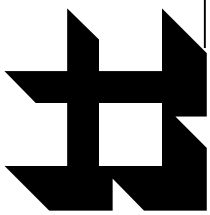

ГЛАВА 10

Использование внешних компонентов

Общие сведения об ActiveX управляющих элементах

Современные приложения создаются как совокупность встроенных компонентов. Вы выбираете элемент управления, устанавливаете его на форму, и вам становятся доступными все его функции. Так, например, кнопка, встроенная в форму, уже знает, как реагировать на щелчок мыши или на перемещение над ней указателя мыши. Она имеет событие `Click`, и все, что требуется от разработчика, — это написать код, обрабатывающий нажатие кнопки.

Получается, что управляющие элементы — это некие "строительные блоки", из которых происходит сборка приложения. Разработчики уже привыкли, что любой современный язык программирования содержит такие многократно используемые "блоки", активно их применяют и воспринимают как должное сам факт их существования.

Visual FoxPro имеет набор стандартных компонентов, таких как `ComboBox` (комбинированный список), `Command Button` (кнопка) и т. д. Однако этих компонентов не всегда достаточно. Поэтому при создании программ можно применять компоненты, лежащие в отдельных файлах — ActiveX.

ActiveX — это технология, разработанная фирмой Microsoft с целью стандартизации программных компонентов. В основе ее лежит **COM** (**C**omponent **O**bject **M**odel — модель компонентного объекта). Компоненты ActiveX представляют собой 32-разрядные объекты, разработанные различными фирмами-разработчиками или отдельными программистами, которые вы можете использовать в своих программах. Для создания ActiveX могут быть использованы различные языки программирования, например, Visual Basic или Visual C++.

Все ActiveX-компоненты можно разделить на два типа:

- *визуальные*, например, ListView, ProgressBar, календарь и пр.;

- *невизуальные*, но обеспечивающие какую-либо функцию, например, таймер или коммуникационные элементы управления.

Для большего удобства работы список часто используемых компонентов можно создать на панели **Form Controls**. Для этого произведите следующие действия:

1. Выберите **Tools | Options | Controls**.
2. Установите опцию **ActiveX Controls** (рис. 10.1).
3. В списке **Selected** отметьте те, которые намереваетесь использовать (список **Selected** содержит компоненты ActiveX, установленные на вашем компьютере).
4. Нажмите кнопку **OK** для закрытия диалогового окна **Options**.

Вы можете также добавлять новые объекты в список **Selected** с помощью кнопки **Add…**. При нажатии этой кнопки открывается диалоговое окно **Open**, которое позволяет выбрать файл с ActiveX-компонентами.

"Где их взять?" — спросите вы. Для обеспечения своевременной информированности разработчиков о новых ActiveX-компонентах Microsoft использует свой сайт, где размещаются сами компоненты, доступные для скачивания. Кроме того, для поиска необходимых компонентов можно использовать Интернет, который изобилует платными и бесплатными объектами ActiveX.

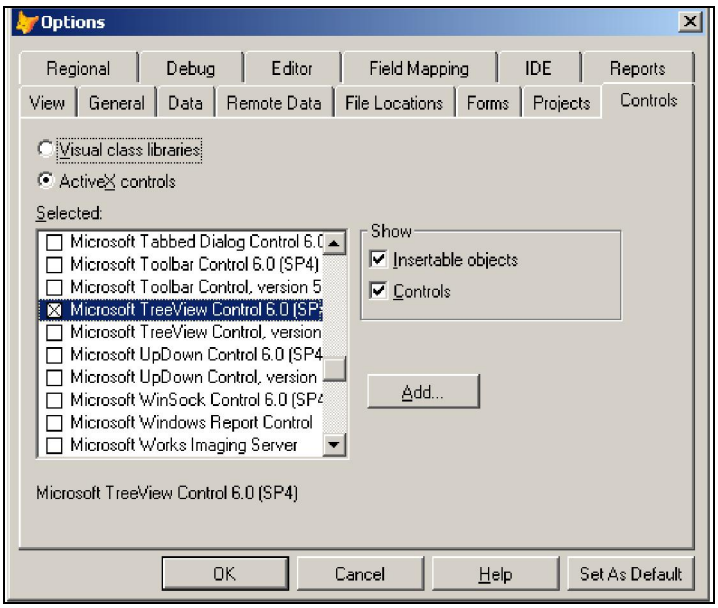

Рис. 10.1. Настройка списка компонентов **ActiveX Controls**

Для того чтобы выбранные вами при настройке компоненты отображались на панели **Form Controls**, выполните следующие действия:

1. Откройте форму, на которой вы планируете разместить ActiveX.
2. Выберите на панели **Form Controls** кнопку **View Classes** (рис. 10.2).

Рис. 10.2. Кнопка **View Classes** на панели **Form Controls**

3. Выберите пункт **ActiveX Controls** в выпадающем меню.
4. На панели инструментов появятся кнопки с пиктограммами выбранных вами компонентов.

Кроме того, установленные вами компоненты появятся в галерее компонентов: **Tools | Component Gallery | ActiveX Catalog | Controls | Installed Controls**. Если компонент не появился в списке, следует нажать правую кнопку мыши на узле **Installed Controls** и выбрать в контекстном меню пункт **Refresh Controls** (рис. 10.3).

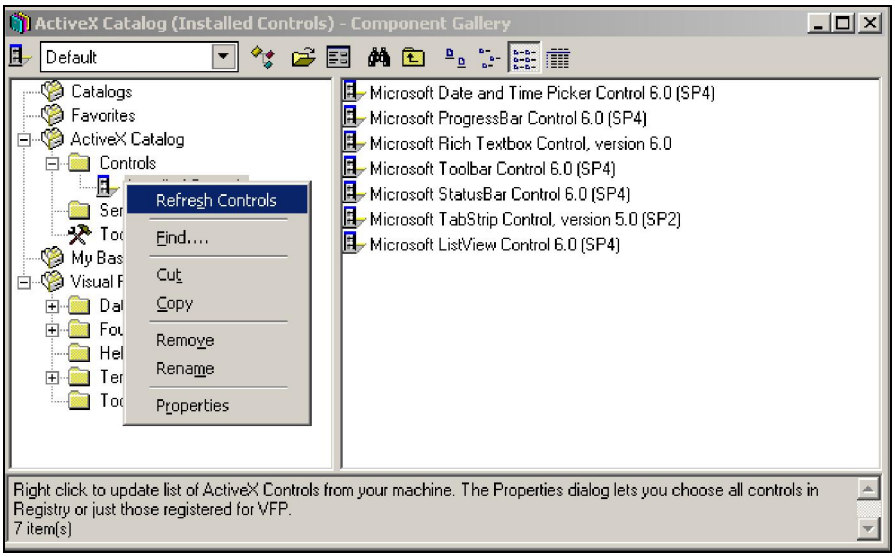

Рис. 10.3. Галерея компонентов

Теперь можно приступать к размещению объектов ActiveX на форме. Собственно, ничего сложного в этих действиях нет, объекты ActiveX размещаются на форме как обычные стандартные компоненты. Рассмотрим установку ActiveX на форме на примере. Допустим, нам необходимо разместить на форме элемент `TreeView`. С этой целью мы выполним следующие действия:

1. Откроем проект.
2. Создадим новую форму (**Documents | Forms | New | New Form**).
3. Нажмем на кнопку **View Classes** () на панели **Form Controls**.
4. Выберем в появившемся меню пункт **ActiveX Controls**, на панели отобразятся пиктограммы ActiveX-компонентов, выберем нужную пиктограмму, нажмем на кнопку и перетащим элемент на форму.

Второй вариант размещения — это использование **Component Gallery**: **Tools | Component Gallery | ActiveX Catalog | Controls | Installed Controls**. Здесь используется тот же самый принцип — найти компонент в списке, "захватить" его левой кнопкой мыши и перетащить на форму.

После запуска формы поведение объектов ActiveX определяется заданными в них алгоритмами.

Описание и настройка отдельных компонентов ActiveX

Некоторые компоненты ActiveX стали очень популярны среди программистов. Поэтому есть смысл описать несколько таких компонентов.

Календарь

Календарь является одним из многих ActiveX-элементов и применяется там, где требуется ввод даты (рис. 10.4).

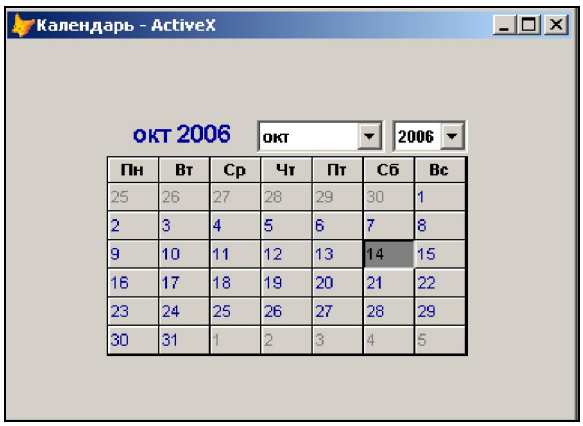

Рис. 10.4. Форма с размещенным на ней элементом ActiveX "Календарь"

Рис. 10.5. Контекстное меню свойств **Календаря**

Если нажать на ActiveX-компоненте правую кнопку мыши, мы увидим еще один пункт контекстного меню — **Календарь Свойства** (рис. 10.5).

После выбора этого пункта меню на экране отображается диалоговое окно свойств календаря (рис. 10.6).

При изменении любого из свойств кнопка **Применить** становится активной, что позволяет запомнить внесенные изменения.

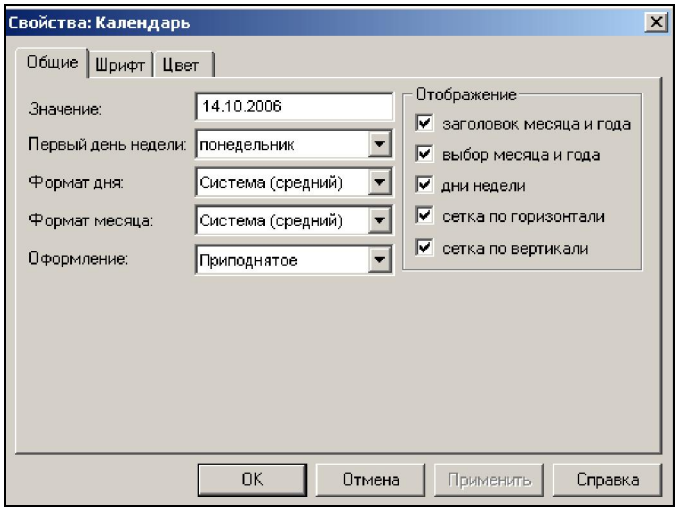

Рис. 10.6. Календарь — настройка свойств

Теперь разместим компонент на форме с данными, требующими ввод даты, и используем календарь для ввода даты в таблицу (рис. 10.7).

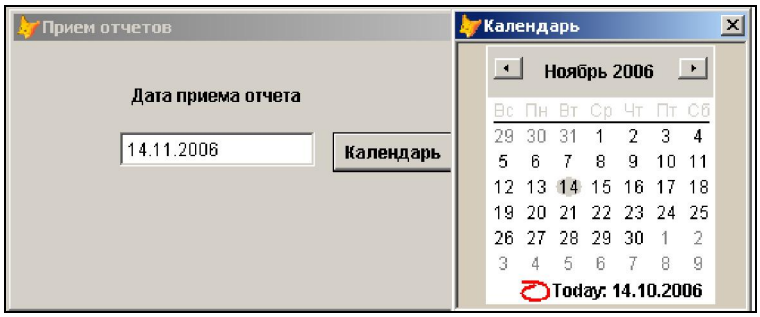

Рис. 10.7. Внешний вид формы с календарем

На форме размещаем `TextBox` для ввода даты. Рядом с ним помещаем кнопку для вызова календаря. Для того чтобы календарь появился при нажатии на кнопку, в его событии `Click()` пишем следующий код (листинг 10.1).

```
oCalform=CREATE('form')
ocalform.NEWOBJECT("oleCalendar","_olecalendar","_datetime")
With ocalform.oleCalendar
        .date_column="otchet.date_ot"
        .refreshDisplay()
        .Visible=.t.
ENDWITH
WITH ocalform
        .left=300
        .height=ocalform.olecalendar.height
        .width=ocalform.olecalendar.width
        .caption="Календарь"
endwith
ocalform.show(1)
thisform.Refresh()
```

Вышеприведенный пример можно посмотреть на CD в Char10\ActiveX\Calen.

TreeView

`TreeView` отображает иерархический список узловых объектов, именуемых `Node`, каждый из которых состоит из метки и необязательной картинки формата bitmap. Типичное использование `TreeView` заключается в отображении заголовков в документах, файлов и каталогов на диске или любого другого вида информации, которую удобнее отображать в виде иерархического списка (рис. 10.8).

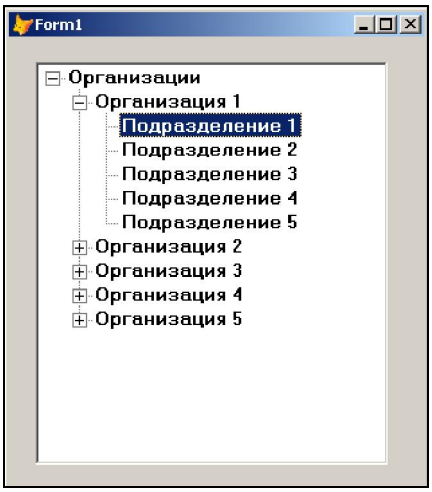

Рис. 10.8. Внешний вид элемента `TreeView`

После настройки компонента TreeView вы можете добавлять, удалять, преображать, в общем, манипулировать узловыми объектами (узловой объект называется Node) при помощи установки свойств и вызовов методов, т. е. то, что мы обычно и делаем с объектами в VFP. Вы можете программным путем разворачивать или сворачивать узловые объекты для отображения или скрытия всех дочерних узловых объектов. Программную функциональность компонента обеспечивают три события — Collapse, Expand и NodeClick.

Компонент TreeView относится к классу ActiveX-компонентов, т. е. он может быть встроен в любой объект, обеспечивающий функциональную поддержку ActiveX-компонентов. Таким объектом в VFP является форма или пользовательский контейнер (который, по сути, базируется на основе той же формы). Для ввода компонента TreeView создайте новую форму. Затем выберите из системного меню VFP **Tools** пункт **Component Gallery**. Откроется окно **Catalogs — Component Gallery**. В левой части окна имеется дерево, в котором нужно развернуть узел **ActiveX Catalog** (если он свернут, разумеется). После этого разверните папку **Controls** и выберите пункт **Installed controls**. В правой части окна появится список установленных компонентов. Расширьте колонку Object и найдите в списке Microsoft TreeView Control, version 6.0. Выберите его щелчком по левой кнопке мыши, затем щелкните по правой кнопке мыши и из всплывающего меню выберите пункт — Add to form и в расширении меню выберите имя вашей формы, в которую вы хотите встроить компонент. Если у вас открыта для редактирования одна форма, то именно ее имя и будет в списке расширения всплывающего меню **Add to form** (рис. 10.9).

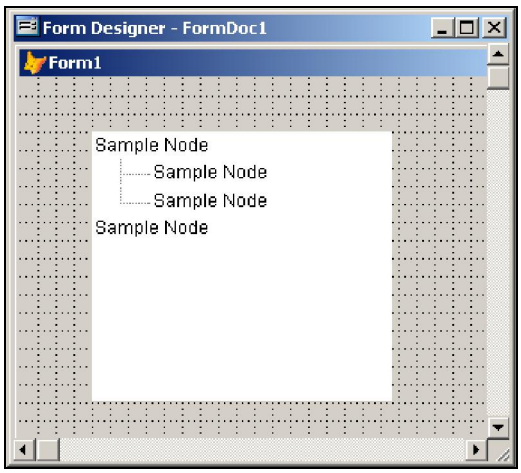

Рис. 10.9. Внешний вид компонента TreeView, установленного на форме

Как установить свойства *TreeView*?

Попробуйте выполнить щелчок по правой кнопке мыши на компоненте TreeView, размещенном в вашей форме. Обратите внимание, что всплывающее меню объекта

формы видоизменилось, и в самой нижней строке указано — **TreeCtrl Properties**. Выберите этот пункт из меню. Откроется новое окно **TreeCtrl Properties** с тремя вкладками — **General**, **Font**, **Picture**. Здесь вы сможете установить свойства, присущие этому объекту как компоненту семейства **Windows Controls**. При этом Windows соберет свойства, уже присвоенные вами этому компоненту, так что возможна некоторая задержка при попытке установить или изменить какие-то свойства.

Большая часть свойств объекта `TreeView` размещена на вкладке **Properties | All**, а из поля, у которого в качестве величины указано значение ***Object***, вы сможете попасть в упомянутое выше окно установки свойств **Свойства: TreeCtrl**. Так что можно пользоваться обоими окнами редактирования свойств компонента.

Обратимся к свойствам компонента. На вкладке **General** вы найдете 9 комбинированных списков, два текстовых поля и восемь флажков (рис. 10.10, табл. 10.1—10.3).

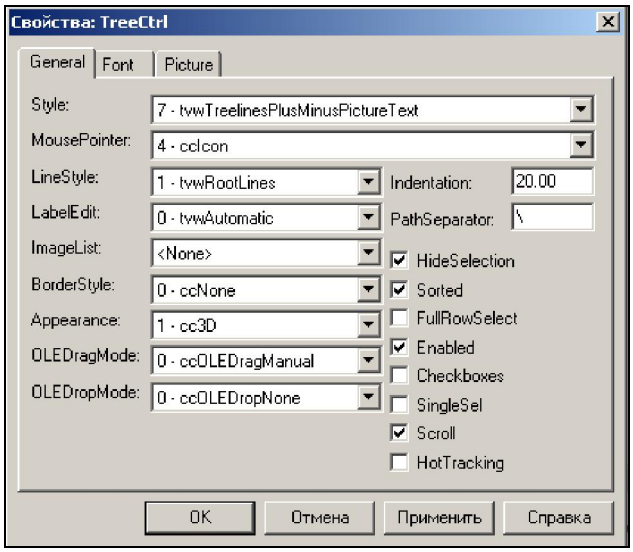

Рис. 10.10. Окно свойств **TreeCtrl**

Таблица 10.1. *Описание комбинированных списков вкладки* ***General***

Свойство	По умолчанию установлено	Возможные варианты
Style	7-tvwTreelinesPlusMinusPictureText	0-tvwTextOnly 1-tvwPictureText 2-tvwPlusMinusText 3-tvwPlusPictureText 4-tvwTreelinesText 5-tvwTreelinesPictureText 6-tvwTreelinesPlusMinusText 7-tvwTreelinesPlusMinusPictureText Если вы желаете использовать картинки с этим компонентом, тогда выберите подходящий вариант с обязательным наличием слова Picture. Однако, если вы выберите значение 7 или иное, с включенным в описание свойства словом Picture, это совсем НЕ означает, что вы ОБЯЗАНЫ использовать картинки с этим компонентом!
MousePointer	0-ccDefault	16 стандартных вариантов указателя мыши, плюс пользовательский. Любители спецэффектов могут установить значение этого свойства в 99 и использовать свою собственную иконку для указателя мыши

Таблица 10.1 *(окончание)*

Свойство	По умолчанию установлено	Возможные варианты
LineStyle	0-tvwTreelines	0-tvwTreelines 1-tvwRootLines Свойство устанавливается, если вы выбрали значение Style равным от 4 до 7. Если вы желаете показать вертикальные линии, установите значение этого свойства в 1. Кроме того, если вы желаете, чтобы у родительского элемента указывался бы плюсик, означающий, что родительский элемент имеет дочерние ветви, установите значение этого свойства также равным 1
LabelEdit	0-tvwAutomatic	0-tvwAutomatic 1-tvwManual Для полного контроля над компонентом установите значение этого свойства равным 1
ImageList	<None>	Определенный вами компонент ImageList

BorderStyle	`0-ccNone`	`0-ccNone`
		`1-ccFixedSingle`
		свойство устанавливается, если вы выбрали значение `Appearance` равным 0. Если вы установили трехмерное изображение в Appearance (1), то игнорируется

Appearance	`1-cc3d`	`0-ccFlat` `1-cc3D`

OLEDragMode	`0-ccDragManul`	`0-ccDragManual`
		`1-ccDragAutomatic`
		на ваше усмотрение, в зависимости от вашей привычки работы с методами Drag&Drop

OLEDropMode	`0-ccOLEDropNone`	`0-ccOLEDropNone`
		`1-ccOLEDropManual`
		если вы желаете корректно использовать метод Drag&Drop, установите это значение в 1

Таблица 10.2. *Описание текстовых полей вкладки* ***General***

Свойство	По умолчанию установлено	Возможные варианты
Indentation	`38`	Устанавливает отступ по вертикали дочерних ветвей от родительских

Таблица 10.2 *(окончание)*

Свойство	По умолчанию установлено	Возможные варианты
PathSeparator	`\`	Используется при обращении к свойству `Path` из программы, для определения глубины отношений между дочерними и родительскими ветвями

Таблица 10.3. *Описание флажков вкладки* ***General***

Свойство	По умолчанию установлено	Возможные варианты
HideSelection	Отмечен	Если вы хотите оставить выбранную ветку подсвеченной, когда вы переходите к другому объекту в форме, то оставьте эту установку
Sorted		Если вы хотите сортировать текст меток, установите флажок

FullRowSelect		При установленном флажке будет подсвечиваться вся строка выбранного элемента `TreeView`, при снятом — только текст метки
Enabled	Отмечен	Снятый флажок запрещает визуальное представление компонента
Checkboxes		Включать или не включать checkbox перед текстом метки. **Очень важное свойство.** Включение checkbox дает вам возможность дополнительного контроля над элементом посредством свойства **NodeChecked**
SingleSel		Означает, будет ли свертываться ранее выбранный элемент, имеющий дочерние ветки, при выборе другого элемента. Упрощает просмотр содержимого
Scroll	Отмечен	При установленном флажке будут отображены полосы прокрутки на компоненте
HotTracking		При перемещении указателя мыши поверх компонента каждый элемент, над которым в данный конкретный момент времени будет проходить указатель мыши, будет подчеркиваться тонкой линией

На некоторых версиях операционной системы Windows 9x шрифт по умолчанию установлен в дробную величину 8,25. Если вы попробуете установить размер точно равным 8, то компонент проигнорирует ваше желание.

Для получения ссылки на объект `TreeView` можно использовать следующий синтаксис:

```
container.object_TreeView.
```

Основные свойства и методы объекта `TreeView` приведены в табл. 10.4.

Таблица 10.4. *Свойства и методы объекта* `TreeView`

Свойство	Описание
Метод `Add`	Добавляет объект `Node` в коллекцию `Nodes`
Метод `Clear`	Удаляет все объекты коллекции `Nodes`. Метод не имеет параметров
Свойство `Count`	Возвращает численное значение равное числу объектов `Node` в коллекции `Nodes`
Свойство `Item`	Возвращает ссылку на конкретный объект `Node` коллекции `Nodes` по его позиции или ключу. *Index* — целое число или строка, позволяющие адресоваться к индивидуальному узлу коллекции узлов
Метод `Remove`	Удаляет конкретный объект из коллекции `Nodes`. *Index* — целое число или строка, которые уникально определяют члена коллекции `Nodes`. Целое число представляет собой значение свойства `Index`; строка — значение свойства `Key` объекта `Node`

Теперь необходимо добавить данные в `TreeView`, те, что будут показаны в узлах этой древовидной структуры. Для этого используется метод `Add`, который добавляет объект `Node` в коллекцию `Nodes` (табл. 10.5).

Синтаксис

```
Object.Nodes.Add(relative,relationship,key,text,image,selectedimage)
```

Таблица 10.5. *Параметры, передаваемые методу* `Add`

Параметр	Описание
`Relative`	Необязательный параметр, представляющий собой индексный номер или ключ уже существующего узлового объекта `Node`. Взаимосвязь нового узлового объекта и уже существующего узлового объекта помещена в следующем аргументе — `relationship`
`RelationShip`	Необязательный параметр, точно определяющий относительное положение узлового объекта `Node`, как описано в установках для этого параметра
`Key`	Необязательный параметр, представляющий собой уникальную строку — ключ (идентификатор), который может быть использован для нахождения и возврата значения объекта `Node` при помощи метода `Item`
`Text`	Обязательный параметр, который представляет собой тот самый текст, который и появится в виде метки узла
`Image`	Необязательный параметр. Индекс картинки в ассоциированном контроле `ImageList`
`SelectImage`	Необязательный параметр. Индекс картинки в ассоциированном контроле `ImageList`, который показывается, когда узловой объект `Node` выбран

Можно опустить все параметры метода, кроме текста. Пропишем в методе `Init()` несколько строк вызова метода `Add()` с использованием следующего синтаксиса:

```
this.Nodes.Add(,,,text)
```

где запятые показывают три опущенных параметра: `relative, relationship и key`

```
this.Nodes.Add(,,,"Организация 1")
this.Nodes.Add(,,,"Организация 2")
```

В результате получаем следующую картинку (рис. 10.11).

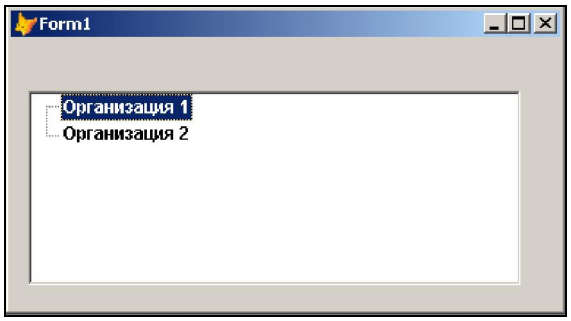

Рис. 10.11. `TreeView` с добавлением 2 объектов `Node`

При выстраивании себя `TreeView` автоматически присваивает цифровой индекс каждому введенному узлу; при этом порядок индексирования определяется порядком добавления узлов. То есть первый введенный узел получает индекс равный единице, второй — двойке и т. д. Значение ключа остается в приведенном примере пустым.

Модифицируем пример. Каждому узлу в описанном выше коде присвоим свои уникальные ключи, т. е. заполним 3-й параметр метода `Add()`. При этом необходимо учесть, что в *качестве ключа может быть использовано только символьное выражение, причем оно не должно начинаться с цифры*. Если вы попытаетесь ввести в качестве ключа численное выражение, или символьное выражение, начинающееся с цифры, вы получите сообщение об ошибке "`OLE IDispatch exception code 0 from Node. Invalid key`". Теперь код из метода `Init()` может выглядеть, например, так:

```
this.Nodes.Add(,,"n1","Организация 1")
this.Nodes.Add(,,"n2","Организация 2")
```

Вы можете запустить и эту форму (см. пример на CD char10\treeview\treeview.scx). Правда, никаких изменений в картинке на экране вы не увидите. Да и откуда им взяться, т. к. мы в данном куске кода добавили к меткам узлов только уникальные ключи.

Нам нужен главный, корневой узел — `Root`. И мы его введем, добавив следующий код:

```
this.Nodes.Add( , ,"r0", "Организации")
```

Наша картинка изменилась, у нас имеется корневой узел, и теперь наша задача — подчинить остальные 2 узла главному.

При выстраивании структуры объект `TreeView` автоматически заносит в свойство `Root` ссылку на первый введенный в структуру узел. Если в процессе строительства дерева будет введен узел и параметру `relationship` будет передано значение, равное нулю, а параметру `relative` будет передана пустая строка, это не изменит ссылку на объект `Root`. Однако, если в процессе строительства будет введен узел с непустым параметром `relative`, а параметру `relationship` будет передано значение, равное нулю, этот узел будет введен в ссылку на корневой узел для всех узлов всех уровней иерархии

объекта `TreeView`. При этом не имеет значения, ключ какого узла указан в качестве `relative`.

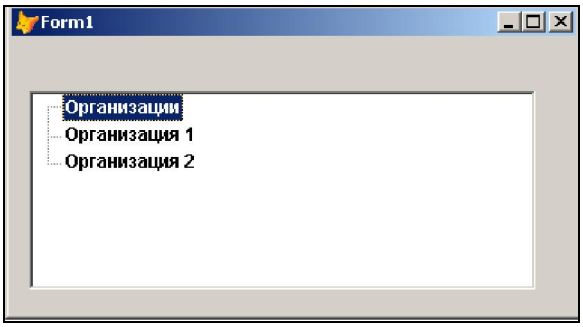

Рис. 10.12. Ввод главного узла

Введите в метод `Init()` следующий код

```
this.Nodes.Add( , ,"r0", "Организации")
this.Nodes.Add("r0",4,"d1","Организация 1")
this.Nodes.Add("r0",4,"d2","Организация 2")
this.Nodes.Add("d1",4,"s1d1","Подразделение 1")
this.Nodes.Add("d1",4,"s2d1","Подразделение 2")
this.Nodes.Add("d2",4,"s1d2","Подразделение 1")
this.Nodes.Add("d2",4,"s2d2","Подразделение 2")
```

Картинка снова меняется, теперь у нас есть корневой узел, два родительских узла и четыре подчиненных (рис. 10.13).

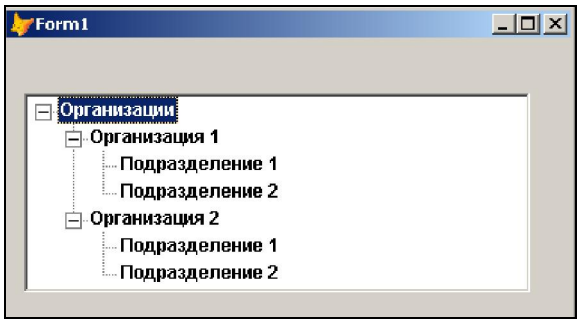

Рис. 10.13. Древовидная структура `TreeView`

Пока что мы использовали в качестве текста символьные константы. Попробуем теперь заполнить `TreeView` данными из таблицы.

В качестве узла ключа можно использовать первичный ключ таблицы или ключ `candidate` из свободной таблицы.

Сколько узлов должно располагаться на первом уровне иерархии? 100? 200? 1000? 500 000? Чем меньше, тем лучше. И это связано не с параметрами компьютера, на котором будет работать ваше приложение, но с параметрами пользователя. Лучше всего, если будет не более 3—4 объемов данных, располагающихся в видимой части `TreeView`. Лучше немного поработать и разработать удобный для пользователя интерфейс, позволяющий быстро отбирать необходимые порции данных.

Каждый родительский узел может иметь "потомков", а может и не иметь. Как разработать структуру таблицы, учитывающей такую "наследственность"?

Вариантов может быть много, мы предлагаем следующий (рис. 10.14).

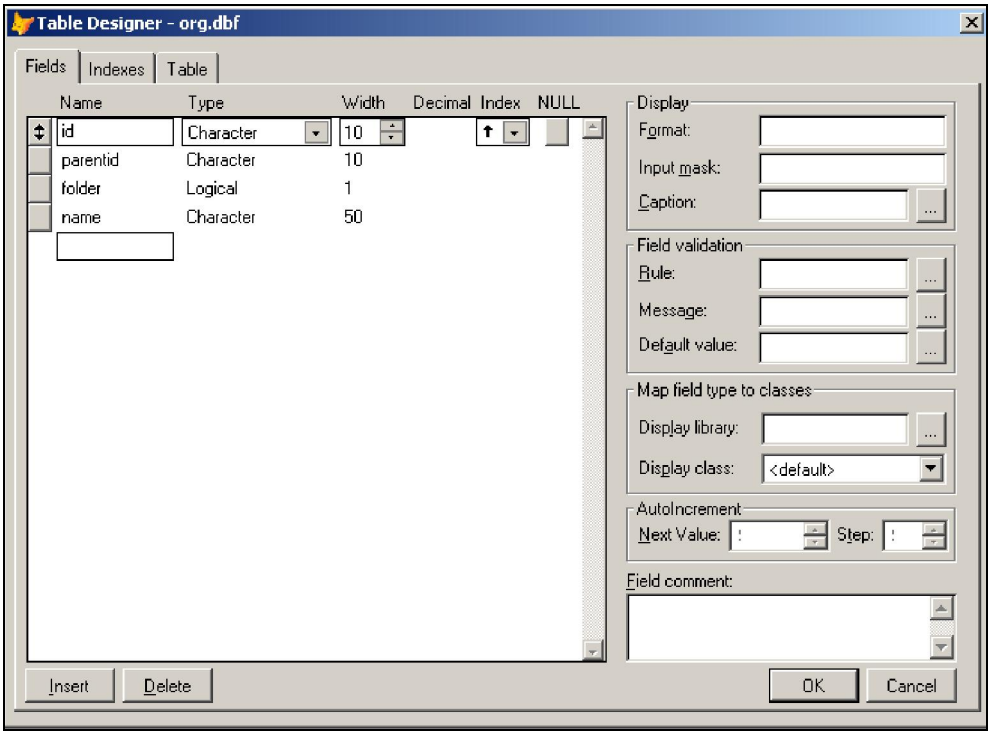

Рис. 10.14. Структура таблицы для `TreeView`

В этой таблице:

- `id` — идентификатор строки;
- `parentID` — идентификатор отношений "родитель-потомок";
- `folder` — является ли узлом;
- `name` — текст наименования узла;
- `primery` — индекс построен по полю ID.

Данные в таблице выглядят следующим образом (рис. 10.15).

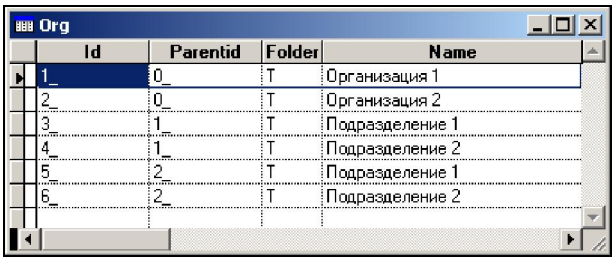

Рис. 10.15. Таблица `Org`

Теперь осталось поместить в метод `Init()` формы следующий код (листинг 10.2):

```
This.LockScreen = .T.
SELECT org.id, org.parentid, org.name;
 FROM org;
 WHERE org.folder=.T.;
 ORDER BY org.parentid ;
 INTO CURSOR tmpTreeView
o = THISFORM.oleControl1.Nodes
o.Clear
SELECT tmpTreeView
SCAN
        IF ALLTRIM(parentid) = '0_'
                o.ADD(,1,ALLTRIM(id),ALLTRIM(name),0)
        ELSE
                o.ADD(ALLTRIM(parentid),4,ALLTRIM(id), ALLTRIM(name),0)
        ENDIF
ENDSCAN
SELECT tmpTreeView
USE
This.LockScreen = .F.
```

Кроме приведенного решения об отношениях "родитель — потомок", существует масса других — от вспомогательных курсоров до быстрых выборок, тем более что эти выборки при грамотно выстроенной базе данных простые по синтаксису и, как правило, полностью оптимизируемые.

Далее смотрим — что может сделать пользователь. Если мы установили для `TreeView` свойства `Style` и `LineStyle` равными соответственно 7 и 1, то пользователь видит компонент в том виде, как он показан на рисунках выше. При этом пользователь может раскрыть нужный ему узел как с помощью щелчка мышью на крестике у соответ-

ствующего узла, так и произведя двойной щелчок на этом узле. В обоих случаях сработает событие `Expand`. Метод, ассоциированный с этим событием, получает в качестве параметра объект узла, который подлежит раскрытию.

Управляющий элемент *ActiveX Bound Control*

Некоторые приложения, предназначенные, например, для ведения автоматизированного делопроизводства, отдела кадров или создания конструкторских документов и чертежей, связаны с хранением большого количества документов. Это могут быть фотографии, письма, чертежи и прочие объемные документы. Для просмотра и редактирования каждого из таких документов существует свое приложение. Например, фотографии можно просматривать с помощью Microsoft FotoEditor или ACDSee, тексты с расширением PDF удобно читать с помощью Adobe Acrobat Reader, а с электронными таблицами XLS можно работать с помощью Microsoft Excel. Возникает вопрос: существует ли технология, позволяющая хранить эти документы или ссылки на них в таблицах Visual FoxPro, извлекать их оттуда по мере необходимости и отображать на экране с помощью того приложения, в котором они были созданы? Да, такой механизм есть. Его называют технологией связывания и внедрения объектов OLE (от *Object Linking and Embedding*).

OLE-технология позволяет передавать часть работы от одной программы к другой и возвращать результаты назад. Она используется при обработке составных документов, может быть использована при передаче данных между различными несвязанными между собой системами посредством интерфейса переноса, а также при выполнении операций с буфером обмена. Однако технология OLE использует архитектуру "толстого" клиента, т. е. сетевой компьютер с мощными вычислительными ресурсами. Это означает, что тип файла либо программа, которую пытаются внедрить, должна присутствовать на машине клиента. Например, если OLE оперирует документами Word, то программа Word должна быть инсталлирована на машине пользователя. Документ приложения Word с расширением doc является OLE-объектом, а само создавшее этот документ приложение — Microsoft Word — будет являться приложением-сервером.

Но вернемся к Visual FoxPro. Visual FoxPro может хранить в таблицах объекты или ссылку на объекты, которые созданы с помощью приложения, выступающего в качестве OLE-сервера. Такими объектами могут быть файлы, содержащие тексты документов, фотографии, звук и т. д. В таблице такие данные хранятся в полях типа `General`. Если поле типа `General` пусто, то при просмотре таблицы в нем имеется пометка gen. Если же поле содержит OLE-объект, то пометка написана с прописной (большой) буквы — Gen. Объекты при этом сохраняют связь со своим исходным приложением, что позволяет использовать OLE-сервер для внесения изменений в объект. Если объект связывается с сервером, то в таблице хранится только ссылка на сервер и объект. Как показывает практика, этот вариант более предпочтительный, поскольку документы могут изменяться. Процесс включения в документ нового или существующего OLE-объекта называется *внедрением*. Внедренный объект сохраняет

всю информацию, необходимую для повторного создания объекта. Процесс включения в объект ссылки на OLE-объект называется *связыванием*.

Каждый OLE-объект относится к некоторому классу, определяемому приложением-родителем. Например, имя класса для Word будет называться `WORDDOCUMENT`. Для определения класса сервера запустите программу REGEDIT и дважды щелкните мышью на OLE-объекте.

Внедрение OLE-объектов

Для внедрения объекта в поле типа `GENERAL` откройте в режиме **Browse** таблицу, содержащую поле данного типа, и нажмите дважды левую кнопку мыши на поле. Откроется окно редактирования этого поля. Выполните команду **Edit | Insert Object**. Откроется диалоговое окно **Вставка объекта**. В этом окне есть два варианта внедрения объекта:

- для создания нового объекта выберите из списка **Тип объекта** нужный сервер и нажмите кнопку **ОК**, тогда сервер откроется для создания объекта. Создайте объект и выйдите из сервера;
- для использования уже готового объекта установите опцию **Создать из файла**, откроется окно **Обзор**, выберите нужный файл и нажмите **ОК**.

Кроме того, можно скопировать объект в буфер и в поле `GENERAL` выполнить команду меню **Edit-paste special** (рис. 10.16).

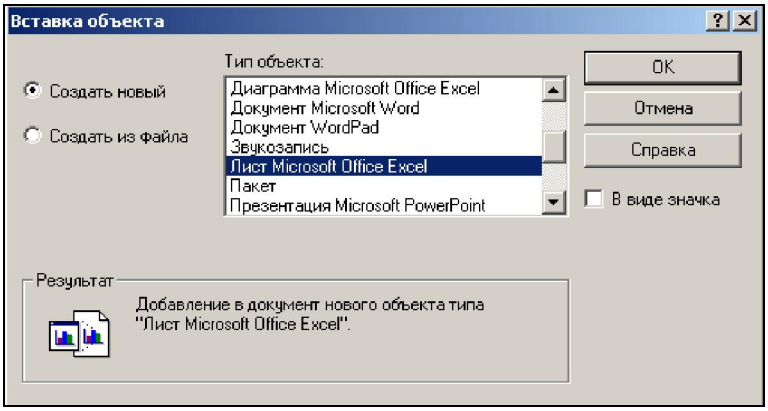

Рис. 10.16. Диалоговое окно **Вставка объекта**

Для отображения картинки из поля `GENERAL` используется **ActiveX Bound Control** (Ole Bound Control) — этот инструмент есть в Конструкторе форм. В частности, этот элемент можно встроить в `Grid` и сопроводить, таким образом, записи иллюстрациями. Размещать картинку в поле `GENERAL` лучше всего через буфер обмена.

Связывание объектов

Для связывания существующего объекта нужно открыть сервер и запомнить связываемый объект в буфере. Затем в поле GENERAL воспользуйтесь командой меню **Edit-paste special**. На экране появится диалоговое окно **Специальная вставка**. Объект, который находится во временном буфере Windows, описан в окне диалога как источник. Список **Как** перечисляет допустимые форматы для объекта. В появившемся окне выберите опцию **Связать**. Ссылка на объект при этом запоминается в поле типа GENERAL. В этом поле будет сохранен путь и имя файла. При каждом изменении оригинала объекта на сервере, копия объекта, хранящаяся в приложении Visual FoxPro, будет автоматически обновляться. Если в момент обновления приложение не открыто, то обновление произойдет при первом же открытии приложения Visual FoxPro. Для связанных OLE-объектов ссылка на объект, как уже говорилось, содержит имя файла и путь к нему. Если файл будет переименован, перенесен или удален, соответственно, исчезнет и ссылка на него. Если связь будет разорвана, воспользуйтесь командой **Links** меню **Edit**.

Для удаления объекта из поля выполните команду **Clear** меню **Edit**.

Для изменения объекта в поле типа GENERAL со связанного на внедренный необходимо удалить объект из поля с помощью команды **Edit | Cut**, а потом вставить объект обратно, но уже без связывания.

Редактирование OLE-объектов

Ряд OLE-объектов допускают редактирование. Например, могут быть отредактированы документы Word, электронные таблицы Excel, рисунки Paint. Приложение-сервер для редактирования объекта можно вызвать путем двойного нажатия левой кнопки мыши. Точечный рисунок можно отредактировать, если использовать команду **Edit | Точечный рисунок объект | Изменить или Edit | Точечный рисунок объект | Открыть**. Если вы хотите отредактировать документ Word, то в меню **Edit** появляется пункт **Документ Object**.

Для работы с объектами программно используются описанные далее команды:

MODIFY GENERAL — открывает окно редактирования для указанного поля GENERAL

Синтаксис команды:

```
MODIFY GENERAL GeneralField1 [, GeneralField2 ...] [NOMODIFY]
   [NOWAIT] [[WINDOW WindowName1] [IN [WINDOW] WindowName2 | IN SCREEN]]
```

При выполнении команды MODIFY GENERAL открывается окно редактирования, выполнение программы приостанавливается до выхода из этого окна. При включении опции NOWAIT выполнение программы будет продолжено.

APPEND GENERAL — импортирует объект из файла в поле GENERAL текущей записи.

```
APPEND GENERAL GeneralFieldName [FROM FileName]
```

```
[DATA cExpression] [LINK] [CLASS OLEClassName]
```

Команда импортирует объект из файла и помещает его в поле типа GENERAL текущей записи.

Для замены данных в поле GENERAL используйте APPEND GENERAL с фразой DATA.

```
CREATE TABLE MYTABLE (NAME M, MYOLEOBJ G)
NFILES=ADIR(AFILES,'*.HTM')
IF NFILES>0
        FOR i=1 TO NFILES
                APPEND BLANK
                REPLACE MYTABLE.NAME WITH AFILES(i,1)
                APPEND GENERAL MYOLEOBJ FROM AFILES(i,1)
        ENDFOR
ELSE
        MESSAGEBOX('FILE .HTM NOT FOUND')
ENDIF
```

Для добавления объекта на форму, в процессе проектирования формы, используется объект ActiveX Bound Control. Кнопка **OLE Bound Control** имеется на панели **Form Controls** и выглядит так:  .

Управляющий элемент ActiveX Bound Control — это элемент управления на форме, в котором отображаются данные из поля GENERAL таблицы. В частности, этот элемент можно встроить в Grid и сопроводить, таким образом, записи иллюстрациями. Вы можете создать на форме связанный OLE-объект, нажав на кнопку **ActiveX Bound Control**, с последующим перетаскиванием объекта в окно формы для установки размера. После создания объекта вы можете присоединить к нему поле типа GENERAL из какой-либо таблицы (используйте для этого свойство RecordSource). Затем вы можете использовать объект для отображения содержимого этого поля. Например, если вы храните в поле GENERAL документы Word, вы можете отобразить содержание этих документов, используя связанный ActiveX-объект на форме.

Программное отображение OLE-объекта

Для программного отображения на форме OLE-объекта, содержащегося в поле GENERAL, следует выполнить следующие действия:

1. Создаем форму

   ```
   ofrm=CREATEOBJECT('FORM')
   ```

2. Добавляем в форму объект ActiveXBound

   ```
   ofrm=ADDOBJECT('AXBC','BOUNDCONTROL')
   ```

3. Устанавливаем для объекта ActiveX Bound значение свойства ControlSource

```
ofrm.AXBC.controlsource='mydbc.mytable'
```

4. Отображаем саму форму и объект ActiveXBound

```
ofrm.AXBC.visible=.T.
ofrm.visible=.T.
```

Объектно-ориентированное программирование и ActiveX-объекты

Вы можете использовать возможности объектно-ориентированного программирования при работе с OLE-объектами. Для создания ActiveX-объекта используется команда CREATEOBJECT.

Пример. Работа с Excel

```
oleApp=CREATEOBJECT('EXCEL.APPLICATION')
oleAPP.VISIBLE=.T.
oleAPP.WORKBOOKS.ADD
oleAPP.CELLS(1,1).VALUE=7
oleAPP.ACTIVEWORKBOOK.SAVEAS('MY.XLS')
oleAPP.QUIT
```

Для объекта (OLE-клиента) сервер может получать и передавать массивы. Передача должна происходить по ссылке. То есть перед именем массива, указываемого в качестве параметра, следует ввести символ @.

При использовании элемента управления ActiveX Control в качестве контейнера, содержащего OLE-объект, для ссылки на свойство объекта нужно перед именем свойства включать Object, например:

```
form1.olecontrol1.object.right=50.
```

Методы *ActiveX Bound Control* для управления приложением, связанным с документом

Поскольку элементы управления ActiveX Bound Control могут быть использованы в средствах разработки, они обладают набором свойств и методов, позволяющим разработчикам управлять приложением.

Для расположенного в форме OLE-объекта можно дополнительно установить значения свойств AUTOACTIVATE и AUTOVERBMENU. Например, чтобы разрешить доступ к OLE-объекту только программно, посредством метода DoVerb, следует установить значение свойства AutoActivate равным 0, а значение свойства AutoVerbMenu — равным .F.

Примечание

Метод DoVerb(0) — выполнение действия по умолчанию.

Метод DoVerb(1) — активизация объекта для визуального редактирования.

Метод DoVerb(2) — открытие объекта в отдельном окне.