	optChoise

Picture	Pic	picNewDocu
Screen	Scr	scrMainScr
Shape	Shp	shpCircle
Text	Txt	txtLittleText
Timer	Tmr	tmrAlarm

Конечно, применение венгерской нотации делает программу более наглядной и удобной в работе, особенно если работает над ней коллектив программистов. Однако она не для всех программистов приемлема, поэтому соглашения предлагаются только в качестве рекомендаций. Если вас не устраивает какая-либо их часть, замените ее по своему усмотрению. Но помните о тех, кто будет работать с вашими программами, и не забывайте ставить в заголовке модулей комментарии, разъясняющие суть ваших изменений.

После этих маленьких, но необходимых отступлений вернемся к нашим объектам, т. е. элементам управления. Рассмотрим их поочередно.

Список элементов управления

Список элементов управления представлен в табл. 5.2.

Таблица 5.2. *Применение элементов управления*

Элемент управления	**Основное назначение**
CheckBox (Флажок)	Флажок. Применяется для отображения полей, которые могут принимать одно из двух значений. Предполагает ответ "Да" или "Нет" на вопрос, содержащийся в тексте объекта
ComboBox (Раскрывающийся список)	Служит для выбора одного элемента из предлагаемого набора значений. Не предназначен для добавления элементов в список или их редактирования. В случае необходимости добавлять или редактировать записи лучше использовать Grid
Command Button (Кнопка)	Применяется для управления пользователем определенной операцией или последовательностью операций
Command Group (Группа кнопок)	Объединение нескольких объектов Command Button в единый блок. Можно использовать для создания меню
Container (Контейнер)	Группировка нескольких разнородных объектов в единый блок (элемент управления)
Control (Управление)	Используется так же, как и Container, кроме того, блокирует прямой доступ к объектам, включенным в него
Custom (Пользовательский)	Организует хранение пользовательских процедур и значений как методы и свойства класса

EditBox (Редактируемое поле)	Применяется для ввода и редактирования символьных полей большого размера и memo-полей
Form (Форма)	Контейнер, в котором размещаются все остальные элементы управления
FormSet (Набор форм)	Контейнер контейнеров. Объединяет в себе несколько объектов Form
Grid (Сетка)	Применяется для просмотра и изменения табличных данных. Подробнее будет рассмотрена в *главе 9*
Hyper Link (Гиперссылка)	Содержит методы и свойства для управления внешним по отношению в VFP объектом Internet Explorer
Image (Изображение)	Служит для отображения графических объектов (рисунков, фотографий и т. д.)

Таблица 5.2 *(окончание)*

Элемент управления	**Основное назначение**
Label (Этикетка)	Служит для отображения статического (не изменяемого в процессе работы) текста — заголовков, поясняющей информации, надписей
Line (Линия)	Служит для отображения графического объекта — линии на форме
List Box (Список)	Применяется для отображения списка значений. Прямое редактирование списка значений невозможно
OLE Bound Control (Связанный элемент управления)	Связанный элемент управления OLE. Управление таким элементом из среды VFP практически невозможно. Подробнее будет рассмотрен в *главе 10*
OLE Control (OLE-контейнер)	OLE-контейнер. Встраивает в VFP компоненты ActiveX для реализации режимов управления, не предусмотренных стандартным набором элементов управления VFP. Подробнее будет рассмотрен в *главе 10*
Option Group (Группа переключателей)	Служит для выбора одного значения из предполагаемого набора значений. При этом выбранное значение всегда совпадает со значением, отображаемым в качестве выбранного значения
PageFrame (Блок страниц)	Блок страниц, может содержать до 99 вкладок. Каждая страница может содержать другие элементы управления
Separator (Разделитель)	Разделяет группы кнопок в элементе управления Toolbar
Shape (Фигура)	Отображает графический объект в виде прямоугольника, окружности или эллипса на форме
Spinner (Счетчик)	Служит для ввода числовых значений, которые могут изменяться в определенном диапазоне
TextBox (Текстовое поле)	Применяется для ввода и редактирования небольших объемов информации: отдельных чисел, дат, текста. При этом предполагается, что вся информация может уместиться в одну строку

Timer (Таймер)	Используется для запуска периодического выполнения определенных операций
ToolBar (Панель инструментов)	Используется как контейнер для элементов, управляющих формой, но при этом располагающихся вне формы

Каждый из этих объектов имеет свой собственный набор свойств, событий и методов. Справочник свойств, событий и методов каждого элемента управления приведен в *приложении 4*.

CheckBox (Флажок)

В стандартном режиме работы CheckBox выглядит как флажок (рис. 5.1).

Считается, что этот элемент управления может иметь только два состояния и применяется для логических переменных или полей таблицы, имеющих логические значения. Только не надо понимать это как два значения, например, как в данном случае: "разрешить" или "запретить". Это надо понимать, как некий признак, дающий ответ на вопрос — разрешить ли? Да или нет? То есть CheckBox — это ответ на вопрос, сформулированный в свойстве Caption объекта. Установленный флажок означает — да, разрешить, а неустановленный — нет, не разрешать.

Рис. 5.1. CheckBox

Из этого следует, что объект CheckBox, да и логические поля в общем случае, не следует применять в тех случаях, когда речь идет о выборе некоторого значения, пусть даже вам кажется, что таких значений может быть только два.

Например, если речь идет об указании пола человека, на первый взгляд кажется, что значений может быть всего два — мужчина или женщина. Использование CheckBox в этом случае будет означать не выбор между мужчиной и женщиной, а ответ на вопрос: является ли данный человек мужчиной? Получается подтверждение одного значения, а не выбор одного из двух значений. Более того, если ввод данных осуществляется на основании каких-либо документов, возможен третий вариант: "не знаю", поскольку просто не заполнена соответствующая графа, а определить пол по фамилии или имени человека не всегда возможно.

Впрочем, объект CheckBox может принимать и третье значение, и это именно значение "не знаю", значение .NULL. Визуально выглядит как серый квадратик с бледной галочкой.

У объекта CheckBox есть еще одна особенность. CheckBox со свойством Style = 1 — Grafical отображается в виде кнопки. В этом случае кнопка может быть нажата и отжата так, как изображено на рис. 5.2. Ответ "Да" выглядит как нажатая кнопка, а ответ "Нет" — как отжатая.

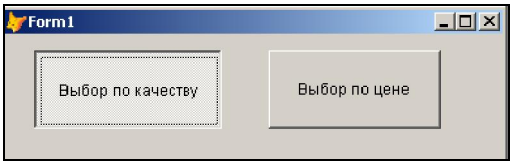

Рис. 5.2. Использование элемента `CheckBox` как кнопки

Надпись на переключателе определяется свойством `Caption`.

Пример использования `CheckBox` можно посмотреть на компакт-диске в \char5 \checkbox1.scx.

ComboBox (Комбинированный список)

`ComboBox` — это список, который в обычном состоянии свернут, но имеет кнопку для раскрытия списка. Несомненным плюсом `ComboBox` является его экономичность — в свернутом виде он занимает мало места на форме. При нажатии на кнопку список раскрывается, давая возможность выбрать из списка одно из значений.

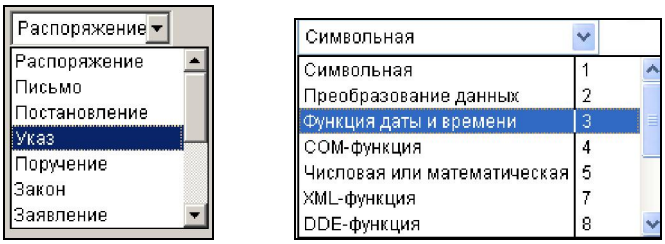

Рис. 5.3. `ComboBox`

Более того, данный объект не просто выбирает одно из значений списка, но и может выбирать значение одного столбца списка, а отображать в качестве выбранного значение другого столбца того же списка. Например, так, как изображено на рис. 5.3 — выбирается наименование, а записывается код. Полезный объект? Несомненно.

Выбранное значение записывается в свойство `Value`, а отображаемое — в свойство `Display Value`.

`ComboBox` не предназначен для ввода и редактирования элементов списка, он этого не умеет. Впрочем, можно и добавлять, и модицифировать, но это требует дополнительных усилий от разработчика, которые связаны с разработкой дополнительного кода. По умолчанию при настройке `Style=0` — `Dropdown Combo` — вы можете ввести в качестве выбранного значение, которого нет в списке. С точки зрения `ComboBox` это будет не новое значение, а неправильное, несуществующее значение. Точнее, это ключ для поиска значений, по которому не было найдено ни одного значения, соответственно, не удалось осуществить выбор, стало быть, выбрали нулевое (пустое) значение.

В результате такого выбора (событие Valid) значение свойства Value будет пустым, а значение DisplayValue — не пустым. Этим можно воспользоваться, чтобы добавить введенное значение к списку. Итак, имеются следующие предпосылки:

- отображаемое в ComboBox значение хранится в свойстве DisplayValue;
- выбранное значение хранится в свойстве Value;
- после завершения процесса выбора выполняется событие Valid().

Этими предпосылками можно воспользоваться для ввода нового значения. Сравнивая значение DisplayValue и Value, можно сделать вывод о том, что такого значения в списке нет, и записать в коде события Valid() следующий код (листинг 5.1).

```
IF !Empty(This.DisplayValue)  AND empty(This.Value)
**Пользователь ввел значение, которого нет в списке и ничего не выбрано
** Добавляем в источник данных новое значение
...
** Обновляем список
This.Requery()
ENDIF
```

Как именно добавить новое значение, зависит от того, что именно вы выбрали в свойстве RowSourceType. Если была выбрана таблица или курсор, можно использовать команду добавления записи, например, INSERT.

Если же источником данных является массив, то необходимо увеличить его размерность и вписать новое значение. Например, вот так:

```
cntMass=ALEN(mass)        && определяем размер массива
cntmass=cntmass+1         && увеличиваем счетчик
DECLARE mass(cntmass)     && увеличиваем размер массива
AINS(mass,ALEN(mass))     && добавляем элемент массива
STORE ALEN(mass) TO mass(ALEN(mass))
* заносим в массив значение, равное порядковому номеру
* последнего элемента (это для примера, можно занести любое
* другое)
```

Другим способом наполнения данными ComboBox может быть использование его методов AddItem и AddListItem.

AddItem	Метод	Добавляет новый элемент списка в элемент управления ComboBox, предоставляя при этом возможность задать индекс элемента
AddListItem	Метод	Добавляет новый элемент списка в элемент управления ComboBox, давая возможность задать идентификатор элемен-

		та

Некоторую сложность представляет наполнение списка вручную при помощи методов `AddItem()` и `AddListItem()`. Использование этих методов возможно, если в качестве типа источника данных указано `0 – None`.

`AddItem()` — добавляет новую строку в список, одновременно заполняя содержимое указанного столбца.

`AddListItem()` — заполняет содержимое отдельной ячейки строки списка.

Метод `AddItem()` добавляет "строку":

```
Control.AddItem(cItem [, nIndex] [, nColumn])
```

У строки возможны три параметра: текст, порядковый номер (индекс) и номер столбца, который указывает, куда именно нужно вставить текст. Таким образом, написав код

```
ThisForm.List1.AddItem("Текст",1,1)
```

мы даем задание добавить в `List Box` новую строку, в которой содержимое первого столбца будет иметь значение "`Текст`", и эту строку нужно установить как первую строку списка. Если второй параметр больше, чем общее количество строк в списке плюс 1, то будет выдано сообщение об ошибке адресации.

Метод `AddListItem()` добавляет "элемент списка". Это может быть модификация или добавление одной ячейки списка.

```
Control.AddListItem(cItem [, nItemID] [, nColumn])
```

Если для метода `AddItem()` речь идет о порядковом номере элемента в списке, о том порядке, в котором этот элемент отображается в списке, то в методе `AddListItem()` речь идет о внутреннем идентификаторе (номере) элемента списка, который не является порядковым номером. Этот идентификатор можно указывать явно в качестве второго параметра метода. Если будет указан номер, которого не существует в списке строк, то будет создана новая строка. Написав код

```
ThisForm.List1.AddListItem("Текст",1,2)
```

мы указываем, что нужно найти строку, внутренний идентификатор которой равен 1, и записать текст во второй столбец. Если строка с таким идентификатором не будет обнаружена, то будет создана новая строка с внутренним идентификатором 1. При этом текст записывается во второй столбец.

Каждая строка, добавляемая в `List Box`, имеет два присвоенных ему идентификационных номера:

- `nItemID`, целое число, соответствующее уникальному идентификатору (ID) строки списка в элементе управления. По умолчанию нумерация начинается с 1 и увеличивается при создании новых строк. Однако нет никаких ограничений на последовательность и порядок присвоения значения этого идентификатора;

- nIndex, целое число, соответствующее порядку, в котором строки отображаются в элементе управления: первой строке соответствует nIndex = 1.

Чтобы не потеряться при создании новых элементов списка, обратите внимание на свойства NewItemId и NewIndex. Эти свойства доступны только в Run-Time (в процессе исполнения) и только на чтение. Они содержат соответствующие значения последних (по времени) созданных строк списка.

Если вы хотите создавать источник данных для ComboBox программно, можно использовать AddItem() для добавления данных первого столбца, а затем AddListItem() для добавления данных второго столбца.

Принимайте решение о добавлении значений в выпадающий список, обдумав все последствия. Дело в том, что можно организовать создание нового значения, но невозможно определить, вводится новое значение или редактируется существующее. В результате можно получить не только большое количество дублирующихся записей, но и погрязнуть в бесконечных просьбах пользователей удалить случайно введенное значение или откорректировать существующее.

В качестве выбранного значения в элементе ComboBox *всегда отображается содержимое первого столбца списка*, повлиять на это невозможно. Но можно его "обмануть". Сформировав нужным образом значение первого столбца, укажите ему нулевую ширину в свойстве ColumnWidths. В результате первого столбца как бы нет (его не видно), но его содержимое отображается как выбранное. То, что реально будет выбрано, отображается в свойстве BoundColumn. Здесь указывается порядковый номер столбца раскрывающегося списка, значение которого будет указано и записано в свойство Value.

Ширина столбца ComboBox определяется свойством ColumnWidths. Однако если настройка ColumnCount=0, то размер раскрывающегося списка автоматически подгоняется под размер самого ComboBox и настройка ColumnWidths игнорируется.

ComboBox имеет несколько свойств, важных для понимания его работы. Такими свойствами, в частности, являются ControlSource и RowSource. У начинающих программистов часто возникает непонимание, что ControlSource отображает данные поля таблицы-источника или переменной, куда будет запоминаться выбранное значение, а RowSource — данные справочника, которые появятся в выпадающем списке. И это *разные* данные.

Приведем такой пример.

Имеется первая таблица "Сотрудники" (SOTR) с данными о работниках, включающая поле "код профессии" (ID_PROF).

Имеется таблица "Справочник профессий" PROF с полями "код" и "наименование профессии" (ID и NAIM).

Организована связь (Relation) по полю ID таблицы PROF с полем ID_PROF таблицы SOTR.

Задача состоит в том, чтобы обычный ComboBox показывал в выпадающем списке справочник профессий, и после выбора профессии из справочника записывал ее код в основную таблицу SOTR. Это обычная, тривиальная задача во многих программах.

Для того чтобы выполнить задание, сделаем следующие настройки у ComboBox (рис. 5.4):

1. Свойство ControlSource указывает на поле или переменную (в нашем случае поле таблицы), принимающее данные из ComboBox.
2. Свойство RowSource указывает поля, которые появятся в выпадающем списке ComboBox. В списке полей на первом месте должно стоять то, что надо отображать как выбранный элемент (DisplayValue). В данном случае — это второе поле — NAIM. Нужно всегда помнить о том, что в качестве DisplayValue выводится содержимое первого столбца RowSource.
3. Свойство RowSourceType устанавливаем равным 6 – Fileds, поскольку надо отобразить поля не в естественной последовательности. Если поля следуют в естественной последовательности, можно установить значение свойства равным 2 – Alias.
4. Свойство BoundColumn определяет, какая колонка многоколоночного элемента ComboBox привязана к свойству Value управляющего элемента. В данном случае это колонка 1.

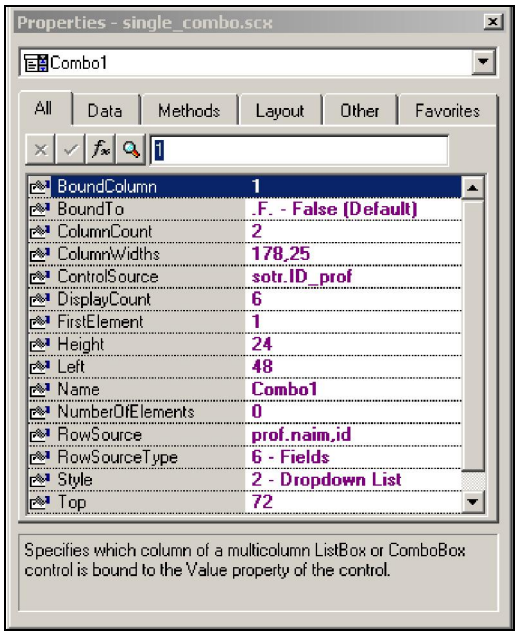

Рис. 5.4. Настройка свойств ComboBox

5. Свойство BoundTo устанавливается в .F. Это означает, что по умолчанию значение свойства Value определяется типом данных переменной или поля, указанным в свойстве ControlSource.
6. Свойство ColumnCount устанавливаем равным 2. Это значит, что в выпадающем списке будет отображаться два столбца — NAIM и ID.
7. Свойство DisplayCount указывает, сколько строк будет содержать выпадающий список. В нашем случае — 6.
8. Свойство FirstElement доступно только в случае, если RowSource представлено массивом (RowSourceType = 5). Заметьте, что свойство FirstElement применяется к указанным управляющим элементам только в случае, если они содержат одну колонку. В нашем случае оно особой роли не играет и применяется по умолчанию.
9. Свойство NumberOfElement доступно только в том случае, когда свойство ListSourceType установлено в значение 5 (Array) и свойство ColumnCount установлено в значение, равное 1. NumberOfElements полезно для ограничения числа элементов массива, которые могут быть отображены в списке.
10. Пример простого ComboBox на форме вы можете посмотреть на CD в CHAR5\Combo\Single_combo.scx.

Давайте рассмотрим подробнее заполнение выпадающего списка. Типы источника значений выпадающего списка задаются в свойстве RowSourceType (табл. 5.3).

Таблица 5.3. *Описание типов источника значений для свойства RowSource*

Наименование	Описание
0 — None (Тип не определен) — по умолчанию	Источник данных для выпадающего списка не определен (возможно, будет определен программно)
1 — Value (Значение)	В качестве источника данных используется список конкретных значений, например — январь, февраль,... декабрь. Если в свойстве Value записать строку "июль", например, то при запуске формы будет выбрано значение "июль"
2 — Alias (Псевдоним таблицы)	Используется псевдоним таблицы, например, PROF
3 — SQL Statement (SQL-выражение)	Если вы устанавливаете RowSourceType=3, то в RowSource вы должны определить команду Select-SQL, обязательно включив опцию назначения, например, INTO TABLE или INTO CURSOR
4 — Query (.QPR) (Запрос)	В этом случае в свойстве RowSource вы должны указать имя файла запроса с расширением qpr
5 — Array (Массив)	В RowSource укажите имя массива
6 — Fileds (Поля таблицы)	В RowSource необходимо указать перечень необходимых полей таблицы через запятую

7 — Files (Файлы)	В RowSource указываются файлы по маске, например, *.* или *.zch
8 — Structure (Структура полей таблицы)	В RowSource нужно указать имя таблицы, структуру которой вы хотите показать в выпадающем списке
9 — Popup (Меню) — включено для совместимости с предыдущими версиями	
10 — Collection (Объект Collection)	Вы можете заполнить combo членами класса и значениями свойств объектов в коллекции, или объекта Collection

Если у вас есть уверенность в том, что данные, представленные в выпадающем списке, могут измениться во время работы, то можно использовать метод Requery для того, чтобы "освежить" выпадающий список, чтобы ваш ComboBox содержал самые актуальные данные. Метод Requery повторно проверяет свойство RowSource и обновляет список новыми значениями.

Свойство BoundTo для ComboBox может принимать два значения — "истина" и "ложь" и определяет, будет ли значение свойства Value элемента управления ComboBox определяться свойствами List или ListIndex.. При BoundTo=.T. значение свойства Value определяется значением свойства List. При BoundTo=.F. значение свойства определяется типом данных переменной или поля, указанным в свойстве ControlSource.

В качестве выбранного значения может быть использовано либо содержимое элемента списка, либо порядковый номер элемента списка. Содержимое элемента списка может быть только символьного типа. Однако тип данных поля таблицы, куда записывается выбранное значение, может быть как символьного, так и числового типа.

Если поле таблицы символьного типа, то в качестве выбранного значения всегда используется содержимое элемента списка. При этом значение свойства Value тоже будет иметь символьный тип данных.

Если поле таблицы числового типа, то по умолчанию в качестве выбранного значения будет использован порядковый номер элемента в раскрывающемся списке. Если необходимо в качестве выбранного значения использовать содержимое выбранного элемента списка, то в этом случае необходимо сделать дополнительную настройку

Bound=.T.

В этом случае содержимое выбранного элемента списка будет автоматически конвертировано в число в момент выбора, и результат этой конвертации будет записан в Value и поле, указанное в ControlSource.

При использовании ComboBox можно столкнуться с некоторыми проблемами, которые, впрочем, можно обойти. Например, при установке фокуса в объект ComboBox пропадает выбранное значение списка. При переходе на другой объект отображение выбранного элемента списка восстанавливается. Проблема возникает, если выполняются следующие условия:

- в качестве типа источника данных для ComboBox указаны поля таблицы, т. е. RowSourceType=6 — Fields;
- в качестве поля, содержащего возвращаемое значение, указано поле числового типа.

Для решения проблемы следует изменить тип источника данных для ComboBox, т. е. выбрать что-нибудь другое, кроме RowSourceType=6 — Fields

Использование в качестве источника данных *Select-SQL*

В качестве источника данных для ComboBox допустим курсор, созданный с помощью оператора Select-SQL.

```
RowSource='SELECT Name,ID FROM MyTab INTO CURSOR curCombo'
RowSourceType=3 — SQL Statment
```

В предложении Select-SQL указание опции INTO обязательно, поскольку в противном случае после выполнения запроса будет отображено BROWSE-окно с результатами выполнения запроса.

Для порядка, не забудьте при закрытии формы закрыть и временный курсор с именем curCombo.

Использование в качестве источника данных массива

В этом случае вам необходимо предварительно создать свойство формы типа "массив". Признаком того, что новое свойство является именно массивом, является указание его размерности, например:

```
ThisForm.AddProperty("aCombo[1,1]")
```

Теперь где-то в событии Init-формы следует наполнить этот массив

```
SELECT Name,ID FROM MyTab INTO ARRAY ThisForm.aCombo
```

И сделайте следующие настройки в ComboBox

```
RowSource='ThisForm.aCombo'
RowSourceType=5 — Array
```

На CD-диске в каталоге CHAR5\Combobox находятся примеры работы с ComboBox.

Command Button (Кнопка)

Кнопки — очень распространенный элемент управления (рис. 5.5). При нажатии кнопки выполняется некоторое запрограммированное действие и выполняется событие Click() кнопки, в нем указываются те действия, которые необходимо выполнить.

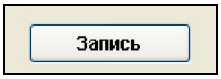

Рис. 5.5. Command Button

Заголовок кнопки задает свойство Caption. Если текст не помещается в одну строку, то, используя свойство WordWrap=.T., можно организовать размещение текста в несколько строк. Хотя размещения на кнопке длинных надписей лучше избегать. Используйте в случае необходимости всплывающие подсказки в **ToolTips**.

На кнопку можно добавить картинку, задав нужное значение в свойстве Picture. Используйте свойство PicturePosition, чтобы расположить картинку и текст кнопки в нужном положении относительно друг друга.

У кнопки имеется свойство Cancel. Если установить его в значение .T., то при нажатии пользователем клавиши <Escape> будет выполнено автоматическое нажатие на эту кнопку. Это удобно для организации закрытия формы по нажатию кнопки <Escape>. Одновременно на форме может быть только одна кнопка со свойством Cancel=.T. , иначе просто невозможно сказать, какая именно из кнопок нажата при использовании клавиши <Escape>.

Свойство кнопки Default удобно использовать для организации поиска. Если установить Default=.T., то при нажатии клавиши <Enter> или <Ctrl>+<Enter> (в зависимости от того, в каком объекте находится фокус) будет выполнено автоматическое нажатие на эту кнопку, т. е. запущен поиск.

В некоторых случаях необходимо организовать "залипание" кнопки. Это значит, что при нажатии кнопка должна остаться в нажатом состоянии. Отжать кнопку в этом случае можно при повторном нажатии. Этого эффекта можно достичь при использовании другого объекта — CheckBox с установленным свойством Style= 1 – Grafical.

Опишем применение кнопок на примере, например, мы хотим напечатать какой-то отчет, и для этого нажимаем кнопку. Чтобы при нажатии кнопки было ясно, какое действие она производит, целесообразно нанести на нее надпись "Печать отчета". Изменим значение свойства Caption кнопки, которая по умолчанию называется "Command1", на "Печать отчета". Кнопки позволяют разместить и небольшие изображения, поэтому поместим рядом с надписью изображение принтера. Воспользуемся для этого свойством Picture и укажем в нем место нахождения нашего изображения принтера. Если название вашей кнопки длинное, придется установить свойство WordWrap=.T., чтобы стал возможным перенос слов.

А в событии Click разместим программный код печати готового отчета на принтер:

```
REPORT FORM namefile1 TO PRINTER.
```

Это, так сказать, лишь эскиз, на самом деле программный код будет несколько сложнее, поскольку надо бы проверить, а есть ли в сети такой принтер и готов ли он к работе.

Пример создания кнопки смотрите на компакт-диске char5\knopka_pechat.scx.

Command Group (Группа кнопок)

Command Group является контейнером, в котором размещается не одна кнопка, а несколько (рис. 5.6). Но, хотя этот объект и является контейнером, он весьма специфичен, поскольку позволяет добавлять внутрь себя только и исключительно объекты Command Button, причем только через изменение свойства ButtonCount. Значение свойства ButtonCount определяет количество кнопок.

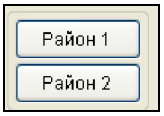

Рис. 5.6. Command Group

Каждая кнопка обрабатывает свои события, благодаря этому Command Button можно использовать для создания меню.

Элемент CommandGroup удобно использовать для организации дополнительного блока управления на форме. Кнопки, расположенные на форме "врассыпную", очень неудобны для пользователя. Гораздо удобнее собрать все кнопки в одном месте.

Если вам необходимо определить, какая же кнопка была нажата, обратитесь к свойству CommandGroup.Value, которое хранит номер нажатой кнопки.

Если вы хотите, чтобы вставляемые кнопки были созданы не на основе базового класса VFP, а на основе вашего собственного класса, используйте свойства MemberClass и MemberClassLibrary, чтобы указать свой собственный класс и ту библиотеку классов, в которой он находится.

Пример организации меню на базе группы кнопок можно посмотреть на CD в \char5\Commandbutton\knopki_menu.scx.

Container (Контейнер)

Container — это элемент управления, который сам может содержать любые элементы управления (рис. 5.7).

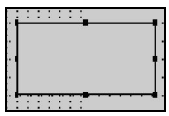

Рис. 5.7. Container

Если вы проектируете форму "с нуля", он вряд ли вам пригодится. Его назначение — это объединение элементов управления при разработке собственных классов. Вне идеологии классов его использование тоже допустимо, но необязательно. Если вам просто необходима "рамочка", используйте лучше объекты Shape или Line.

Control

Объект `Control` очень похож на объект `Container`. Он также объединяет внутри себя разные другие элементы управления. Однако его принципиальное отличие от объекта `Container` заключается в том, что он блокирует доступ извне ко всем объектам, находящимся у него внутри.

Например, если вы включили объект `TextBox` в объект `Container`, вы можете обратиться к нему из события `Init()` формы:

```
ThisForm.Container1.TextBox1.Value=1
```

Но если вы включили объект `TextBox` в объект `Control`, то обращение

```
ThisForm.Control1.TextBox1.Value=1
```

вызовет сообщение об ошибке, что ваш `TextBox`, оказывается, не существует! Объект `Control` "спрятал" все доверенные ему объекты. Общение с внешним миром организуется только через сам `Control` и его методы.

Custom

Объект `Custom` используют в случае, когда процедуры или свойства невозможно определить в рамках имеющихся элементов управления. В этом случае используют `Custom`. В нем создаются все необходимые методы и свойства, затем экземпляр созданного класса размещается на форме. При необходимости происходит обращение к созданным методам и свойствам данного объекта. `Custom` — это инструмент для разработки собственной библиотеки классов.

EditBox (Редактируемое поле)

Поле `EditBox` предназначено для просмотра и редактирования текстовых полей большого размера, содержимого memo-полей, переменных, массивов элементов и полей типа `Character`. Как правило, в любой системе управления есть необходимость просмотра и изменения текста неопределенной длины, например, краткое содержание документа.

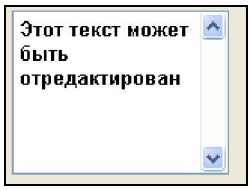

Рис. 5.8. `EditBox`

Особенностью данного элемента является наличие вертикальной полосы прокрутки. Если количество строк не помещается в видимом окне просмотра, можно прокрутить текст вниз для нужной строки. Горизонтальной полосы прокрутки элемент не имеет.

Если строка не умещается в видимом окне просмотра по горизонтали, то она автоматически переводится на следующую строку либо по пробелам, либо, если пробелов в строке нет, по уместившимся символам.

Следует учитывать тот факт, что при нажатии клавиши <Enter> внутри этого объекта вы не переходите на другой объект, а создаете новую строку. Это означает, что в текст будут записаны два служебных символа — с кодом ASCII=13 и ASCII=10.

Если вы редактируете в `EditBox` memo-поле, то это не имеет особого значения. Если же вы редактируете в нем обычное символьное поле таблицы, то эту особенность надо учитывать. Хотя символы эти и невидимые, но невидимы они только внутри объекта `EditBox`, а при просмотре через `Browse` они очень даже видимы и отображаются как пара непонятных значков в тексте. Именно по этой причине желательно избегать редактирования в `EditBox` простых символьных полей.

Лучше применять `EditBox` для редактирования memo-полей или текста, не привязанного к полю таблицы. Поля типа `Blob` позволяется лишь просматривать.

Свойство `ControlSource` позволяет связать `EditBox` с полем таблицы. В случае, если `TextBox` нежелательно связывать с полем таблицы, вы можете связать его с переменной.

Пример создания `EditBox` смотрите на CD-диске \char5\editbox_in_form.scx.

Form

Форма тоже может быть отнесена к элементам управления. Подробнее о форме *см. главу 4*.

Form Set

Объект `FormSet` можно назвать контейнером для форм. Он не имеет визуального отображения, но объединяет в себе несколько объектов — контейнеров — форм. Такое объединение форм позволяет передать ряд свойств форм объекту-контейнеру.

Во-первых, контейнеру делегируется свойство `WindowType`. Это значит, что модальной становится не сама форма, а весь набор форм. Формы внутри `FormSet` действительно становятся модальными в том смысле, что, не закрыв эти формы, невозможно обратиться ни к одному ранее открытому объекту или пункту меню. Но переключаться между формами можно без ограничений.

Для форм, включенных в `FormSet`, их личное свойство `WindowType` просто игнорируется, оно делегируется объекту `FormSet`.

Все формы, включенные в `FormSet`, имеют общее окружение. Они имеют одну общую `DataEnvironment`, работают с одной и той же `DataSession`.

Поскольку объект `FormSet` не имеет никакого визуального воплощения, то работа с ним происходит несколько не так, как с обычными элементами управления.

Прежде всего, собственно создание `FormSet` начинается с создания формы. Затем созданная форма трансформируется в `FormSet` через пункт системного меню **Form | Create FormSet**. Новые формы добавляются в созданный объект `FormSet` либо через пункт системного меню **Form | Add New Form**, либо перетаскиванием формы из соответствующей библиотеки классов.

Чтобы создать новый объект `FormSet` на основе ранее созданного класса `FormSet`, используют общие настройки Visual FoxPro через пункт меню **Tools | Options | Forms | Form set**. После выбора соответствующего класса стандартный режим создания новой формы приведет к созданию `FormSet` на основе указанного класса.

Grid

Объект `Grid` (рис. 5.9) применяется для просмотра и модификации табличных данных. Подчеркнем — только табличных! Курсор — это тоже таблица, только временная.

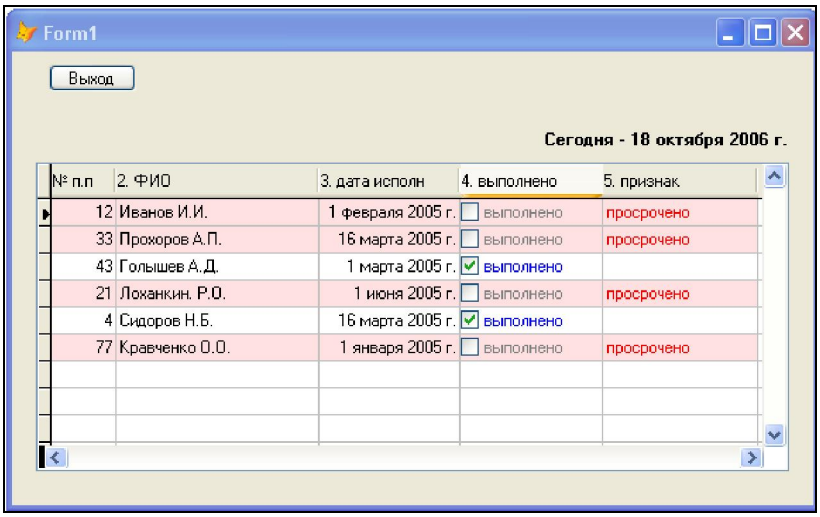

Рис. 5.9. `Grid`

Ничто другое не может служить источником данных для `Grid`. Более подробно об этом объекте читайте в *главе 9*.

HyperLink (Ссылка)

Буквальный перевод "гиперссылка" или "сверхссылка". На самом деле этот объект очень похож на объект `Custom`. Он не имеет никакого визуального отображения в процессе работы формы. Цель этого объекта — управление внешним по отношению к Visual FoxPro приложением Microsoft Internet Explorer. Он создает объект, содержа-

щий ссылку на страницу Интернета, обеспечивает возможности навигации для приложений Visual FoxPro в Internet Explorer. Отметим, что объект `Hyperlink` поддерживается только в Internet Explorer.

У `HyperLink` есть ряд специфических методов:

- `NavigateTo` — запускает приложение Microsoft Internet Explorer и открывает страницу по переданной в качестве параметра ссылке;
- `GoBack` — симулирует нажатие кнопки "Назад" в открытом приложении Microsoft Internet Explorer;
- `GoForward` — симулирует нажатие кнопки "Вперед" в открытом приложении Microsoft Internet Explorer.

Чтобы отобразить на форме некоторую ссылку в том виде, как это делается на страницах Интернета, прибегают к разным трюкам. Как правило, это просто объект `Label` с некоторыми настройками, но это могут быть также кнопки или графическое изображение. Нажатие на этот объект вызывает срабатывание события `Click`, в котором и делается вызов соответствующего метода объекта `HyperLink`.

Пример использования гиперссылки вы найдете в форме

CHAR5\hyper.scx.

Image (Графический объект)

Этот элемент управления предназначен для размещения на форме графического изображения, которое контролируется динамически в процессе работы программы (рис. 5.10).

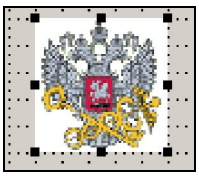

Рис. 5.10. `Image`

Вы можете указать либо имя файла в свойстве `Picture`, либо бинарные данные в свойстве `PictureVal` как источник, содержащий графическую информацию. Это может быть не только статическая картинка, но и динамическое изображение в формате GIF или GFA.

Как правило, физический размер картинки не совпадает с выделенной под картинку областью формы, т. е. не совпадает с размерами объекта `Image`. В этом случае интересно свойство `Stretch`, которое определяет, как изменяются размеры изображения с тем, чтобы поместиться внутри элемента управления. По умолчанию это свойство установлено равным 0, это означает, что размеры картинки обрезаются, чтобы она поместилась в заданный прямоугольник. Если `Stretch=1`, то размер изображения ме-

няется, с тем, чтобы поместиться внутри заданного прямоугольника, с сохранением пропорций. В случае, когда `Stretch=3`, картинка изменяется в размерах, чтобы вместиться в элемент управления, но ее оригинальные пропорции не соблюдаются.

Предполагается, что файл картинки должен быть указан на этапе проектирования формы. Однако, как и со многими другими объектами, можно указать нужный файл в процессе работы формы. Это удобно, например, при просмотре карточек сотрудника. На форме есть список сотрудников и один объект `Image`, у которого значение свойства `Picture` меняется при перемещении по списку сотрудников.

Изображение может иметь рамку (свойство `BorderStyle`), может быть повернуто на 90, 180 или 270 градусов.

Label (Этикетка)

Объект `Label` *(Этикетка)* предназначен для вывода на экран статического текста либо в виде заголовка, либо в виде поясняющей информации. Например, так (рис. 5.11).

Рис. 5.11. `Label`

Вероятно, вы заполняли самые разнообразные бланки и анкеты. Все они построены по одному принципу. Есть какое-то обозначение очередного пункта бланка и оставлено пустое место для того, чтобы вы что-то записали. В основном, формы строятся по тому же принципу. И это самое "обозначение очередного пункта" как раз и осуществляется при помощи объекта `Label`. Он отображает некий "заголовок" — текст, который не меняется в процессе работы с формой. Впрочем, в случае необходимости, этот текст может быть заменен и в процессе работы формы. Например, если дать в событии Init-формы такую команду:

```
ThisForm.Label1.Caption = "Сегодня – " + DTOC(Date())
```

Если введенный текст не умещается в одну строку, то установите свойство `WordWrap = .T.` для автоматического переноса текста на следующую строку. Выровнять текст внутри выделенного размера можно при помощи свойства `Alignment`. Определите размер и вид шрифта, используйте для этого свойства `FontName` и `FontSize`. Установите нужный цвет фона при помощи свойства `"BackColor"` и цвет текста внутри этикетки при помощи свойства `ForeColor`. Для выбора цвета можно дважды щелкнуть мышью в поле свойства `BackColor` и `ForeColor` — появится цветовая палитра, из которой можно выбрать нужный цвет. Если же цветов палитры вам недостаточно, используйте табл. 5.4.

Три числа, разделяемые запятыми, определяют цвет. По умолчанию установлен серый цвет — 212, 208, 200. Первая цифра означает количество красного цвета в цветовой палитре, вторая — количество зеленого, и третья — количество синего. Мак-

симальное значение каждого числа — 255. Изменяя числа, получаем различные цвета.

Таблица 5.4. *Цветовая палитра (наиболее популярные цвета)*

Цвет	Значение функции RGB	Значение Color
Белый (White)	255, 255, 255	16777215
Черный (Black)	0, 0, 0	0
Ярко-красный (Red)	255, 0, 0	255
Ярко-зеленый (Green)	0, 255, 0	65280
Ярко-синий (Blue)	0, 0, 255	16711680
Ярко-желтый (Yellow)	255, 255, 0	65535
Ярко-голубой (Cyan)	0, 255, 255	16776960
Ярко-малиновый (Magenta)	255, 0, 255	16711935
Темно-красный(Dark Red)	128,0,0	128
Темно-зеленый (Dark Green)	0,128,0	32768
Темно-синий (Dark Blue)	0,0,128	8388608
Темно-голубой (Dark Cyan)	0,128,128	8421376
Фиолетовый (Dark Magenta)	128,0,128	8388736
Темно-желтый (Dark Yellow)	128,128,0	32896
Темно-серый (Dark Grey)	128,128,128	8421504
Серый (Grey)	192,192,192	12632256

Пример создания этикетки смотрите на CD в \CHAR5\Label\.

Line (Линия)

Элемент `Line` (Линия) позволяет отображать на экране линии, а также с помощью них прямоугольники и другие фигуры (рис. 5.12). С помощью `Line` можно визуально отделять одну группу текстов от другой.

Наклон линий можно изменять программным путем. Для горизонтальных и вертикальных линий лучше использовать элемент управления `Shape`, потому что `Line` несколько сложнее позиционировать.

Пример использования `Line` на форме можно посмотреть в CHAR5\Line.

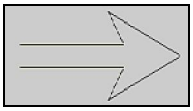

Рис. 5.12. `Line`

List Box (Список)

`List Box` выводит на экран список перечисленных в нем элементов и позволяет выбрать один из них. Он содержит определенное количество элементов списка, которые могут представлять собой элементы массива, список файлов, поля таблицы, пункты меню (рис. 5.13).

Рис. 5.13. `List Box`

Данный объект используется исключительно для отображения некоторого списка значений, прямое изменение этого списка невозможно. Это одна из причин, почему данный объект достаточно редко используется.

Прежде всего, следует понять, что содержимое объекта `List Box` — это не исходные данные. Содержимое объекта `List Box` — это коллекция элементов. Причем содержимое этой коллекции только и исключительно символьного типа.

Это значит, что, указывая для `List Box` тип источника данных в свойстве `RowSurceType`, вы, по сути, указываете способ наполнения коллекции элементов в объекте `List Box`, откуда брать значения для формирования элементов. Но это вовсе не означает, что в объекте `List Box` будут отображены сами данные, указанные в свойстве `RowSource`. В `List Box` будут отображены уже собственные данные. Они могут быть связаны с исходными данными, но могут и не быть, в зависимости от типа этих исходных данных.

Типы исходных данных для наполнения `List Box` приведены в табл. 5.3 (`ComboBox`). Методы добавления строки `AddItem()` и `AddListItem()` описаны там же.

OLEBound Control

Связанный элемент управления OLE. Управление таким элементом из среды FoxPro практически невозможно. Подробнее будет рассмотрен в *главе 10*.

OLE Control

OLE-контейнер, встраивает в Visual FoxPro компоненты ActiveX для реализации режимов управления, не предусмотренных стандартным набором элементов управления VFP. Подробнее будет рассмотрен в *главе 10*.

Option Group (Группа переключателей)

`Option Group` — это контейнер, который содержит группу переключателей, из которых может быть выбран только один. При этом выбранное значение всегда совпадает со значением, отображенным в качестве выбранного значения.

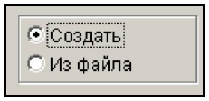

Рис. 5.14. `Option Group`

Объект `Option Group` похож на объект `Command Group` в том смысле, что также является контейнером для одного определенного типа объектов. В данном случае, для объекта `Option Button`.

Однако если объект `Command Button` имеет смысл и "сам по себе", т. е. вне `Command Group`, то объект `Option Button` не имеет смысла вне `Option Group` просто по самой своей природе.

Логика работы объекта `Option Group` предполагает выбор одного значения из предложенного набора. А какой смысл в отдельном значении без возможности выбора? Хотя вы можете создать собственный класс объекта `Option Button` и положить его на форму вне объекта `Option Group`, но особого смысла в этом нет. Он будет работать как объект `CheckBox`.

Количество переключателей в объекте `Option Group` регулируется настройкой `ButtonCount`, и никак иначе. Вы не можете вставить в этот контейнер никакой объект стандартным способом.

Если вы хотите, чтобы вставляемые переключатели были созданы не на основе базового класса FoxPro, а на основе вашего собственного класса, то используйте свойства `MemberClass` и `MemberClassLibrary`, чтобы указать свой собственный класс и ту библиотеку классов, в которой он находится.

По умолчанию в качестве выбранного значения в свойстве `Value` указывается порядковый номер выбранного переключателя. Однако можно использовать и символьное

значение. В этом случае в качестве выбранного значения будет храниться текст выбранного элемента.

Пример, содержащий группу переключателей, смотрите на CD в CHAR5\optgr_in_form.scx.

PageFrame (Набор страниц)

Этот элемент управления представляет собой многостраничную форму в виде пачки страниц, имеющих вкладки. Каждая страница может содержать другие элементы управления (рис. 5.15).

Рис. 5.15. `PageFrame`

Форма может иметь до 99 страниц, количество страниц определяется свойством `PageCount`. `PageFrame` часто используются для того, чтобы на первой странице размещать основную информацию, а на остальных — дополнительную. Набор вкладок можно использовать там, где требуется разместить большое количество данных на ограниченном пространстве, `PageFrame` позволяет экономить место на форме, при этом на одном и том же.

Shape (Контур)

`Shape` позволяет изображать на экране прямоугольники, круги или эллипсы (рис. 5.16).

Используя пункт главного меню **Format | Send To Back**, можно нарисовать рамку для уже существующих объектов. Это может быть полезным в случае, если вы уже наполнили форму объектами, а потом решили выделить их рамкой.

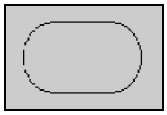

Рис. 5.16. Shape

Spinner (Счетчик)

Чтобы ввести числовые значения в определенном диапазоне, используют **Spinner** (рис. 5.17).

Следует учитывать, что Spinner работает только с целыми числами.

Рис. 5.17. Spinner

Счетчик позволяет увеличивать и уменьшать вводимое значение. При превышении границы диапазона выдается сообщение о допустимом диапазоне для вводимого числа.

За диапазон значений Spinner "отвечают" два свойства:

- SpinnerHighValue — представляет собой верхнюю границу диапазона (до 2147483647.00);
- SpinnerLowValue — представляет собой нижнюю границу диапазона (до –2147483647.00).

TextBox (Текстовое поле)

Текстовое поле, в которое можно вводить любую информацию – цифры, даты или текст – неотъемлемая часть многих работающих форм (рис. 5.18). В Visual FoxPro TextBox позволяет вводить только одну строку символов. Если вам требуется больше одной строки, используйте Edit Box.

Рис. 5.18. TextBox

Если данные для текстового поля берутся из таблицы, в первую очередь необходимо связать текстовое поле с таблицей. Воспользуемся свойством ControlSource и выберем из списка то поле, которое необходимо добавить в форму. Если же вы по каким-либо причинам не хотите связывать TextBox с таблицей, можно связывать поле с переменной, предварительно присвоив ей нужное значение (можно пустое). По умолча-

нию вам будет предоставлена для ввода символьная строка. Введенное значение будет храниться в свойстве `Value`. Свойство `Value` может содержать символьную строку, число, дату, или комбинацию дата-время, или логическое значение, которое является текущим в элементе управления. Свойство `Value` изменяет поведение, когда для элемента управления указан источник `ControlSource`. Когда источник для элемента управления установлен, свойство `Value` элемента управления представляет собой тип данных переменной или поля, указанного в свойстве `ControlSource`. Если тип данных не является действительным для данного элемента управления, то Visual FoxPro сгенерирует ошибку.

Настроив `FontName` и `FontSize`, определим нужный вид и размер шрифта.

Используем свойство `Alignment`, чтобы выровнять текстовую информацию в поле (табл. 5.5).

Таблица 5.5. *Допустимые значения свойства* `Alignment`

Значение	Описание
`0 — Left`	Выравнивает текст влево
`1 — Right`	Выравнивает текст вправо
`2 — Center`	Выравнивает текст по центру поля
`3 — Automatic (Default)`	Выравнивает текст в соответствии с типом данных источника: числовые данные выравниваются вправо, остальные — влево

Применяя управляющий элемент `TextBox`, вы получите полноценные возможности редактирования. Управляющий элемент `TextBox` позволяет в любом месте строки вставить символы, вырезать и вставить текст (с помощью комбинаций клавиш <Ctrl>+<C> и <Ctrl>+<V> соответственно).

Можно использовать свойство `Comment`, чтобы внести комментарии с кратким описанием каждого объекта. Это неактуально, если проект один и небольшой, а если большой, и не один — придется писать. Иначе, откладывая один проект, чтобы заняться другим, и возвращаясь затем к первому, вы уже не вспомните, что за поля вы ввели в форму и с какой целью.

При проектировании форм с `TextBox` часто необходимо ограничить ввод только цифрами, или только символами, т. е. при вводе необходимо задать определенный формат. Для форматирования данных поля ввода используется свойство `Format`. Свойство `Format` "отвечает" за форматирование поля в целом, а свойство `InputMask` — за форматирование каждого символа в отдельности, поскольку `InputMask` определяет шаблон ввода данных в поле. Допустимые шаблоны приведены в табл. 5.7.

В табл. 5.6 приведены допустимые значения свойства `Format`.

Таблица 5.6. *Допустимые значения свойства* `Format`

Опция	Описание
!	Преобразует буквы в прописные
$	Выводит денежный символ
^	Отображает число в экспоненциальном виде. Свойство ControlSource должно быть определено как число
A	Разрешает ввод буквенных символов без пробелов или знаков пунктуации
D	Использует текущий формат SET DATE
E	Выводит дату в формате British Date (выглядит как 31/12/05 07:35:20 PM)

Таблица 5.6 *(окончание)*

Опция	Описание
F	Не разрешает ввод ведущих пробелов в значения, определенные Varchar, и ведущих нулей в значения, определенные Varbinary
K	Выделяет поле целиком при наведении на него курсора
L	Высвечивает ведущие нули для числовых данных. ControlSource должно быть определено как числовое
M	Включено для обратной совместимости, не используйте в новых разработках
R	Отображает данные в формате маски, определенной в InputMask, для данных в формате Character и Numeric. Например, если маска для ввода телефона выглядит как 999-99-99, то число 1234567 будет выглядеть как 123-45-67
T	Удаляет ведущие и конечные пробелы
YS	Отображает дату в коротком формате
YL	Отображает дату в полном формате
Z	Отображает нули пробелами, если объект не выбран

Таблица 5.7. *Допустимые значения свойства* `InputMask`

Значение свойства	Описание
!	Переводит строчные буквы в прописные
#	Разрешает вхождение цифр, пробелов и знаков, например знак "Минус"
$	Показывает текущий денежный символ
$$	Выводит на экран плавающий денежный символ
,	Разрешает использование запятой для разделения целой и дробной части числа
.	Разрешает использование точки для разделения целой и дробной части числа

9	Разрешается ввод только цифр и знака числа
A	Разрешается ввод только буквенных символов
H	Предотвращает ввод не-шестнадцатеричных символов в определенные позиции
L	Разрешается ввод только логических символов
N	Разрешены только буквы и цифры
U	Разрешается ввод только буквенных символов с переводом в заглавные буквы (A-Z)
W	Разрешается ввод только букв с переводом их в строчные (a-z)
X	Разрешаются любые символы
Y	Разрешается ввод букв y,Y и N,n для логических значений `.T.` и `.F.`

Для размещения поля ввода в форме существует специальное средство — построитель.

Чтобы использовать построитель поля ввода, нужно выполнить следующие действия.

Нажать правую кнопку мыши и выбрать из контекстного меню пункт `Builder`. На экране вы увидите окно с тремя вкладками, в котором вы можете задать источник данных, стиль оформления и формат поля ввода (рис. 5.19).

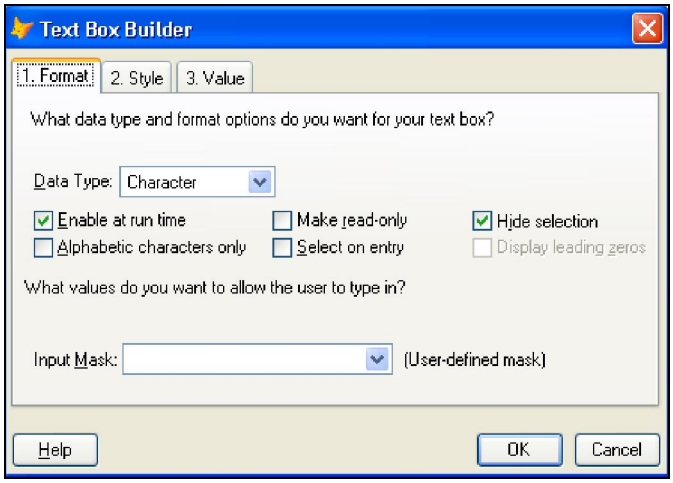

Рис. 5.19. Окно построителя текстового поля

Вкладка **Format** имеет 6 флажков, значения которых приведены в табл. 5.8.

Таблица 5.8. *Параметры вкладки* ***Format*** *построителя текстового поля*

Опция	Описание
Enable at run time	Устанавливает свойство `Enabled` в `.T.` Разрешает доступ к полю при запуске формы
Alphabetic characters only	Разрешает ввод только буквенных символов
Make read-only	Запрещает ввод данных в поле
Select on entry	При перемещении курсора в поле выделяет все поле целиком
Hide selection	Определяет свойство `HideSelections`, не выделяет или выделяет текст, не имеющий фокуса
Display leading zeros	Отображает или не отображает ведущие нули в поле ввода

Вкладка **Style** задает стиль оформления поля ввода. В табл. 5.9 приведен перечень настраиваемых параметров.

Таблица 5.9. *Параметры вкладки* ***Style*** *построителя текстового поля*

Опция	Описание
Special Effect	Может принимать значения `3D` или `Plain`. (3D — "утопленная" вглубь кнопка, `Plain` — ровная). Определяет свойство `SpecialEffect Text Box`
Border	Определяет рамку текстового поля, задает свойство `BorderStyle`
Character alignment	Задает тип выравнивания
Size text box to fit	Автоматически определяет ширину поля

Построитель можно вызывать автоматически, если нажать кнопку **Builder Lock** на панели **Form Controls**. Почти все элементы формы имеют построители.

Пример создания `TextBox` смотрите на CD в \char5\textbox_in_form.scx.

Timer (Таймер)

Таймер — это элемент управления, размещаемый в форме и обрабатывающий данные системных часов. Данный объект используется для периодического выполнения определенных операций, например, периодический опрос какого-либо системного журнала на предмет проверки появления нового сообщения.

Ключевыми для работы этого объекта являются следующие свойства и события (табл. 5.10).

Таблица 5.10. *Основные свойства и события объекта* `Timer`

Название	Тип	Описание
`Enabled`	Свойство	Применительно к объекту `Timer` включает или выключает его работу

`Interval`	Свойство	Указывает интервал времени в миллисекундах (одна тысячная доля секунды), через которое должно срабатывать событие `Timer`
`Timer`	Событие	Событие, которое срабатывает через интервал времени, установленный в свойстве `Interval`

Существует некоторое непонимание того, как именно работает таймер. Да, он срабатывает через указанный промежуток времени, но все дело в том, что событие `Timer` не всегда может быть выполнено немедленно, как только настало время его работы.

Дело в том, что, пока не завершилось выполнение одной процедуры или метода в Visual FoxPro, выполнение другой процедуры не может быть начато. Кроме случаев принудительного запуска событий через команду `DOEVENTS` изнутри этих долгих процедур. Это относится и к событиям таймера.

Например, вы установили интервал срабатывания таймера каждую секунду и одновременно запустили некую процедуру, время выполнения которой примерно равно 10 секунд.

Через секунду пришло время срабатывания события таймера. Но это событие немедленно выполнено быть не может, поскольку в этот момент идет обработка другой процедуры и событие таймера ставится в очередь на выполнение.

Еще через секунду настало время выполнения следующего события таймера, но оно также не может быть выполнено, и также становится в очередь.

В результате, через 10 секунд, когда закончилось выполнение процедуры, в очереди окажется 10 событий таймера, которые только сейчас и начнут выполняться, тоже по очереди.

То есть получилось, что в течение 10 секунд не произошло ни одного срабатывания события таймера, а через 10 секунд выполнилось сразу 10 событий таймера с очень маленьким интервалом.

Эту особенность обработки событий в FoxPro следует учитывать при использовании таймеров в вашем приложении.

Например, если вы делаете "одноразовый" таймер, т. е. таймер, который должен сработать только один раз и тут же сам себя выключить, то обязательно перед запуском события таймера надо проверить: а действительно ли таймер выключен или это просто настало время выполнения накопившейся очереди.

Другими словами, код события `Timer` для "одноразового" таймера желательно писать примерно так (листинг 5.2).

```
IF This.Enabled = .T.
        * Необходимая обработка
        This.Enabled = .F.
ENDIF
```

Toolbar

Можно сказать, что Toolbar — это специфическая форма. Более того, открытые Toolbar попадают в коллекцию форм _VFP.Forms или _SCREEN.Forms.

Однако в отличие от форм, объекты Toolbar создаются только и исключительно через классы. Никаким другим способом их создать невозможно.

Сам экземпляр нужного Toolbar создается при открытии формы прямой командой CreateObject() или NewObject().

Если экземпляр Toolbar связан с конкретной формой, то, как правило, в форме создается свойство, содержащее прямую ссылку на объект Toolbar, чтобы можно было его удалить при закрытии формы.

Собственно взаимодействие самого Toolbar с формой похоже на взаимодействие меню с формой. Обычно при нажатии на кнопку Toolbar вызывается некий метод активной в данный момент формы. Делается это путем создания в форме соответствующих методов с заранее фиксированными именами. Например, если надо сохранить документ, то создаем метод с именем SaveDocument во всех формах, где такая операция в принципе имеет смысл.

Теперь при нажатии на кнопку Toolbar необходимо сделать две вещи. Проверить, что у активной в данный момент формы существует метод с таким именем, и запустить его на выполнение, если он существует.

Тонкость заключается в том, что сам Toolbar активным быть не может. Даже при нажатии на кнопку объект Toolbar не получает фокуса и не становится активным. Как следствие, активной может быть только и исключительно какая-либо форма.

В Visual FoxPro у основного окна _SCREEN существует свойство ActiveForm, которое возвращает ссылку на активную в данный момент форму. Поэтому собственно вызов метода из Toolbar будет выглядеть так:

```
_SCREEN.ActiveForm.SaveDocument()
```

Однако если никакой формы не открыто или форма не имеет такого метода, то выполнение этой команды приведет к ошибке. Чтобы этого не произошло, необходимо выполнить дополнительные проверки, как на факт существования активной формы, так и на факт существования у нее нужного метода. В результате код получится примерно таким:

```
IF TYPE("_VFP.ActiveForm")="O" AND
PemStatus(_VFP.ActiveForm,"SaveDocument",5)=.T.
        _SCREEN.ActiveForm.SaveDocument()
ENDIF
```

Здесь использована системная переменная _VFP, а не _SCREEN. Но в отношении свойства ActiveForm обе они дают один и тот же результат.