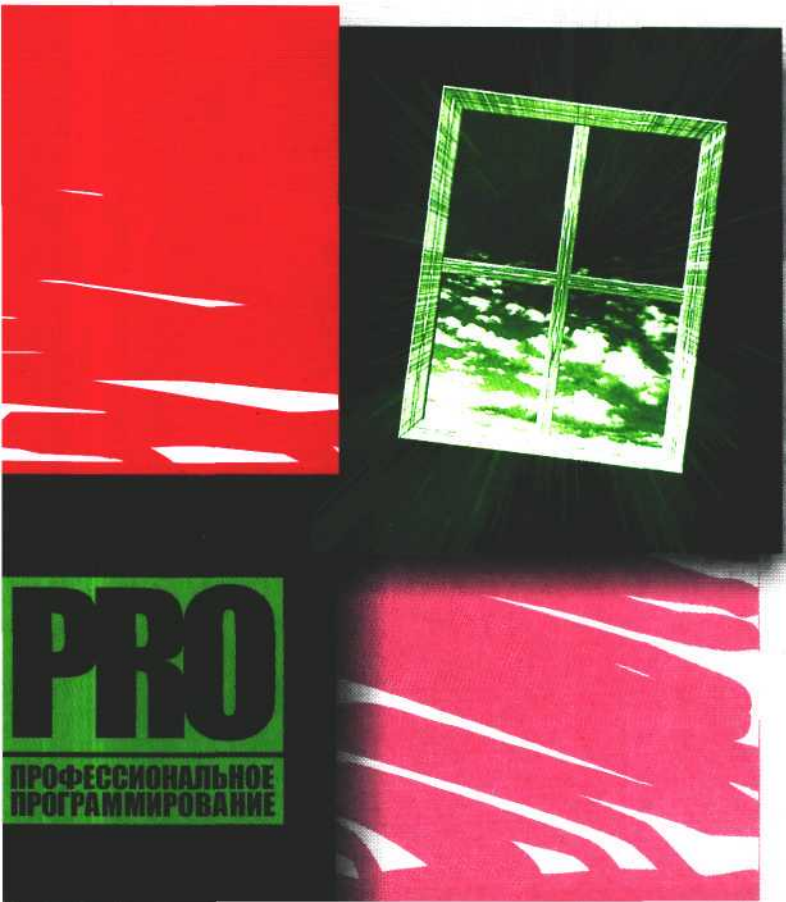
ВАСИЛИЙ КОРНЯКОВ

bhv

ПРОГРАММИРОВАНИЕ ДОКУМЕНТОВ И ПРИЛОЖЕНИЙ MS OFFICE В DELPHI



PRO
ПРОФЕССИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ

- РАЗРАБОТКА ПРИЛОЖЕНИЙ ДЛЯ РАБОТЫ С ТЕКСТОМ, ТАБЛИЦАМИ, РИСУНКАМИ И ВНЕШНИМИ ОБЪЕКТАМИ В ДОКУМЕНТАХ WORD И EXCEL
- ПРОГРАММИРОВАНИЕ ПАНЕЛЕЙ, МЕНЮ И ДРУГИХ ЭЛЕМЕНТОВ УПРАВЛЕНИЯ ПРИЛОЖЕНИЙ MS OFFICE
- РАЗРАБОТКА И ИСПОЛЬЗОВАНИЕ ДИНАМИЧЕСКИХ БИБЛИОТЕК ДЛЯ РАБОТЫ С ДОКУМЕНТАМИ WORD И EXCEL

+CD

Василий Корняков

ПРОГРАММИРОВАНИЕ ДОКУМЕНТОВ И ПРИЛОЖЕНИЙ MS OFFICE В DELPHI

Санкт-Петербург
«БХВ-Петербург»
2005

УДК 681.3.06
ББК 32.973.26-018.2
К67

Корняков В. Н.

К67 Программирование документов и приложений MS Office в Delphi. — СПб.: БХВ-Петербург, 2005. — 496 с.: ил.

ISBN 5-94157-458-4

Книга посвящена созданию приложений в среде Delphi для работы с текстом, таблицами, объектами, диаграммами, макросами, настройками параметров страниц и др. в документах MS Word и Excel. Большое внимание уделено объектным моделям документов MS Office. Рассмотрено программирование элементов управления редакторов MS Word и Excel, а также создание динамических библиотек, которые можно использовать в макросах документов MS Office. Приведены варианты программной реализации типовых задач и ответы на типовые вопросы, с которыми встречаются разработчики. На прилагаемом к книге компакт-диске размещены полные исходные тексты примеров из книги.

Для программистов

УДК 681.3.06
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Татьяна Темкина*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 24.12.04.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 40.
Тираж 3000 экз. Заказ № 723
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.02.953.Д.006421.11.04 от 11.11.2004 г. выдано Федеральной службой
по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 5-94157-458-4

Содержание

От автора

Выражаю благодарность всем, кто ждал эту книгу и вселял в меня силу и уверенность, — в основном это читатели моих статей в компьютерном издании "Компьютерные Вести" (**http://www.kv.by**). Их число не так велико (до 10 000), но благодаря их активности я рискнул продолжить работу уже в виде книги. Неслучайно один из разделов посвящен ответам на наиболее частые вопросы моих читателей.

Отдельно благодарю первых читателей и критиков и одновременно коллег по работе — Ольгу Нетребину и особенно Ивана Ивановича Бардаша, который, являясь специалистом высокого класса, всегда готов поделиться опытом и знаниями, высказать свое мнение и оказать бескорыстную помощь как начинающим, так и более опытным специалистам.

В период работы над книгой я познакомился с замечательным коллективом редакции издательства "БХВ-Петербург", который обладает удивительными, почти волшебными способностями и делает замечательное дело. Этот коллектив, работая над рукописью совместно с автором, превращает ее в интересную и полезную для читателей книгу. Выражаю признательность директору издательства Сергееву Вадиму Александровичу и главному редактору Кондуковой Екатерине Владимировне за понимание, поддержку и политику редакции, позволяющую потенциальным авторам создавать книги, а также зам. главного редактора Шишигину Игорю Владимировичу за терпение, настойчивость и профессиональный подход, помогающие автору правильно расставить акценты и рассчитать силы при подготовке материалов книги. Особую благодарность выражаю редактору Темкиной Татьяне Анатольевне, благодаря профессионализму и усилиям которой работа над книгой была завершена. Я также благодарю всех сотрудников издательства "БХВ-Петербург", отвечающих за верстку, корректуру и оформление книги.

Читателей прошу рассматривать эту книгу как инструмент, отправную точку для дальнейшего самостоятельного развития своих навыков в этой области программирования.

Корняков Василий Николаевич
www.kornjakov.ru
vasily@kornjakov.ru

Введение

Разрабатывая программные продукты, мы по-разному, в большей или меньшей степени задумываемся и представляем себе то, как с ними будет работать пользователь. Но если вообще не учитывать это обстоятельство, забыв о человеке, который будет работать с нашей программой, или пытаться навязать ему свои представления, то нас ждут неудачи и разочарование. Пользователь нас не поймет и не оценит наши усилия и интеллект, потому что мы просто не смогли их проявить. Главное — дать человеку почувствовать уверенность в собственных силах и то, что программный продукт был создан именно для него.

Несомненно то, что в отношениях пользователя с программой есть противоречия, и совершенно ясно, что проблема интерфейса "программа—пользователь" отчасти определяет эти противоречия и может быть решающей. Известно, что при создании приложений, с которыми будут общаться пользователи, разработка интерфейса требует больших затрат, особого подхода, опыта и знаний. Где взять опыт, который так необходим для построения взаимодействия человека и программы? Можно накопить и применять собственный опыт, а можно пойти другим путем и использовать существующие инструменты построения интерфейса, которые являются общеизвестными и общедоступными.

В данной книге речь пойдет об использовании программных продуктов, которые можно применять в качестве инструментов построения интерфейса. Такими инструментами могут служить программные продукты фирмы Microsoft.

- Приложения MS Office вобрали в себя богатый опыт взаимодействия пользователя персонального компьютера и программного продукта и являются универсальными инструментами, которые могут быть интегрированы в приложения, создаваемые в различных средах разработки и служить тем связывающим элементом, который может положить начало успешному внедрению ваших разработок.

- Подавляющее большинство пользователей изучили MS Word и Excel и работают с этими приложениями, не помышляя переходить к другим.
- Многие справочные и правовые системы обладают обширными наборами различных шаблонов документов, выполненных в формате документов Word и Excel, и их можно и нужно использовать.
- Документы MS Word и Excel, как и приложения MS Word и Excel, имеют четкие объектные модели и могут управляться как внутренними, так и внешними программами.

Эти факты служат аргументами в пользу того, чтобы использовать данные продукты в качестве универсального интерфейса для разрабатываемых программных продуктов.

В этой книге речь пойдет об использовании приложений из состава MS Office, а именно — процессора электронных таблиц MS Excel и текстового процессора MS Word, для формирования выходных документов (например, отчетов), экспорта и импорта информации и т. д. в приложениях, разрабатываемых в среде визуального программирования Delphi.

Delphi — один из самых популярных и эффективных инструментов разработки сложных приложений, и возможность интегрировать средства приложений MS Office в разрабатываемые в этой среде проекты — бесспорное тому доказательство.

В настоящее время есть достаточно много разнородной, отрывочной и разобщенной информации по использованию MS Word и MS Excel в приложениях Delphi. В основном эти сильно размытые сведения существуют в виде интернет-страниц в различных каталогах для программистов. Эта книга — попытка создать единый источник с целостным представлением, а также показать пути и методы решения задач, возникающих в связи с использованием Word и Excel в разрабатываемых приложениях.

Книга состоит из четырех частей.

Часть I посвящена обзору объектных моделей документов и приложений MS Office, особенностям встроенного языка Visual Basic. В ней также рассмотрены инструменты и общие принципы создания контроллеров автоматизации MS Office.

Часть II книги полностью посвящена разработке в среде Delphi контроллеров автоматизации приложения и документов MS Word. В этой части рассмотрены практические вопросы работы с текстом документа Word, создание простого текстового документа по шаблону и методы, используемые для заполнения шаблона. Отдельные главы посвящены работе с таблицами и созданию табличных документов, работе со шрифтом, с внутренними и внешними объектами документа Word. Рассмотрено использование процедур встроенного языка WordBasic — основного встроенного инструмента ранних версий MS Word. Приведены примеры его использования для соз-

дания документов Word из приложений Delphi. Также в этой части описаны объекты самого приложения MS Word с их свойствами и методами на основе практических примеров программирования меню, панелей управления, кнопок. На отдельных примерах рассмотрен доступ к макросам из приложений Delphi.

Часть III книги посвящена разработке в среде Delphi контроллеров автоматизации MS Excel, основы создания которых приведены в *главе 11*. На практических примерах рассматриваются объекты и методы, обеспечивающие доступ к рабочим книгам и листам MS Excel из приложений, создаваемых в Delphi. В *главе 12* описаны объекты и методы, обеспечивающие доступ к отдельным ячейкам и к областям ячеек листа рабочей книги. На основе этой информации разбирается пример программы, реализующей заполнение налоговой декларации на основе подготовленного шаблона документа. *Глава 13* посвящена внутренним объектам (надписи, выноски, геометрические фигуры). В ней рассматриваются все вопросы, связанные с созданием этих объектов и настройкой их свойств из приложений Delphi. Отдельные главы третьей части посвящены детальному рассмотрению объектной модели диаграммы и программированию ее свойств, а также настройке параметров страницы, настройкам принтера и печати документов. В заключение третьей части рассматриваются объектные модели элементов управления приложения MS Excel — панели, меню, кнопки, их создание и программирование их свойств; описан доступ из приложений Delphi к элементам проектов, созданных с использованием встроенного языка Visual Basic, которые могут входить в состав рабочих книг MS Excel. Описание работы всех объектов и методов подкрепляется фрагментами исходного текста, который можно использовать в своих приложениях.

Часть IV книги посвящена динамическим библиотекам. Рассматривается создание и использование динамических библиотек, содержащих процедуры и функции: для работы с документами MS Office из приложений Delphi; для использования в макросах документов Word и Excel. В *главе 17* описана структура исходных файлов таких библиотек и рассмотрены варианты и примеры их явной и неявной загрузки. В *главе 18* на примере использования двух функций, одна из которых преобразует числовое значение переменной в его текстовый эквивалент, а другая отображает диалоговое окно и возвращает введенные в нем пользователем значения в документ, продемонстрировано использование разрабатываемых динамических библиотек в документах MS Office.

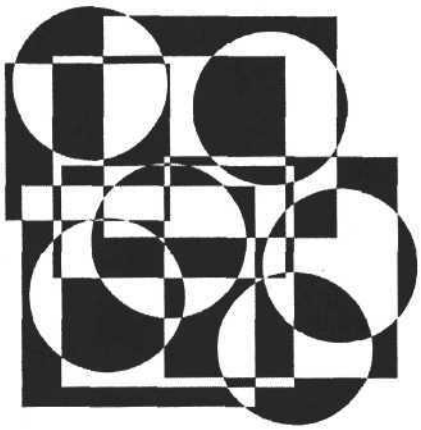
Книга содержит три приложения. *Приложение 1* является справочником по рассматриваемым в книге объектам, их свойствам и методам. *Приложение 2* посвящено ответам на типичные вопросы, возникающие при работе с документами MS Office в среде Delphi. В *Приложении 3* описан сопроводительный компакт-диск книги.

В *Списке литературы* указаны дополнительные источники информации.

Предметный указатель способствует быстрому поиску нужного термина.

Книга построена по принципу "от простого к сложному", содержит много примеров, позволяющих быстро использовать полученные знания. Она адресована разработчикам программного обеспечения, которые работают в среде Delphi, имеют достаточный опыт работы и стремятся к получению дополнительных знаний и совершенствованию опыта работы. Книга полезна и начинающим программистам, знакомым с основами работы в среде программирования Delphi. Знание и опыт использования встроенного языка Visual Basic приложений MS Office ускорят обучение и помогут достичь лучших результатов.

Книга сопровождается компакт-диском, на котором отдельно для каждой главы размещены полные исходные тексты примеров. Все примеры выполнены в виде законченных проектов, которые можно откомпилировать и запустить на выполнение. Каждый проект размещен в отдельной папке и устанавливается на жесткий диск компьютера путем выбора определенного пункта меню загрузочной программы. На сопроводительном компакт-диске также размещены библиотеки, которые вы можете использовать в своих проектах, и несколько проектов, предназначенных для формирования бухгалтерских документов в виде документов приложений Word и Excel. Это проект "Платежные документы" и проект с отлаженным фрагментом программы формирования налоговой декларации в формате Excel. Все предложенные на сопроводительном компакт-диске примеры выполнены таким образом, чтобы их исходный текст можно было легко, без существенных изменений и временны́х затрат перенести в любой разрабатываемый вами проект.



ЧАСТЬ I

Основы и общие принципы разработки документов и приложений MS Office из внешних программ

ГЛАВА 1

Объектные модели MS Office

Если предстоит решать сложную задачу, мы всегда задумываемся о том, с чего начать и как подойти к ее решению, как правильно распределить время и как сохранить при этом силы. Если перед нами стоит задача автоматизации разработки документов в формате MS Word или MS Excel, то с чего можно было бы начать?

Текстовый процессор MS Word и табличный процессор MS Excel, входящие в состав MS Office, являются COM-объектами. Это означает, что любая программа, написанная для современных версий Windows, может управлять этими объектами, если она поддерживает интерфейс COM. Тонкости механизма работы этого замечательного интерфейса описаны в специальной литературе, а мы займемся объектами Word, Excel и документами, с которыми они работают, поскольку они также являются объектами.

Ответ на вопрос "с чего начать?" напрашивается сам собой — начать нужно с рассмотрения структуры моделей объектов MS Office, взаимосвязей их составных частей, особенностей встроенного языка программирования, подойти к рассмотрению общих принципов автоматизации MS Office и разобраться, как этим можно управлять из программ, написанных в среде Delphi.

Рассмотрим структуру объектных моделей MS Office.

Объектная модель MS Word

Как видно из рис. 1.1, объектная модель MS Word имеет древовидную структуру, на вершине которой находится объект Application. Коллекции (Collection), входящие в состав объектной модели, представляют собой списки однотипных объектов, доступ к которым осуществляется по индексу. Отдельные объекты могут содержать коллекции, в свою очередь элементы

коллекции являются обычными объектами. Доступ к любому объекту или элементу коллекции возможен только через корневой объект Application. На рис. 1.1 объектная модель изображена в укрупненном виде. Можно выделить и более детально рассмотреть некоторые части этой модели, например, элемент коллекции Documents — объект Document (документ). На рис. 1.2 представлена структура объекта Document в том виде, который дает представление о его основных элементах и их взаимосвязи.

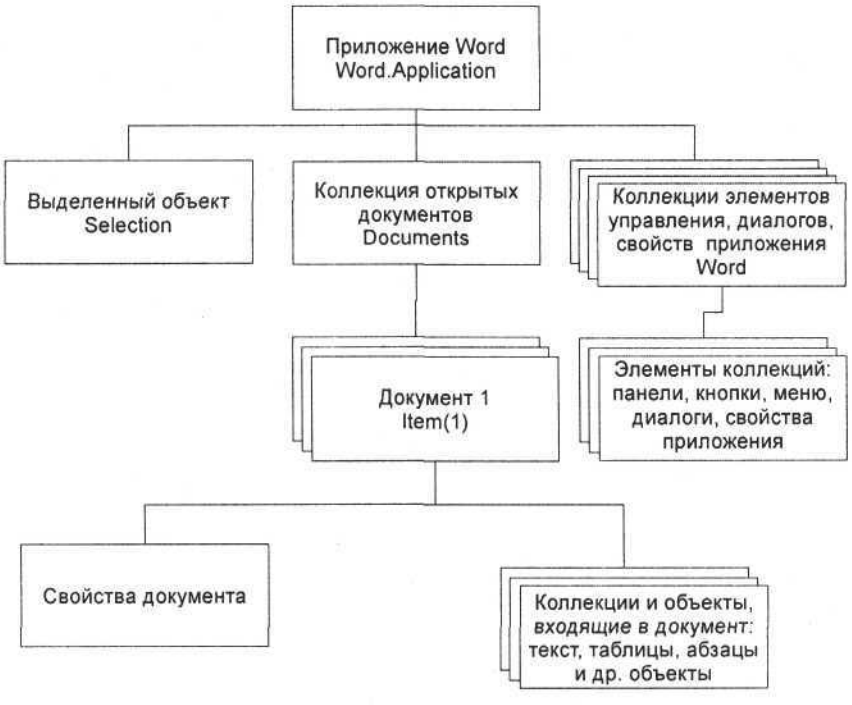


Рис. 1.1. Объектная модель MS Word

В состав объекта Document входят как элементы, описывающие его в целом, так и элементы, являющиеся содержанием документа. Например, коллекция свойств, описывающих пользовательские характеристики документа (название, тема, автор и др.), относится к документу в целом и визуально не отображается в документе. Коллекция таблиц — элемент, который является частью содержания документа и отображается на его страницах. В свою очередь элемент коллекции может содержать свои объекты и коллекции. Например, каждая таблица содержит коллекции строк и столбцов, а также элементы типа "ячейка". Коллекции могут включать не только полностью однотипные элементы, но и элементы, которые могут существенно отличаться друг от друга. Коллекция форм (Shapes) содержит элементы, однотипные по способу создания и размещения, а по содержанию — различные.

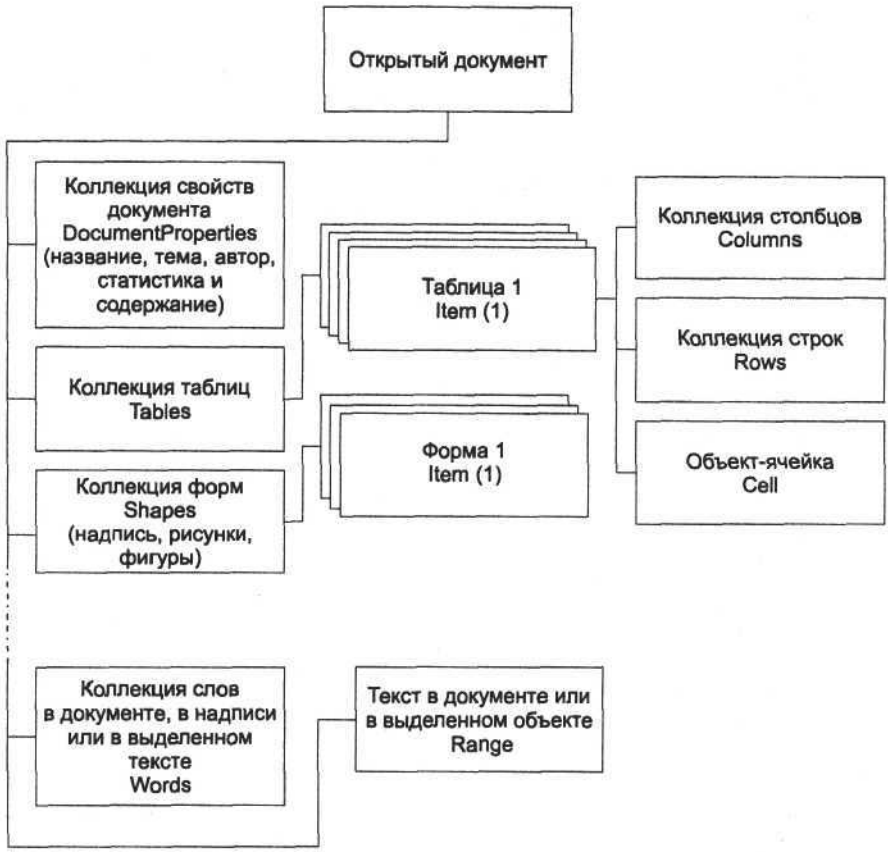


Рис. 1.2. Объектная модель документа MS Word

Элемент коллекции Shapes может содержать надпись (Textbox), внешний объект (OLEObject), линию (Line), полилинию (Polyline) и другие типы объектов. Основной объект документа это, конечно, текст. Для доступа к тексту можно использовать коллекцию Words или объект Range. Элементы коллекции Words — объекты, которые отображают слова в тексте и содержат их характеристики. Объект Range описывает текст или часть текста целиком. Конечно, объектная модель документа на рис. 1.2, описывает его далеко не полностью. Полезно рассмотреть еще и объектную модель элементов управления, объект "шрифт" *(см. главу 7)* и объект, который описывает параметры страницы *(см. главу 8)*. На рис. 1.3 приведен общий вид структуры объектной модели элементов управления MS Word.

Как видно из рис. 1.3, все элементы управления MS Word собраны в одну коллекцию, имеющую древовидную структуру, каждый узел которой может содержать как объекты, так и вложенные коллекции. Эту структуру можно настраивать и дополнять. Пользователь и программист, используя опреде-

ленные инструменты, могут добавить, удалить, модифицировать или активизировать любой узел или объект этой структуры. При этом будут выполнены методы, соответствующие активизированным элементам управления.

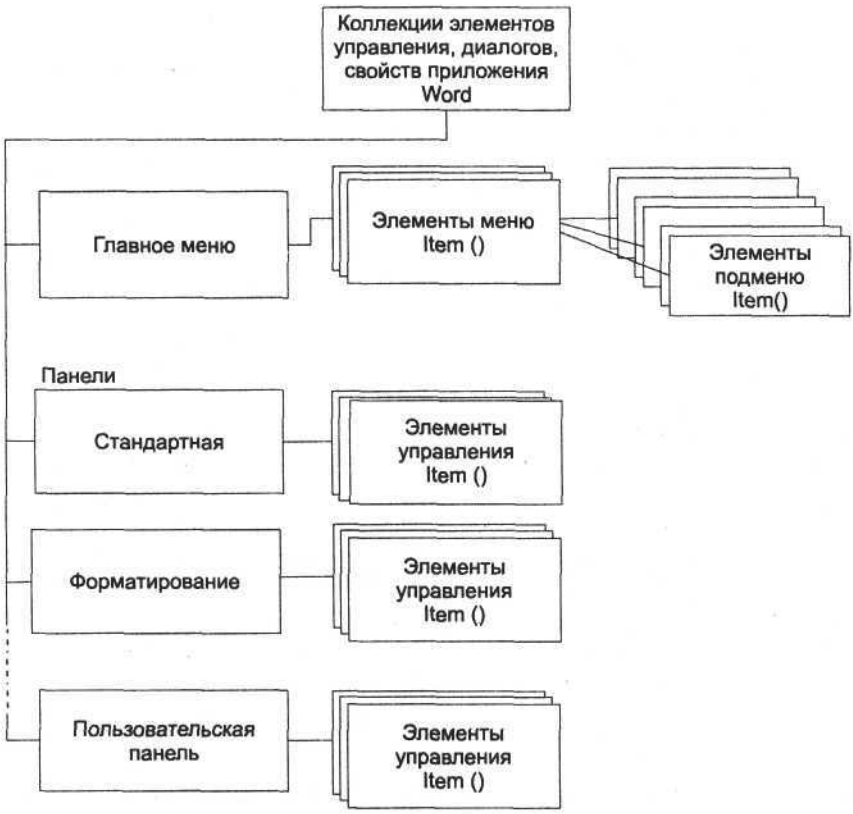


Рис. 1.3. Объектная модель элементов управления MS Word

Мы рассмотрели основу объектной модели MS Word, представление о которой поможет нам при создании приложений.

Объектная модель MS Excel

Объектная модель MS Excel по общим принципам идентична объектной модели MS Word. Эта модель также имеет иерархическую структуру, в корне которой находится объект Application (Excel.Application), через который обеспечивается доступ к любой коллекции или внутреннему объекту приложения MS Excel или к компонентам открытых рабочих книг. Общая структура объектной модели MS Excel представлена на рис. 1.4.

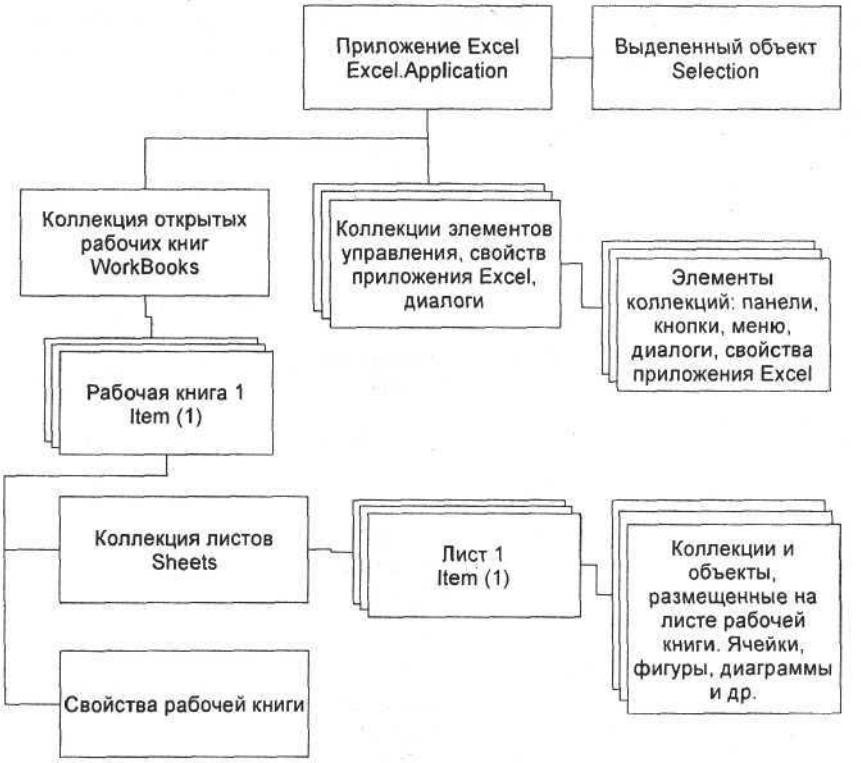


Рис. 1.4. Объектная модель MS Excel

Как уже сказано, вершиной объектной модели MS Excel является объект Application, непосредственно включающий такие объекты и коллекции, как Selection — текущий выделенный объект, WorkBooks — коллекция открытых рабочих книг, коллекции различных элементов управления, диалоговых окон и другие свойства приложения MS Excel. Объект Selection имеет свойства текущего выделенного объекта, поэтому нет смысла рассматривать здесь структуру модели этого объекта. Если выделена ячейка, то Selection = Ячейка, если диаграмма, то Selection = Диаграмма. Когда выделена область ячеек, Selection = Коллекции столбцов и строк. Коллекция рабочих книг представляет собой список открытых рабочих книг, доступ к которым осуществляется по индексу. Каждый элемент такой коллекции представляет собой объект "рабочая книга". Панели управления, меню, кнопки, диалоги и другие объекты, предназначенные для управления приложением, собраны в соответствующие коллекции элементов управления.

Основным элементом объекта "рабочая книга" является коллекция рабочих листов. Элементом коллекции является отдельный рабочий лист, который может представлять собой обычный лист (в виде таблицы) или быть в виде

диаграммы. В последнем случае он ничего, кроме диаграммы, содержать не может. Рассмотрим вариант листа в виде таблицы (рис. 1.5).

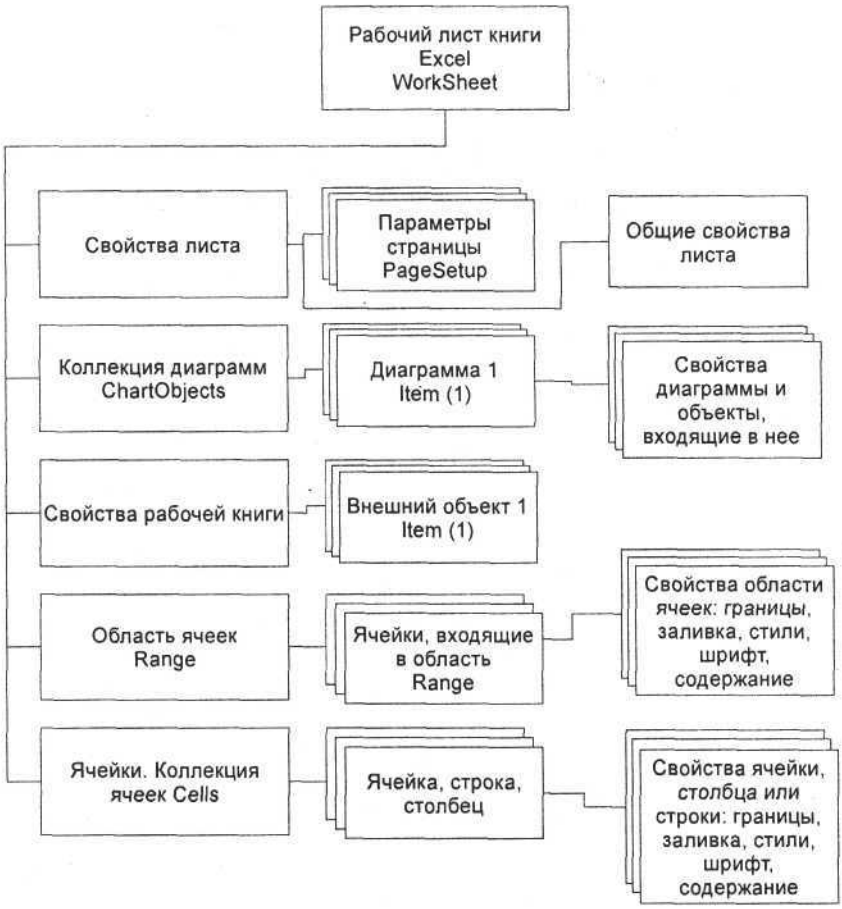


Рис. 1.5. Объектная модель листа рабочей книги

Основной объект рабочего листа, с которым приходится работать, — ячейка. Ячейка как объект сама обладает множеством свойств и объектов, входящих в нее. Из них наиболее важными и часто используемыми являются: текст, шрифт, стиль текста, границы, заливка. Чтобы получить доступ к ним, необходимо получить доступ к самой ячейке, а затем изменять ее свойства. Ячейки объединены в области ячеек Range. Свойства области ячеек во многом совпадают со свойствами самой ячейки, но есть и отличия, состоящие в задании координат и размеров области. Ячейки объединены в строки и столбцы. Объединения строк и столбцов представляют собой коллекции, доступ к которым производится по числовому индексу или по буквенному обозначению столбца. На рабочем листе могут располагаться внешние объ-

екты: рисунки, фрагменты документов Word, звуки, видеозаписи и другие объекты, которые объединены в коллекцию внешних OLE-объектов. Прорисовка или, точнее, воспроизведение этих объектов полностью выполняется внешними программами, зарегистрированными в системе как OLE-серверы. Доступ к таким объектам производится через элементы коллекции OLEObjects, а доступ к их свойствам возможен только через эти OLE-серверы. Приложение Excel обладает большим набором собственных графических объектов, которые можно разместить на рабочем листе. Мы можем использовать рисунки, надписи, геометрические фигуры, диаграммы, которые обычно объединены в коллекции. Например, коллекция ChartObjects содержит набор диаграмм, которые располагаются на рабочем или на отдельном листе. Каждая диаграмма, в свою очередь, также содержит набор объектов и коллекций. Рассмотрение свойств, состава и содержания диаграмм — отдельная тема этой книги *(см. главу 14)*.

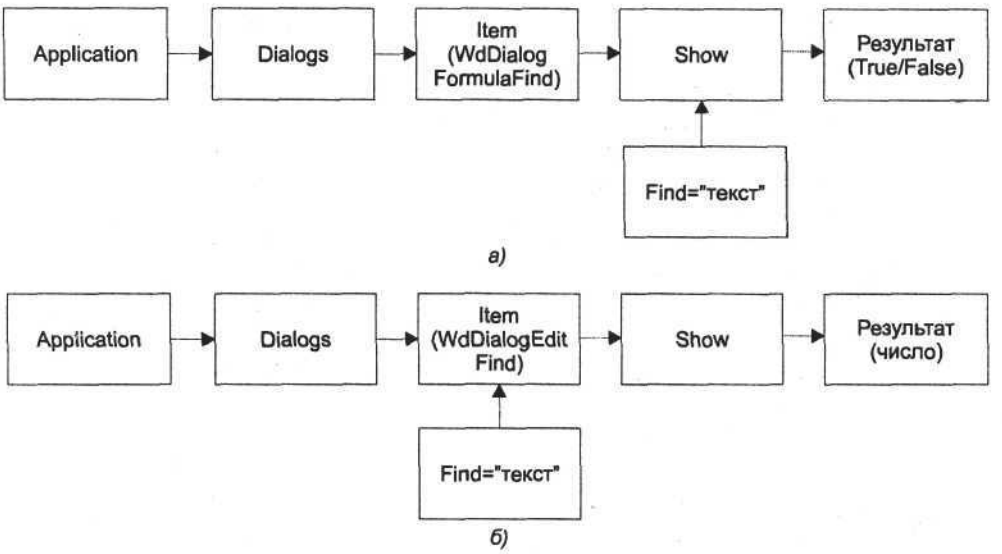


Рис. 1.6. Объектные модели диалогов **Найти** в Excel (*а*) и **Найти и заменить** в Word (*б*)

Чтобы убедиться в гибкости, универсальности и больших возможностях для программирования объектов MS Office, рассмотрим еще одну коллекцию объектов, присутствующую как в Word, так и в Excel. Это коллекция диалогов (диалоговых окон), которые пользователь обычно открывает нажатием той или иной кнопки или выбором команды меню. Она принадлежит объекту Application. В объектной модели все диалоги представлены в виде элементов коллекции Dialogs, доступ к которым обеспечивается через числовой индекс. Посредством параметров метода Show элемента коллекции происходят передача параметров в диалог и его выполнение — такова модель диалогов для приложений Excel (рис. 1.6, *а*), для приложений Word модель

диалога несколько отличается. Отличие заключается в том, что в диалог Word параметры передаются через свойства объекта-элемента коллекции (рис. 1.6, *б*).

У объекта Item() наряду с типичными свойствами и методами есть присущие только ему свойства и методы. Например, у диалога **Найти и заменить** есть свойство Find, определяющее текст для поиска — до запуска диалога.

В Excel объект коллекции Dialogs несколько отличается от диалогов Word. Здесь рассмотрим только объектную модель коллекции диалогов для Excel в целом (рис. 1.7).

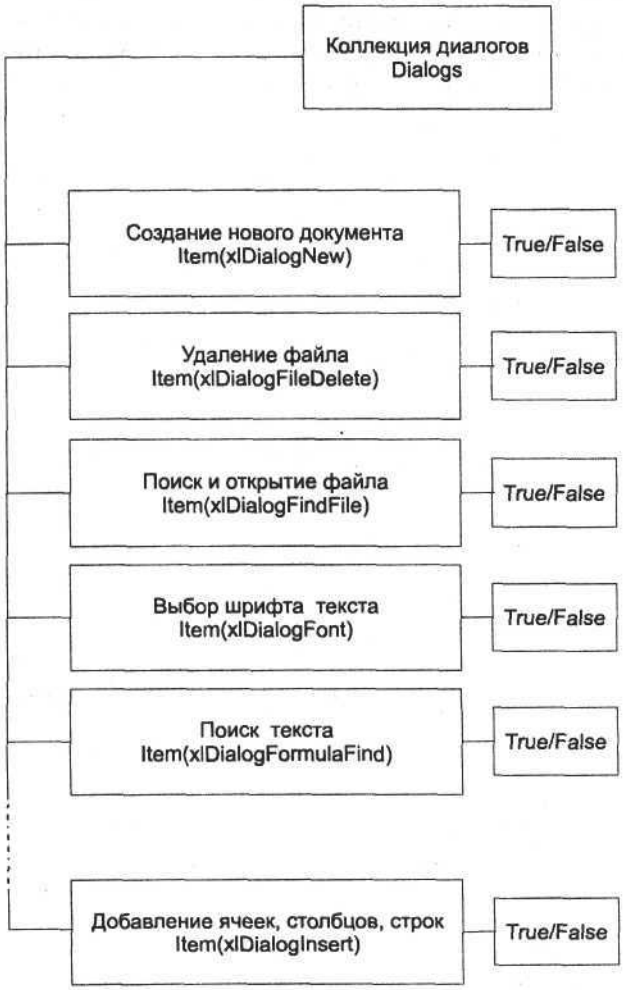
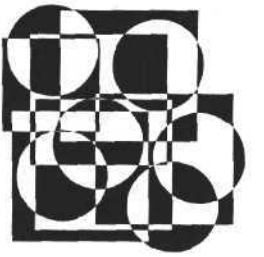


Рис. 1.7. Объектная модель коллекции диалогов Excel

На этом рисунке представлена только малая часть всевозможных диалогов коллекции — показаны диалоги, наиболее часто используемые программистами и пользователями. Обычно в диалог передаются значения, которые устанавливают значения элементов управления, а возвращаются значения True или False, в зависимости от результата действия пользователя.

Можно подвести итог: приложения из состава MS Office, например Word и Excel, представляют собой взаимосвязанные объекты и коллекции объектов. Каждый объект или коллекция включает в себя множество параметров и других объектов, коллекций. В свою очередь, сами объекты и коллекции имеют хозяина (Parent), в состав которого они входят. На вершине объектной модели находятся объекты Word.Application для текстового процессора Word и Excel.Application — для табличного прцессора Excel. Через эти объекты, в основном, и осуществляется связь этих приложений с внешними программами. Как все это работает, вы узнаете из следующих глав книги.

ГЛАВА 2



Особенности встроенного языка программирования MS Office

В состав приложений MS Word и MS Excel входит язык программирования Visual Basic. Он появился не сразу — в первых версиях MS Office применялся язык написания макросов, который постепенно развился в полноценный язык, позволяющий разрабатывать в среде MS Office довольно сложные приложения. Чтобы разрабатывать приложения Delphi, использующие контроллеры автоматизации, необходимо знать некоторые особенности встроенного языка MS Office. Для начала можно ознакомиться с ними, используя справочную систему, где он достаточно хорошо описан (рис. 2.1). Справку по Visual Basic можно установить, используя установочный диск MS Office.

Однако для оперативного создания и отладки некоторой части программы справки может быть недостаточно или в ней не найдется подходящего примера. Как поступить в этом случае? Для решения таких вопросов в Word и Excel предусмотрен режим автоматической записи действий пользователя в формате встроенного языка. Записывая таким способом части текста программы, анализируя и изменяя их, мы существенно сокращаем время на разработку приложений. Рассмотрим, как это делается, на простом примере: в документе Word создадим таблицу и заполним текстом несколько ячеек. Для начала включим режим записи макросов, выполнив команду **Сервис > Макрос > Начать запись** главного меню (рис. 2.2, *а*) или нажав кнопку **Записать макрос** на панели **Visual Basic** (рис. 2.2, *б*).

Откроется диалоговое окно, в котором нужно ввести имя макроса, определить документ, для которого он будет доступен, и задать его описание. Назовем наш макрос **Макрос1** и приступим к работе с документом — как это обычно делают пользователи, используя команды меню и кнопки панелей инструментов. После того как таблица создана и заполнено несколько ячеек, выключим режим записи макроса (рис. 2.2, *в*) и откроем окно редактора Visual Basic с помощью пункта **Сервис > Макрос > Редактор Visual Basic** главного меню или кнопки на панели **Visual Basic**.

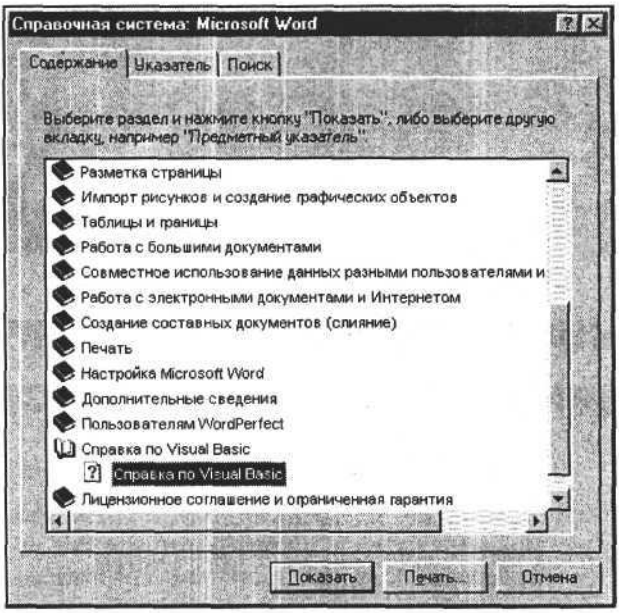
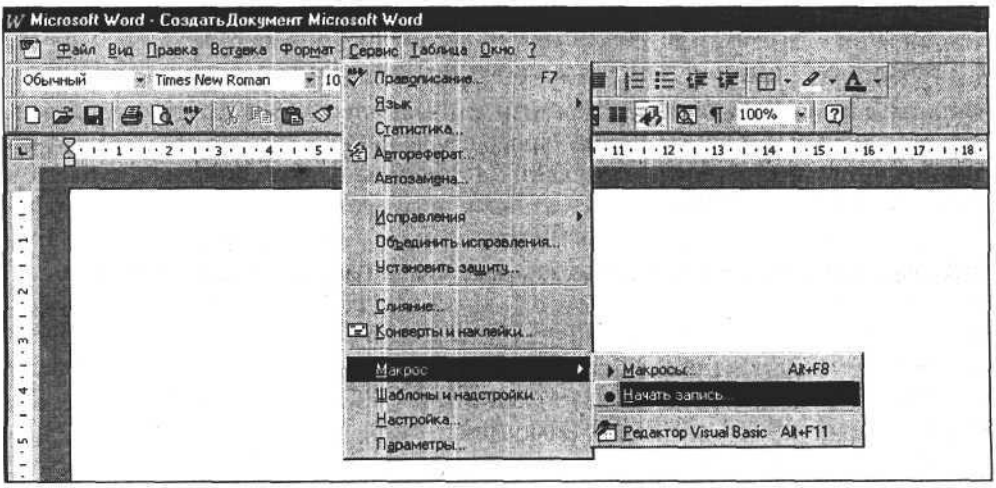


Рис. 2.1. Доступ к справке по Visual Basic для Word



а)

б)

в)

Рис. 2.2. Включение (*а*, *б*) и выключение (*в*) режима записи макроса

В окне редактора Visual Basic мы увидим примерно такой программный текст на встроенном языке (рис. 2.3), операторы которого повторяют и описывают действия пользователя.

```
Sub Макрос1()
'
' Макрос1 Макрос
' Макрос записан 02.01.04 Корняков Василий Николаевич
'
    ActiveDocument.Tables.Add Range:=Selection.Range, NumRows:=2, NumColumns:=4
    Selection.PageSetup.TopMargin = CentimetersToPoints(1.75)
    Selection.Tables(1).Rows.SetLeftIndent LeftIndent:=55.05, RulerStyle:=wdAdjustNone
    ActiveWindow.ActivePane.LargeScroll ToRight:=1
    ActiveWindow.ActivePane.HorizontalPercentScrolled = 0
    Selection.TypeText Text:="№ п/п"
    Selection.MoveRight Unit:=wdCell
    Selection.TypeText Text:="Товар"
    Selection.MoveRight Unit:=wdCell
    Selection.TypeText Text:="Цена"
    Selection.MoveRight Unit:=wdCell
    Selection.TypeText Text:="Количество"
    Selection.MoveRight Unit:=wdCell
    Selection.TypeText Text:="1"
End Sub
```

Рис. 2.3. Текст записанного макроса в окне редактора Visual Basic

Очевидно, что возможность автоматической записи макросов существенно облегчает понимание и сокращает время создания приложений в MS Office. Из синтаксиса операторов видно, что мы имеем дело с объектами и объектно-ориентированным языком.

К этому можно добавить, что для приложения Word встроенный язык Visual Basic имеет одну особенность. Он сам является объектом, и его процедуры и функции доступны из внешних программ аналогично тому, как это реализовано для объекта Application. Для доступа к Visual Basic используется идентификатор Word.Basic.

Обеспечение доступа к встроенному языку Visual Basic приложения Word

```
uses ComObj;
var B:variant;
procedure TForm1.Button1Click(Sender: TObject);
begin
B:=CreateOleObject('Word.Basic');
messagebox(handle,'Word.Basic - запущен','Внимание!',0);
B.FileQuit;
B:=UnAssigned;
end;
```

В данном примере мы просто загружаем объект Word.Basic и выгружаем его без какого-либо практического применения (конечно, его возможности этим не исчерпываются).

Объект Word.Basic можно также использовать для создания документов из приложений, написанных на Delphi, но все же лучше использовать объект Application, т. к. мы работаем с документом как с объектом, и дополнительное звено в цепи "приложение-объект "Документ Word"" может оказаться лишним и стать помехой. В конечном итоге, выбор пути решения поставленной задачи — дело самого программиста.

Другой важной особенностью встроенного языка MS Office является возможность использования в макросах внешних процедур и функций из динамических библиотек (DLL, Dynamic Link Library — библиотека динамической компоновки). Для этого достаточно описать спецификацию процедуры или функции с указанием ссылки на динамическую библиотеку. Использование внешних процедур и функций, написанных в том числе и в среде Delphi, позволяет интегрировать пользовательские приложения в документы Word и Excel и создавать сложные документы и отчеты на комплексной основе. Эта возможность важна, например, если отчеты создаются с помощью баз данных, с которыми Excel, Word и Visual Basic не работают или работают некорректно, или если вы хотите использовать эксклюзивные возможности среды программирования Delphi, или если просто требуется вызывать функции API Windows или любого приложения. На рис. 2.4 представлена структура объявления (описания) процедур и функций, импортируемых из динамических библиотек. Подробно этот материал будет рассмотрен в *главе 18.*

Здесь можно добавить только одно важное замечание о совместимости типов данных, используемых в качестве переменных процедур и функций. Типы данных, используемых во встроенном языке и в среде программирования Delphi, должны быть идентичны друг другу. Подробности можно узнать из справки по Visual Basic и Delphi.

После того как в общих чертах рассмотрены модели MS Word, MS Excel и особенности встроенного языка, можно перейти к рассмотрению общих принципов создания контроллеров автоматизации.

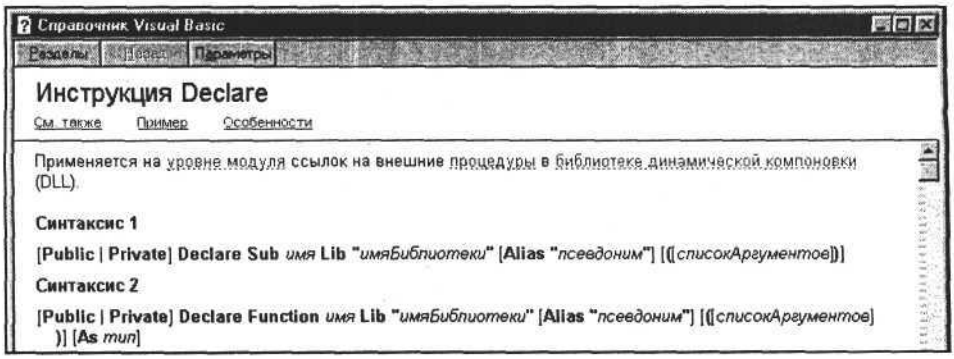


Рис. 2.4. Синтаксис описания внешних процедур и функций в макросах Word и Excel

ГЛАВА 3

Общие принципы создания контроллеров автоматизации MS Office

Одной из замечательных особенностей многозадачных операционных систем является поддержка взаимодействия и обмена информацией между различными программами. Операционная система (ОС) Windows — не исключение из этого правила и предоставляет множество механизмов такой поддержки. Работа ОС Windows предусматривает передачу и обработку сообщений как между ОС и приложением, так и между приложениями, а также использование динамических библиотек. Этот механизм был использован еще в первых версиях Windows и отразился на методах разработки приложений для этой ОС. Основой программирования в первых версиях Windows стало программирование объектов, потому что любая программа этой ОС представляла собой объект. С развитием операционной системы развивались и механизмы взаимодействия программ. Развитие шло от обмена сообщениями, использования DLL (Dynamic Link Library, динамически подключаемые библиотеки процедур и функций) и механизма DDE (Dynamic Data Exchange, динамический обмен данными) к современным технологиям, основанным на OLE (Object Linking and Embedding, связывание и внедрение объектов), COM (Component Object Model, компонентная модель объектов), DCOM (Distributed Component Object Model, распределенная компонентная модель объектов). Этот переход был обоснован тем, что стало недостаточным использование только функций и процедур, предоставляемых внешними программами и библиотеками. На первый план вышла необходимость управления целыми объектами, которые представляют собой приложения или документы и расположены как в одном адресном пространстве управляющей программы, так и вне этого пространства или даже на другом компьютере локальной сети. Такая постановка задачи повлекла за собой революционные изменения как в структуре ОС Windows, так и в программах, предназначенных для разработки приложений. Приложения MS Office представляют собой объекты-серверы, которые могут управляться внешними программами, и здесь не последнюю роль играют механизмы COM и OLE.

Модель COM предоставляет возможность создания многократно используемых объектов в различных приложениях, поддерживающих этот интерфейс. Объектами COM являются приложения-серверы, специальным образом оформленные и зарегистрированные в системе. Они могут быть представлены в формате EXE- или DLL-модулей. Эти серверы могут загружаться и выполняться как в адресном пространстве вызывающего приложения, так и в виде самостоятельного процесса, или на другом компьютере сети (распределенная модель COM — DCOM). Они должны быть написаны на любом языке, поддерживающем интерфейс COM.

Развитие технологии COM продолжает ее подмножество — технология OLE Automation (автоматизация OLE). Ее отличие в том, что она позволяет использовать возможности COM не только языкам-компиляторам, но и интерпретаторам, и обеспечивает связь с вызываемыми методами на стадии выполнения приложения. Такой способ вызова называется *поздним связыванием*. Методы при таком способе вызова выполняются медленнее, причем заранее нельзя проверить правильность написания объектов и их методов. Преимуществом такого метода является независимость выбора среды разработки от объекта, который нужно программировать.

Среда Delphi поддерживает вызовы методов серверов автоматизации.

Примечание

Сервер автоматизации представляет собой программу, которая может управляться внешней программой — *контроллером автоматизации*. Сервером в данном случае является Word или Excel, а контроллер разрабатывается программистом.

Для этого используются переменные типа Variant, которые содержат ссылки на объекты автоматизации. На этапе выполнения программы серверу автоматизации передается команда в виде строки, предварительно записанной в переменную типа Variant. Рассмотрим в качестве примера следующий фрагмент программы:

Загрузка Internet Explorer

```
Uses ComObj;
    Var IE:variant;

procedure TMainForm.Button1Click(Sender: TObject);
begin
    IE:=CreateOleObject('InternetExplorer.Application');
    IE.Visible:=true;    // Нужно выбрать один из операторов
    IE.Visible:=false;   // и проверить, как это работает
    ...
end;
```

Переменная IE типа Variant не имеет никаких свойств и методов, но, тем не менее, программа откомпилируется и будет выполняться. Если мы вместо свойства Visible напишем любое другое свойство, которое не поддерживает IE, то компилятор ошибки не выдаст, а на стадии выполнения произойдет ошибка. Это особенности позднего связывания — свойства и методы проверяются в самом приложении-сервере на стадии выполнения. Можно добавить, что весь этот механизм работает за счет COM API. В данной книге нет смысла подробно рассматривать вызовы его функций и процедур. Наша задача — научиться создавать и использовать контроллеры автоматизации (что это такое, рассказано чуть позже) для приложений MS Office Word и Excel, которые наиболее часто используются как приложения для пользователей и как серверы для создания приложений. Используя контроллеры автоматизации, в приложениях, созданных в среде Delphi, можно так же просто, как при обычной работе с Word и Excel, создавать документы со всеми возможными элементами. Таблицы, надписи, текст, диаграммы и другие компоненты полноценных документов появятся в таком виде, как будто они созданы пользователем. Далее в книге все примеры будут описывать в основном только работу с Word и Excel и позднее связывание.

Перейдем непосредственно к контроллерам автоматизации. Из уже сказанного понятно, что *контроллер автоматизации* — это программа, которая "умеет" управлять приложениями MS Office и процессом создания документов в среде Word и Excel. Для того чтобы все это работало корректно, программа-контроллер должна выполнить следующие функции:

1. Проверить, запущено приложение (Word, Excel) или нет.
2. Если приложение не запущено, запустить его.
3. Выполнить ряд необходимых манипуляций с приложением, документом.
4. Закрыть документ и приложение.
5. Очистить память.

Как уже сказано, доступ к документам и приложениям можно осуществлять через объекты, стоящие на вершине объектной модели приложений MS Office. Через них мы получаем доступ к внутренней структуре документов и приложений, поэтому при создании контроллеров автоматизации мы можем использовать только их. В табл. 3.1 и 3.2 приведены эти объекты и дана их краткая характеристика.

Таблица 3.1. *Объекты MS Word*

Идентификаторы	Краткая характеристика
Word.Application Word.Application.8 (9,10)	С помощью этих идентификаторов осуществляется доступ к объекту Application и запускается приложение Word

Таблица 3.1 *(окончание)*

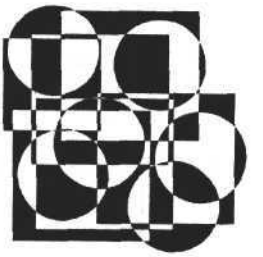
Идентификаторы	Краткая характеристика
Word.Document Word.Document.8 (9,10) Word.Template.8	С помощью этих идентификаторов осуществляется доступ к объекту Document, запускается приложение Word, если оно не было запущено, создается новый документ. Если до этого приложение Word было запущено, то новый документ создается в нем

Таблица 3.2. *Объекты MS Excel*

Идентификаторы	Краткая характеристика
Excel.Application Excel.Application.8 (9,10)	С помощью этих идентификаторов осуществляется доступ к объекту Application и запускается приложение Excel
Excel.AddIn	С помощью этого идентификатора запускается приложение Excel и создается специальный объект Add-In
Excel.Chart Excel.Chart.8	С помощью этих идентификаторов запускается приложение Excel и создается рабочая книга, содержащая диаграмму и рабочий лист с данными для диаграммы
Excel.Sheet Excel.Sheet.8	С помощью этих идентификаторов запускается приложение Excel и создается рабочая книга, содержащая рабочий лист

Мы подошли к вопросам создания контроллеров автоматизации в среде Delphi. В следующей главе мы рассмотрим инструменты среды Delphi.

ГЛАВА 4



Обзор инструментов среды разработки приложений Delphi для работы с MS Office

Среда разработки приложений Delphi предоставляет программистам массу возможностей по созданию приложений, способных взаимодействовать с внешними программами, такими как Word, Excel, Internet Explorer, Outlook и другими, использующими механизмы Windows для обмена данными. Даже ранние версии Delphi поддерживали технологию DDE и OLE. Начиная с пятой версии Delphi, возможности были дополнены целым набором компонентов для работы с приложениями MS Office. Благодаря странице **Servers** Палитры компонентов, разработчик получил возможность достаточно быстро создавать для своих приложений отчеты в формате Word, Excel, работать с почтовой программой и другими приложениями MS Office. Страница **Servers** содержит компоненты для работы с приложениями Word, Excel, документами и рабочими книгами и компоненты для работы с текстом. На рис. 4.1 показан внешний вид страницы **Servers**.

Рис. 4.1. Страница **Servers** — компоненты для работы с приложениями MS Office

Упомянем компонент, просто необходимый в приложениях, работающих с документами в Интернете. Это WebBrowser — замечательный компонент, обладающий всеми возможностями приложения Internet Explorer, потому что на самом деле это и есть Internet Explorer в виде внешнего компонента (рис. 4.2). В вашем приложении его можно разместить в форме, после чего у нее появятся все качества обозревателя Интернета.

Если вам нужно больше гибкости и универсальности, используйте компонент OleContainer со страницы **System** (рис. 4.3). Расположив этот компо-

нент в форме своего приложения, вы с легкостью сможете придать ему функции как приложения Word или Excel, так и любого другого приложения, зарегистрированного в системе как сервер OLE.

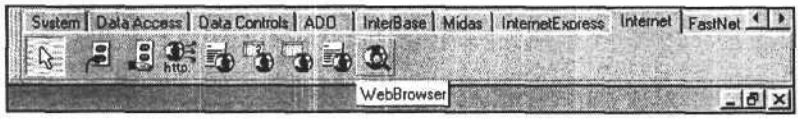


Рис. 4.2. Компонент WebBrowser на странице **Internet** Палитры компонентов

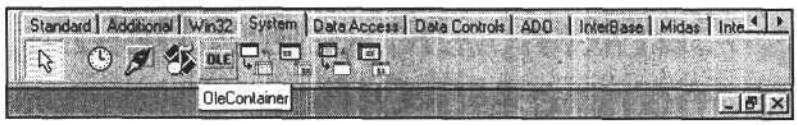


Рис. 4.3. Компонент OleContainer на странице **System** Палитры компонентов

Было бы несправедливо обойти вниманием набор компонентов для работы с базами, представленный на странице **ADO** (ActiveX Data Object) Палитры компонентов.

Рис. 4.4. Страница **ADO** — компоненты для работы с базами данных

Для реализации больших возможностей, предоставляемых приложениями-серверами, в дополнение к компонентам, описанным выше, Delphi предоставляет возможность подключения внешних компонентов ActiveX. На рис. 4.5 приведено диалоговое окно подключения внешних компонентов ActiveX.

Их особенностью является то, что они напоминают обычные компоненты Delphi, но могут быть написаны на любом языке программирования, лишь бы это средство разработки поддерживало возможность их использования в разрабатываемых приложениях. Технология ActiveX базируется на технологии Microsoft COM.

Обычно компоненты ActiveX располагаются на одноименной странице Палитры компонентов. Если вы ее откроете, то увидите несколько компонентов, в том числе F1Book (рис. 4.6).

При определенных навыках и опыте разработчика программного обеспечения универсальным инструментом для создания отчетов в формате Word и Excel может стать библиотека ComObj.pas Delphi. Достаточно и одной функции из этой библиотеки — разумеется, это функция function CreateOleObject(const ClassName: string): Idispatch; (она возвращает

ссылку на интерфейс объекта, предназначенный для управления этим объектом). Эта функция всегда используется для позднего связывания — когда в момент написания и компиляции исходного текста программы неизвестно, будет работать приложение или нет. Ошибки синтаксиса при позднем связывании проявляются только в момент выполнения программы. Книга посвящена, в основном, таким методам создания приложений.

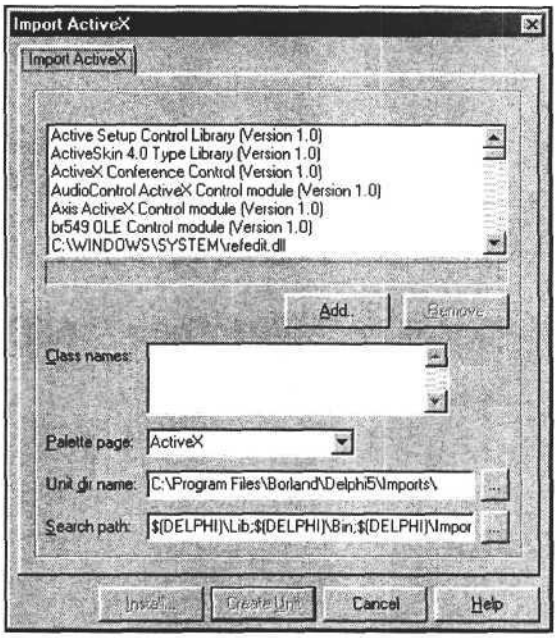


Рис. 4.5. Диалоговое окно импорта внешнего компонента ActiveX в систему

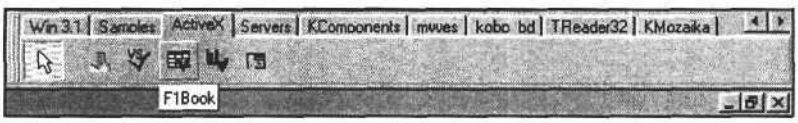
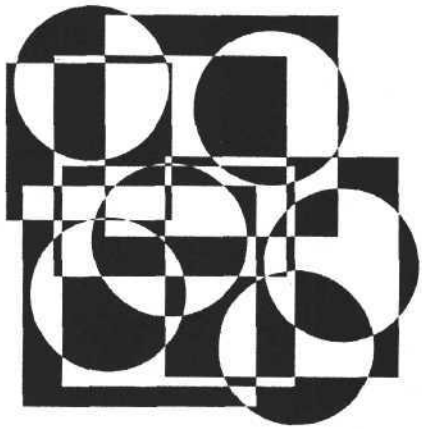


Рис. 4.6. Компонент F1Book на странице **ActiveX** Палитры компонентов

В следующей главе рассмотрим использование позднего связывания для доступа к приложению и документам Word.



ЧАСТЬ II

Разработка документов и приложений MS Word в Delphi

ГЛАВА 5

Работа с объектом Word.Application

В данной главе мы приступим к практическому созданию контроллера автоматизации для текстового редактора MS Word. Как уже сказано, для создания контроллера автоматизации необходимо получить доступ к объекту, стоящему на вершине объектной модели приложения Word, — к объекту Application (для доступа к нему используется идентификатор Word.Application).

Создание объекта Word.Application, запуск и визуализация окна приложения

Функция, реализующая механизм доступа к OLE-объекту, находится в библиотеке ComObj.pas — функция CreateOleObject. Ее единственным аргументом является строка-идентификатор, а возвращает она ссылку на объект.

Рассмотрим пример использования этой функции.

Создадим новый проект Delphi, в котором есть одна форма. В модуле формы укажем ссылку на использование библиотеки ComObj и объявим переменную W:variant. В форме расположим кнопку. В процедуре обработки нажатия кнопки напишем следующий программный текст.

Создание объекта Word.Application

```
uses ComObj;
  var W:variant;
procedure TForm1.Button1Click(Sender: TObject);
  begin
  W:=CreateOleObject('Word.Application');
end;
```

Если выполнить этот фрагмент программы, то приложение Word запустится, но его окно не отобразится на экране монитора. В память компьютера будет загружен объект Application, обеспечивающий доступ ко всем внутренним объектам, коллекциям и свойствам. В данный момент нас интересует только одно свойство этого объекта — Visible. Если его значение установить в True, то окно приложения Word станет видимым. Расположим в нашей форме компонент CheckBox1, в процедуре отклика которого напишем следующий программный текст:

Задание видимости окна приложения Word

```
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  W.Visible:=CheckBox1.Checked;
end;
```

Манипулируя состоянием объекта CheckBox1, мы обнаружим, что окно приложения то появляется, то пропадает с экрана монитора. Приложение загружено в память, без каких-либо открытых документов (рис. 5.1).

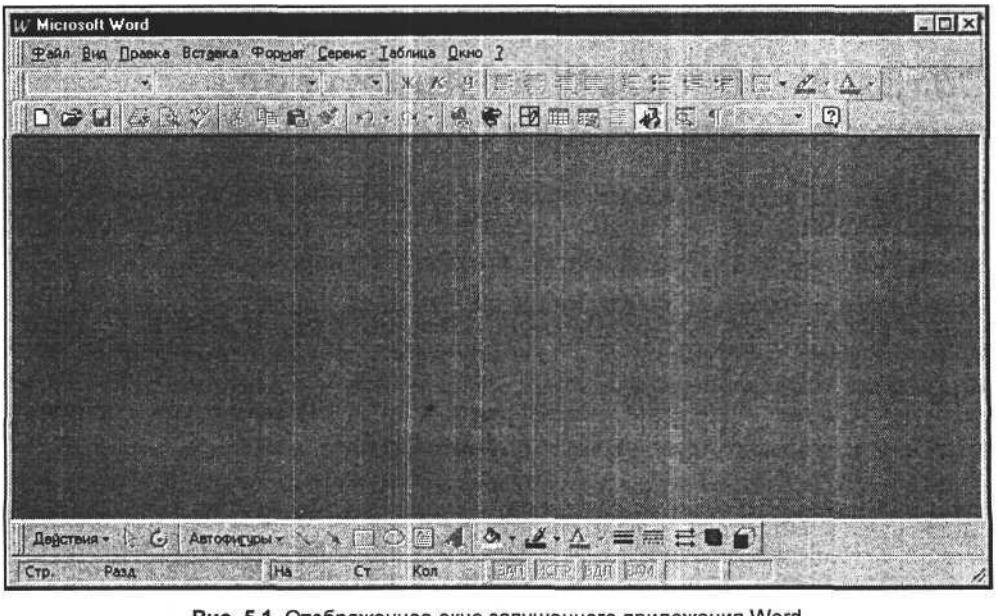


Рис. 5.1. Отображенное окно запущенного приложения Word

При формировании документов свойство Visible лучше установить в значение False, и устанавливать его в значение True только для отображения полностью созданных документов свойство Visible. Так можно сократить

время создания отчетов и повысить производительность работы приложений. Отлаживая приложение или изучая свойства объекта Application, лучше видеть все действия на мониторе. Поэтому установим Visible := True и приступим к работе с документами Word.

Создание документа

Итак, объект Application загружен в память компьютера, и у нас есть доступ к нему через переменную W:variant. Исследуем свойства этого объекта. Обратим внимание на коллекцию Documents. Она содержит документы, их свойства и методы для работы с ними. Элементами коллекции являются открытые в настоящий момент документы. Доступ к ним осуществляется через объекты Item(doc:variant), где doc — имя или индекс документа в коллекции. Поле Count коллекции содержит количество элементов коллекции, если Count=0, то нет ни одного открытого документа. Создадим новый документ. Для этого используем метод ADD этой же коллекции. Разместим в форме кнопку, в процедуру отклика которой на нажатие запишем следующий текст:

Создание документа в коллекции документов

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  W.Documents.Add;
end;
```

После выполнения метода ADD будет создан документ, который отобразится в окне приложения (рис. 5.2).

Обращаться к методу ADD коллекции Documents можно как без аргументов, так и с аргументом. Когда аргумента нет, создается обычный документ. Если метод вызывается с аргументом (строкой-указателем на файл шаблона), то создается документ по шаблону. В комплекте поставки MS Office есть несколько шаблонов, которые вы, вероятно, используете в своей работе. Шаблоны находятся в папке ...*\Microsoft Office\Шаблоны*. Для своих приложений можно создать дополнительные шаблоны документов, например шаблон формы платежного поручения или налоговой декларации. В дальнейшем, используя метод ADD с указанием на шаблон, легко создать нужный документ, заполняемый информацией из программы. Использование шаблонов при формировании новых документов позволит создавать гибкие и удобные для пользователя приложения в среде Delphi. Создадим новый документ на основе шаблона. Для этого разместим в форме кнопку и напишем следующий фрагмент программы.

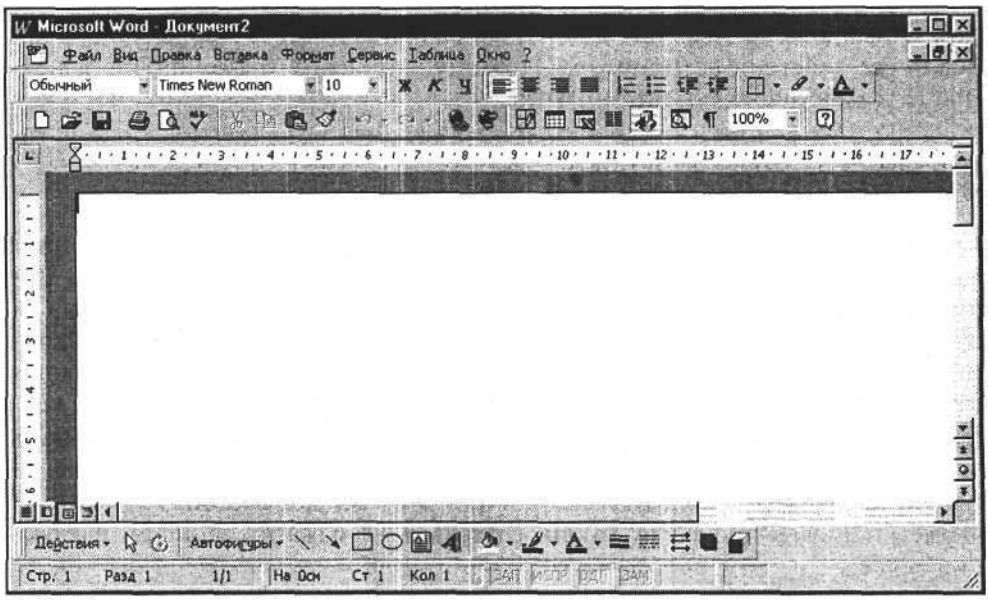


Рис. 5.2. Документ, созданный методом ADD без аргументов

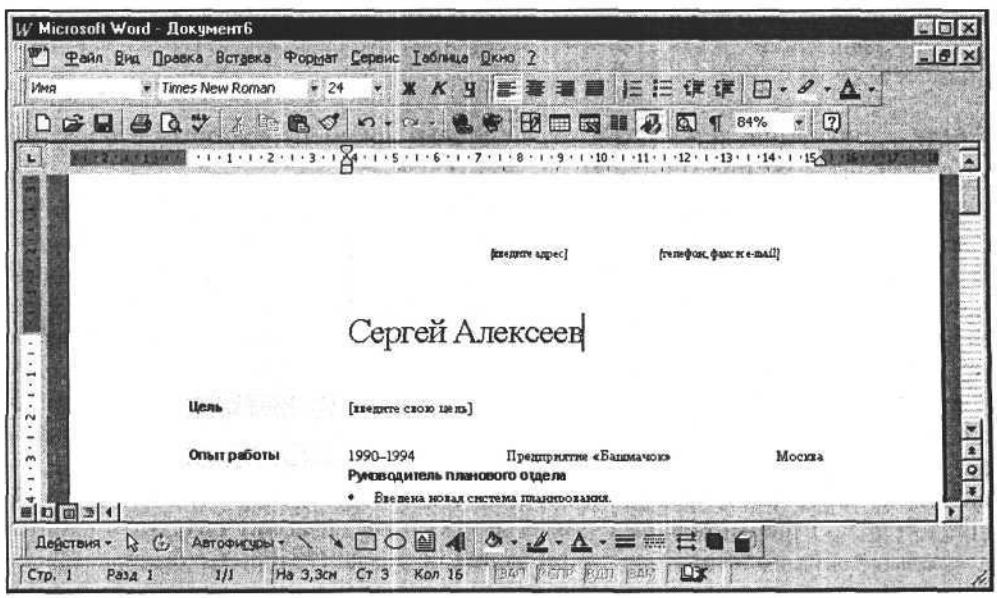


Рис. 5.3. Документ, созданный методом ADD с использованием шаблона

Создание документа по шаблону

```
procedure TForm1.Button3Click(Sender: TObject);
 var dir_:string;
begin
  GetDir(0,dir_);
  if not OpenDialog1.Execute then begin chdir(dir_); exit; end;
  chdir(dir_);
  W.Documents.Add(OpenDialog1.FileName);
end;
```

Данный фрагмент программы позволяет открыть диалоговое окно, найти и выбрать файл шаблона, используемый при создании нового документа. Выберем файл *Стандартное резюме.DOT* из папки *...\Microsoft Office\Шаблоны \Другие документы*. Результат выполнения метода ADD представлен на рис. 5.3.

Если требуется внести изменения в созданный ранее документ, тогда открываем его, используя методы коллекции Documents.

Открытие документа

Ранее созданный документ можно открыть с помощью метода Open. При вызове метода можно указать и несколько аргументов, но главный из них — ссылка на путь и имя файла. Откроем ранее созданный документ. Фрагмент текста программы выглядит так:

Открытие документа

```
procedure TForm1.Button4Click(Sender: TObject);
 var dir_:string;
begin
  GetDir(0,dir_);
  if not OpenDialog2.Execute then begin chdir(dir_); exit; end;
  chdir(dir_);
  W.Documents.Open(OpenDialog2.FileName);
end;
```

Результат выполнения данного фрагмента программы представлен на рис. 5.4.

Метод Open коллекции Documents можно вызывать с несколькими аргументами. Обратимся к справочной системе Visual Basic и рассмотрим синтаксис и аргументы этого метода (табл. 5.1).

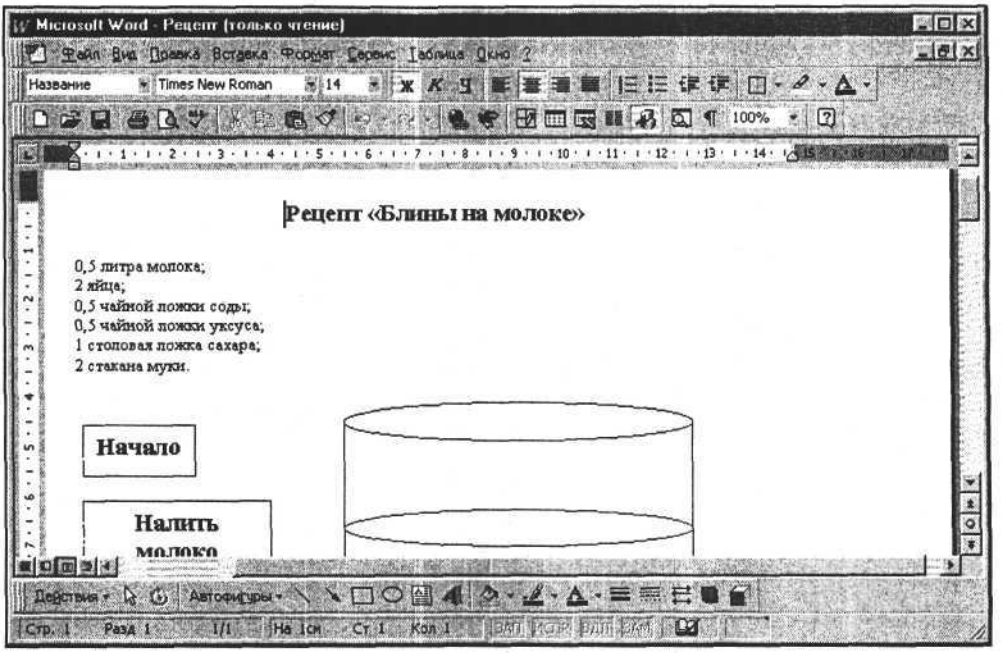


Рис. 5.4. Документ, открытый методом Open

Синтаксис метода Open

```
Documents.Open(FileName, ConfirmConversions, ReadOnly, AddToRecentFiles,
  PasswordDocument, PasswordTemplate, Revert, WritePasswordDocument,
  WritePasswordTemplate, Format)
```

Таблица 5.1. *Аргументы метода Open, их типы и функциональное значение*

Аргумент	Тип	Значение
FileName	String	Путь и имя файла
ConfirmConversions	Boolean	False — не открывать диалоговое окно **Преобразование файла** при открытии файла, формат которого отличается от DOC
ReadOnly	Boolean	True — открыть документ в режиме "только для чтения"
AddToRecentFiles	Boolean	True — добавляет имя файла в список файлов меню **File**
PasswordDocument	String	Пароль для открытия документа

Таблица 5.1 (окончание)

Аргумент	Тип	Значение
PasswordTemplate	String	Пароль для открытия шаблона
Revert	Boolean	True — возврат к сохраненному документу, если этот документ открывается повторно
WritePasswordDocument	String	Пароль для сохранения измененного документа в файле
WritePasswordTemplate	String	Пароль для сохранения изменений в шаблоне
Format	Число[1]	Формат открываемого документа

При вызове метода Open можно игнорировать некоторые аргументы и не указывать их, например, вызов W.Documents.Open(FileName); просто открывает файл без каких-либо дополнительных возможностей. Если немного изменить синтаксис вызова — W.Documents.Open(FileName:= 'c:\Документ1.doc');, то конечный результат будет таким же, как и в первом случае. Когда нам потребуется открыть документ в режиме "только для чтения", то используем следующий синтаксис:

```
W.Documents.Open(FileName:='c:\Документ1.doc', ReadOnly:=True);
```

Разместим в нашей форме кнопку и напишем программный текст, позволяющий открывать документы в режиме "только для чтения".

Открытие документа в режиме "только для чтения"

```
procedure TForm1.Button5Click(Sender: TObject);
 var dir_:string;
begin
  GetDir(0,dir_);
  if not OpenDialog2.Execute then begin chdir(dir_); exit; end;
  chdir(dir_);
  W.Documents.Open(OpenDialog2.FileName,ReadOnly:=true);
end;
```

Если для открытия документа используется не два, а три аргумента, независимо от их последовательности, то синтаксис изменится на одну запись.

[1] Некоторым свойствам (имеющим разные числовые типы) можно присваивать значения типа Integer или Extended, это не вызовет ошибки.

Например: откроем документ в режиме "только для чтения", который защищен паролем.

```
W.Documents.Open(FileName:='c:\Документ1.doc',
  ReadOnly:=True, PasswordDocument:='123');
```

Здесь строка '123' — значение пароля.

Обратим внимание на последний аргумент метода ADD — Format. Этот аргумент может принимать целые числовые значения и определяет формат открываемого документа (табл. 5.2).

Таблица 5.2. *Значения аргумента Format и форматы открываемых документов*

Константа	Значение	Формат открываемого документа
WdOpenFormatAuto	0	Выбирается автоматически
WdOpenFormatDocument	1	Документ Word (файл с расширением имени DOC)
WdOpenFormatRTF	3	Документ в формате RTF (файл с расширением имени RTF)
WdOpenFormatTemplate	2	Шаблон Word (файл с расширением имени DOT)
WdOpenFormatText	4	Текст (файл с расширением имени TXT)
WdOpenFormatUnicodeText	5	Кодированный текст (файл с расширением имени TXT) — текст в формате UNICODE

Откроем текстовый файл, используя возможность задания формата открываемого документа. Для этого добавим в форму кнопку и напишем следующий фрагмент программы:

Открытие документа в формате TXT

```
procedure TForm1.Button7Click(Sender: TObject);
const WdOpenFormatText=4;
 var dir_:string;
       a_:integer;
     eee_:string;
begin
  GetDir(0,dir_);
  if not OpenDialog3.Execute then begin chdir(dir_); exit; end;
```

```
  chdir(dir_);
  W.Documents.Open(OpenDialog3.FileName,Format:=wdOpenFormatText);
end;
```

Результат выполнения этой процедуры представлен на рис. 5.5.

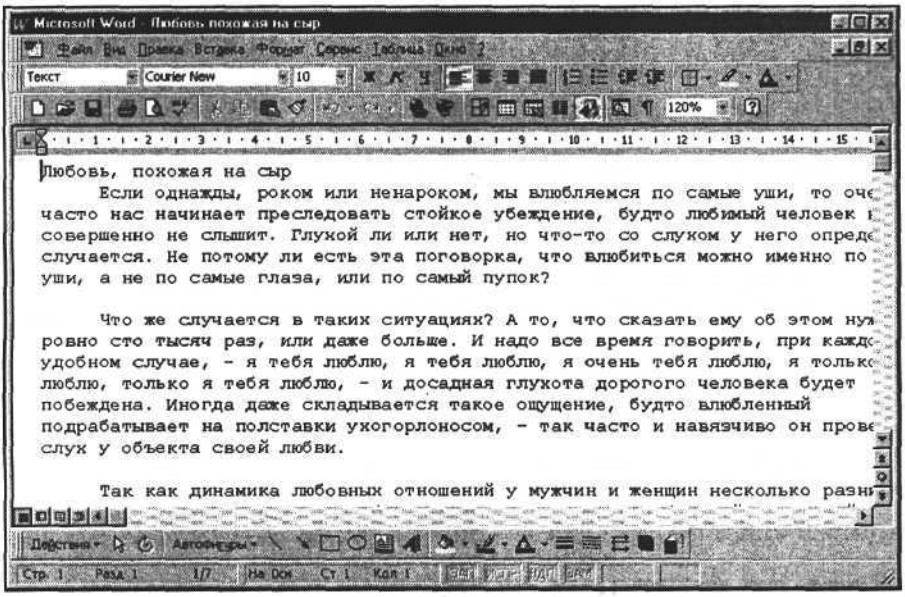


Рис. 5.5. Документ в формате TXT, открытый методом Open

Но если мы попробуем открыть документ в формате DOC, указав значение Format:=wdOpenFormatText, то получим сообщение об ошибке (рис. 5.6, *а*).

В режиме выполнения приложения сообщение об этой ошибке будет выглядеть, как показано на рис. 5.6, *б*.

Подобные ошибки, возникающие во время работы с объектом Application, можно обрабатывать в приложении. Для этого используется синтаксис Delphi для обработки исключительных ситуаций, например:

```
try
// программный код, который может вызвать исключительную ситуацию
except
// программный код для обработки исключительной ситуации
end;
```

О том, как использовать обработку исключительных ситуаций на практике, рассмотрим в конце главы, когда создадим несколько функций нашей будущей библиотеки для работы с приложением Word. Сейчас продолжим ис-

следование. В запущенном приложении Word объект Application содержит несколько созданных и открытых документов, поэтому далее рассмотрим работу со списком открытых документов.

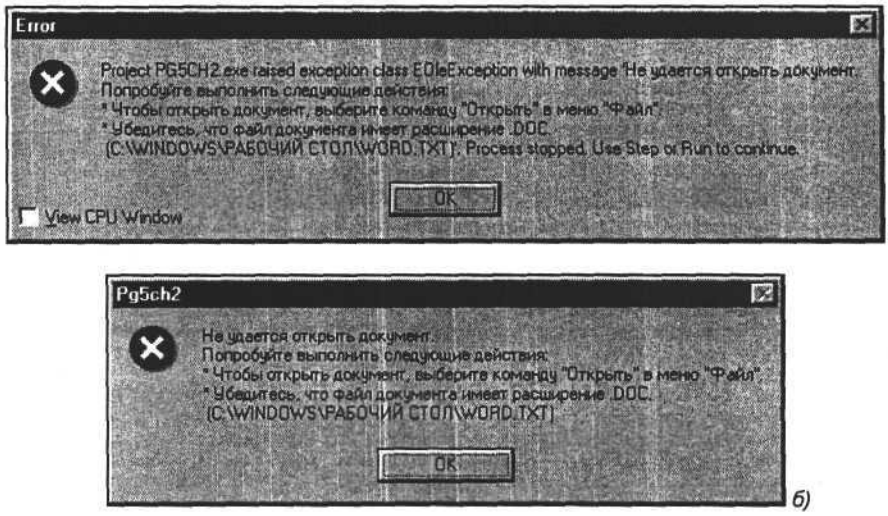


Рис. 5.6. Сообщение об ошибке при попытке открыть документ в формате DOC как текстовый файл: режим отладки приложения (*а*), режим выполнения приложения (*б*)

Работа со списком открытых документов

Вся информация об открытых и созданных документах содержится в коллекции Documents, элементами коллекции являются документы, а свойство коллекции Count определяет количество документов. Для начала получим список открытых документов, используем свойство Count коллекции и переберем все элементы коллекции. Создадим новую форму и разместим в ней компонент типа ListBox1:TListBox. В момент появления этой формы на экране заполним ListBox1 именами документов, используя следующий программный текст.

Получение списка открытых документов

```
procedure TUG5CH21_.FormShow(Sender: TObject);
 var a_:integer;
    eee_:string;
begin
  ListBox1.Items.Clear;
  for a_:=1 to W.Documents.count do begin
```

```
    eee_:=W.Documents.Item(a_).Name;
    ListBox1.Items.Add(eee_);
  end;
end;
```

В результате выполнения этой процедуры получим список документов, открытых в данный момент времени. Пример формы с таким списком показан на рис. 5.7.

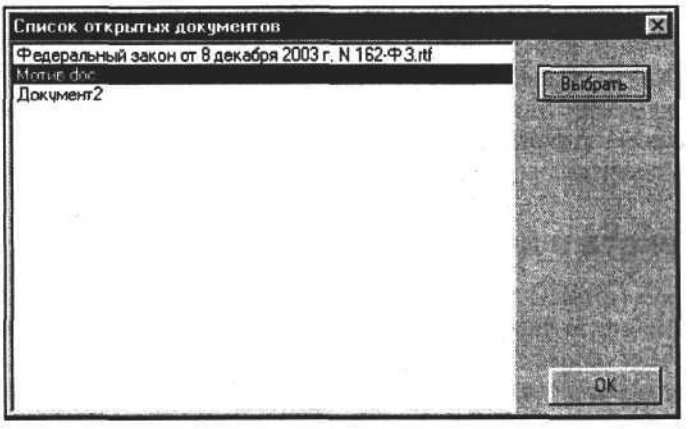


Рис. 5.7. Список открытых документов приложения Word

Когда документы открыты, мы можем выполнять с ними любые действия, например редактировать и сохранять, но перед этим нужно выбрать в списке документ, с которым будем работать. Для этого можно использовать метод Select или Activate элемента коллекции Documents. Первый метод выбирает и выделяет документ, второй — только выбирает. Вызвать метод Activate можно двумя следующими способами, но в обоих случаях результат будет одним и тем же.

- `W.Documents.Item(<номер документа в списке>).Activate;`
- `W.Documents.Item(<имя документа в списке>).Activate;`

В нашем примере на рис. 5.7 по нажатию кнопки выполняется процедура, которая выбирает документ в списке ListBox1 по номеру документа. Номер первого документа в коллекции — 1.

Выбор документа в списке по номеру

```
procedure TUG5CH21_.Button1Click(Sender: TObject);
begin
  W.Documents.Item(ListBox1.ItemIndex+1).Activate;
end;
```

После того как документ создан (или открыт) и выбран, можно приступить к внесению в него изменений, т. е. к редактированию. Самое простое — записать текст в документ.

Запись и чтение текста документа

Для того чтобы работать с документом, он не обязательно должен быть активным, но для удобства в основном будем рассматривать примеры работы с активным документом.

Рассмотрим объект Range, который входит в объект-документ и является его свойством. Объект Range представляет собой содержание части документа или всего документа. Методы этого объекта позволяют записывать и считывать информацию документа. Воспользуемся этим объектом и его методами для работы с содержимым документа.

Запись текста в документ

Для начала рассмотрим два метода, которые вставляют (записывают) текст. Метод InsertBefore записывает текст в начало содержимого объекта Range. Метод InsertAfter записывает текст в конец содержимого объекта Range. Не нужно забывать, что объект Range может содержать как весь документ, так и его часть, а его методы действуют только на содержимое конкретного объекта Range. Допустим, объект Range включает в себя весь текст документа. Тогда метод InsertBefore вставит текст в начале документа, сместив текст, который уже был в документе. Метод InsertAfter при этом вставит текст в конце документа. Если объект Range включает только часть текста документа, то эти методы вставят новый текст в начало или в конец этой части, сместив текст, который был в документе до их выполнения.

Рассмотрим практическую реализацию вызова этих методов в приложениях Delphi. В форме разместим кнопки и напишем следующие процедуры:

Запись текста в документ с помощью методов InsertBefore и InsertAfter

```
procedure TUG5CH22_.Button1Click(Sender: TObject);
begin
  W.ActiveDocument.Range.InsertAfter(Memo1.Text+'('+inttostr(nn)+')  ');
  nn:=nn+1;
end;

procedure TUG5CH22_.Button2Click(Sender: TObject);
begin
  W.ActiveDocument.Range.InsertBefore(Memo1.Text+'('+inttostr(nn)+')  ');
  nn:=nn+1;
end;
```

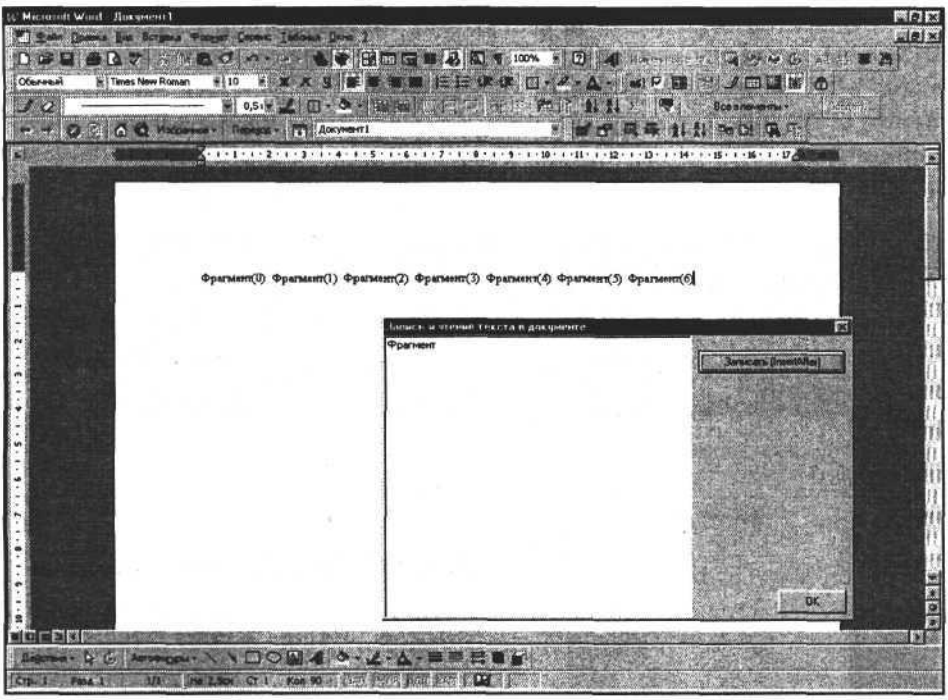


Рис. 5.8. Запись текста в документ (метод InsertAfter)

Пример результата работы метода InsertAfter представлен на рис. 5.8.

На рис. 5.8 видно, что следующий фрагмент записывается правее предыдущего, т. е. новый текст добавлен в конец документа.

Пример результата работы метода InsertBefore представлен на рис. 5.9.

В данном случае новый фрагмент текста записывается в начало, смещая к концу документа текст, уже имевшийся в документе.

Как выделить часть текста в документе и работать с ней, применяя описанные методы не ко всему документу, а к фрагменту? Для этой цели можно использовать функцию Range, возвращающую объект типа Range; аргументы функции — позиции начала и конца фрагмента.

В качестве примера рассмотрим небольшой фрагмент программы Delphi.

Запись текста в заданный фрагмент документа

```
var MyRange:variant;
begin
  MyRange:= W.ActiveDocument.Range(100,200);
  MyRange.InsertAfter('Вставляем текст в конце заданного фрагмента')
end;
```

Первый оператор определяет новый объект MyRange типа Range, включающий в себя данные с позиции 100 до позиции 200 (позиции соответствуют символам). После этого с новым объектом можно работать так же, как с базовым объектом, т. е. использовать его для записи текста в определенную позицию документа. В данном случае объект MyRange можно использовать для записи и чтения документа с позиции 100 по 200.

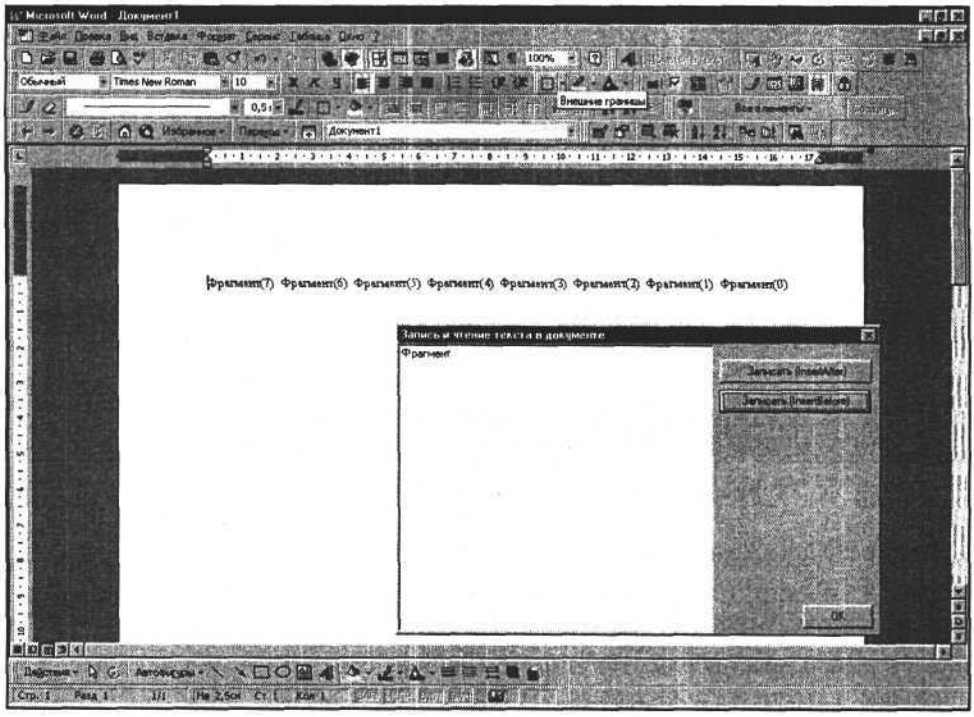


Рис. 5.9. Запись текста в документ (метод InsertBefore)

Чтение текста из документа

Рассмотрим чтение текста из открытого документа Word. Создадим новую кнопку и в процедуре обработки ее нажатия напишем следующий код.

Чтение текста из документа, открытого в приложении Word

```
procedure TUG5CH22_.Button3Click(Sender: TObject);
begin
  Memo1.Text:=W.ActiveDocument.Range.Text;
end;
```

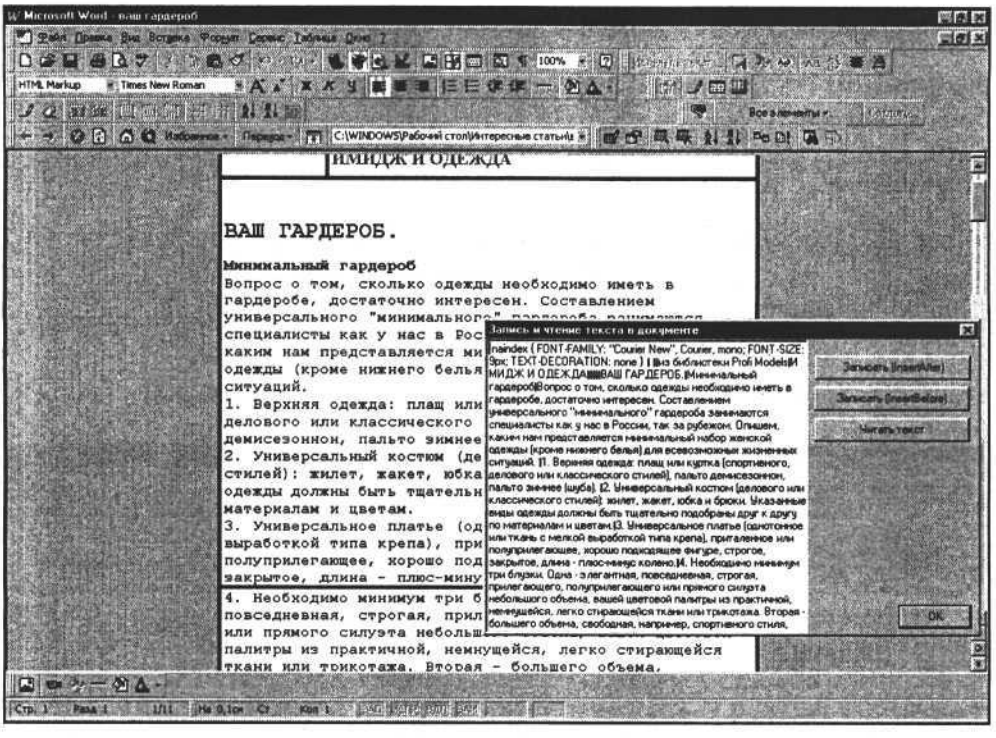


Рис. 5.10. Чтение текста из документа

На рис. 5.10 представлен результат работы этого кода.

Если требуется экспортировать в программу-контроллер очень большой текст, то можно работать с его фрагментами в документе. Для этого используйте процедуру Range(first, last), где first, last — границы диапазона (позиции символов), в котором находится рабочий фрагмент.

Другой способ чтения текста документа основан на чтении слов.

Примечание

За *слово* принимается непрерывный фрагмент текста, состоящий из букв и цифр, который оканчивается пробелом, знаком препинания, арифметическим, логическим или другим символом. Все перечисленные символы или их комбинации за исключением пробела также представляют собой отдельные слова.

Рассмотрим коллекцию Words слов документа. Синтаксис может быть таким:

```
W.ActiveDocument.Words
```

Эта коллекция включает в себя слова объекта-хозяина. В следующем примере коллекция Words содержит все слова активного документа.

Чтение коллекции слов в активном документе

```
procedure TUG5CH22_.Button4Click(Sender: TObject);
 var a_:integer;
begin
  Memo1.Lines.Clear;
  for a_:=1 to W.ActiveDocument.Words.count do begin
    Memo1.Lines.Add(W.ActiveDocument.Words.item(a_).Text);
  end;
end;
```

Результат выполнения данной процедуры представлен на рис. 5.11.

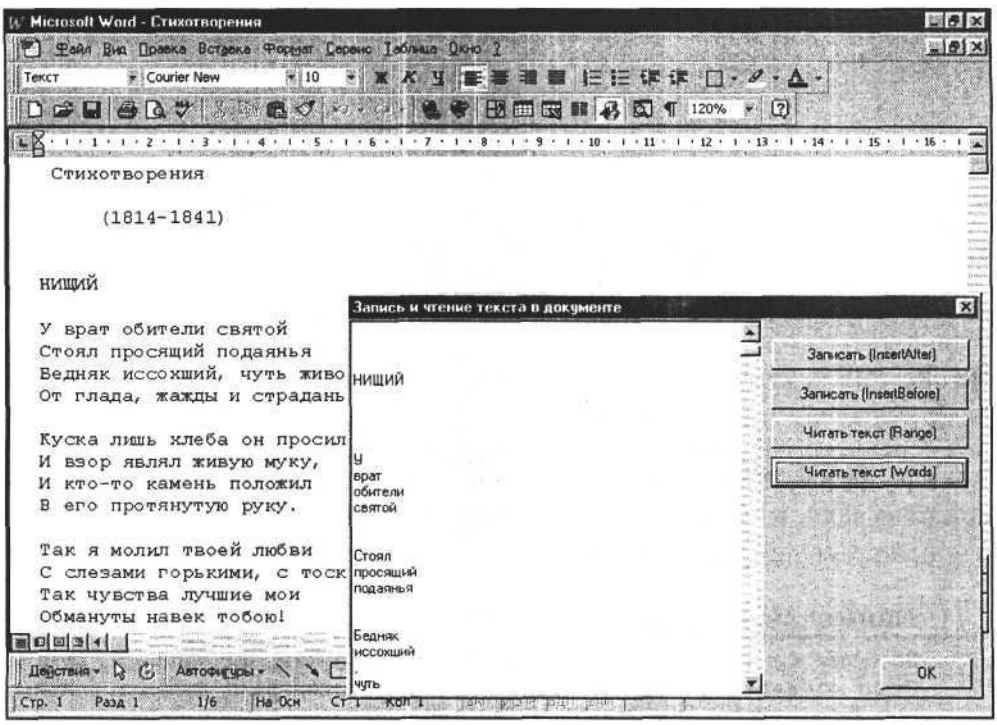


Рис. 5.11. Чтение коллекции слов в активном документе

Очевидно, что данный способ подходит для работы с отдельными словами, но его можно использовать только для чтения, а не для записи. Когда документ открыт, прочитан, изменен, то, возможно, потребуется сохранить эти изменения.

Сохранение документа

Самый простой способ сохранения документа заключается в вызове метода Save объекта-документа, например: W.ActiveDocument.Save. Данный способ удобно использовать, если документ ассоциирован с файлом на диске. Когда мы попробуем сохранить с помощью этого метода вновь созданный документ, то откроется диалоговое окно (рис. 5.12), активированное приложением Word.

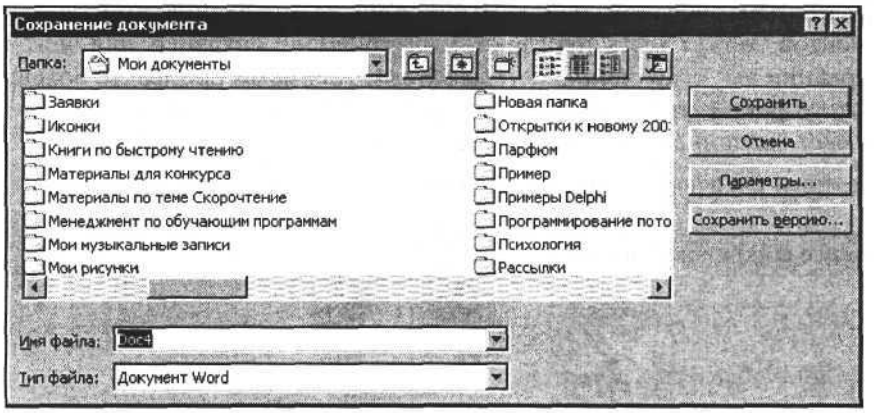


Рис. 5.12. Попытка сохранить новый документ с помощью метода Save

Если мы закроем диалог без сохранения документа, то получим ошибку. На рис. 5.13 отображено диалоговое окно с сообщением об исключительной ситуации, возникшей в режиме отладки программы.

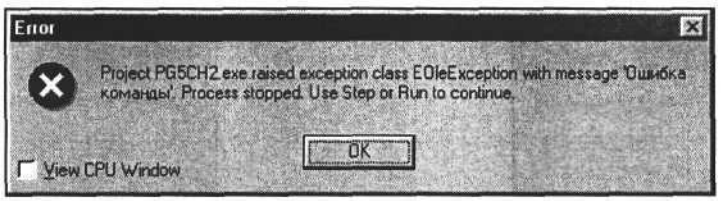


Рис. 5.13. Отображение ошибки выполнения метода Save (режим отладки приложения)

В режиме выполнения откомпилированного приложения сообщение об этой ошибке будет выглядеть, как показано на рис. 5.14.

Чтобы избежать появления ошибок выполнения, необходимо при выполнении процедур сохранения документа обрабатывать исключительные ситуации. Используйте конструкцию try ... except ... end языка Delphi. В случаях, когда не нужно выводить диалоговое окно сохранения документа, его можно подавить. Метод подавления диалоговых окон Word будет рассмотрен позже.

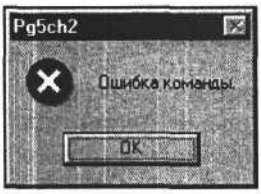


Рис. 5.14. Отображение ошибки выполнения метода Save (режим выполнения приложения)

Эффективным способом избежать ошибок, возникающих при сохранении документов, может быть проверка "был документ сохранен или нет" и использование для сохранения метода SaveAs. Для проверки факта сохранения открытого документа на диск используйте свойство документа Saved — если оно имеет значение True, то документ был сохранен, если False — нет.

Рассмотрим следующий фрагмент программы.

Проверка факта сохранения документа

```
procedure TUG5CH22_.Button5Click(Sender: TObject);
begin
  if W.ActiveDocument.Saved then
           messagebox(handle,'Документ сохранен!','Внимание!',0);
  if not W.ActiveDocument.Saved then
        messagebox(handle,'Документ не сохранен!','Внимание!',0);
end;
```

Результат выполнения может быть одним из двух (рис. 5.15 и 5.16).

Рис. 5.15. Документ был сохранен

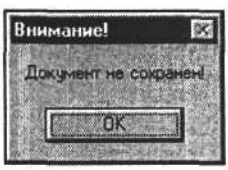


Рис. 5.16. Документ не был сохранен

Если вновь созданный документ не был сохранен, то для его записи на диск используется метод SaveAs. В спецификации языка Visual Basic (VB) он описан так:

```
ActiveDocument.SaveAs(FileName, FileFormat, LockComments, Password,
  AddToRecentFiles, WritePassword, ReadOnlyRecommended,
  EmbedTrueTypeFonts, SaveNativePictureFormat, SaveFormsData,
  SaveAsAOCELetter)
```

Аргументы метода SaveAs определяют режимы и формат файла для сохраняемого документа. Используя их при вызове метода, можно добиться такого же результата, как и в режиме обычного пользователя приложения Word. Аргументы метода, их типы и назначение представлены в табл. 5.3.

Таблица 5.3. *Аргументы метода SaveAs, их типы и функциональное назначение*

Аргумент	Тип	Функциональное назначение
FileName	String	Путь и имя файла
FileFormat	Число	Формат файла, см. табл. 5.4
LockComments	Boolean	True — не сохранять комментарии
Password	String	Пароль, который будет использоваться при открытии документа
AddToRecentFiles	Boolean	True — добавить имя файла в список меню **File**
WritePassword	String	Пароль, который будет использоваться для сохранения документа
ReadOnlyRecommended	Boolean	True — в последующем документ можно открыть "только для чтения"
EmbedTrueTypeFonts	Boolean	True — при сохранении перевести шрифты документа в TrueType
SaveNativePictureFormat	Boolean	Используется для импорта графики из форматов, не поддерживаемых Windows. True — импортировать только графику, поддерживаемую Windows
SaveFormsData	Boolean	True — сохранить форму документа без текста
SaveAsAOCELetter	Boolean	Используется в версиях Word для компьютеров Apple Macintosh

При вызове метода SaveAs можно задавать как один, так и несколько аргументов — количество определяется только необходимостью их использования в создаваемом приложении. Обычно достаточно первого аргумента (путь и имя файла).

При вызове метода SaveAs из приложений, созданных в Delphi, есть некоторые отличия синтаксиса от вызова в VB, но они не принципиальны.

Рассмотрим примеры синтаксиса вызовов метода SaveAs в Delphi.

Сохранение документа с указанием пути и имени файла:

- `W.ActiveDocument.SaveAs(MyFileName);`
- `W.ActiveDocument.SaveAs(FileName:=MyFileName);`

где MyFileName — строка, содержащая путь и имя файла.

Сохранение документа с указанием пути и имени файла, а также пароля:

-
```
W.ActiveDocument.SaveAs(FileName:=MyFileName,
                        Password:='мой пароль')
```

Сохранение документа, ассоциированного с именем файла, с указанием пароля:

- `W.ActiveDocument.SaveAs(Password:='мой пароль')`

Для задания или изменения формата сохраняемого документа необходимо использовать числовой аргумент FileFormat, значения которого представлены в табл. 5.4.

Таблица 5.4. *Значения аргумента FileFormat метода SaveAs и форматы файлов сохраняемых документов*

Константа	Значение	Формат сохраняемого документа
WdFormatDocument	0	Документ Word (файл с расширением имени DOC)
WdFormatDOSText	4	Текст DOS (файл с расширением имени TXT)
WdFormatDOSTextLineBreaks	5	Текст DOS с разбиением на строки (файл с расширением имени TXT)
WdFormatRTF	6	Текст в формате RTF (файл с расширением имени RTF)
WdFormatTemplate	1	Шаблон Word (файл с расширением имени DOT)
WdFormatText	2	Только текст (файл с расширением имени TXT)
WdFormatTextLineBreaks	3	Текст с разбиением на строки (файл с расширением имени TXT)
WdFormatUnicodeText	7	Кодированный текст (файл с расширением имени TXT) — текст в кодировке UNICODE

Итак, мы открыли документ, считали из него текст, внесли в документ необходимые изменения и сохранили его. Теперь нужно закрыть его и закончить работу с приложением Word.

Закрытие документа и приложения Word

Если приложение обрабатывало несколько документов, и все они уже сохранены, то эти документы можно закрыть одновременно с помощью метода Close коллекции Documents. Добавим в форму приложения кнопку и напишем программный текст, который закроет все открытые документы.

Закрытие всех открытых документов

```
procedure TForm1.Button15Click(Sender: TObject);
begin
  W.Application.Documents.Close;
end;
```

Если требуется выборочно закрывать документы коллекции, используйте метод Close объекта-документа. Например, чтобы закрыть активный документ, используйте оператор:

```
W.ActiveDocument.Close;
```

Особенность метода Close — возможность задать режим сохранения документа во время его закрытия. Оператор

```
W.ActiveDocument.Close(True);
```

сохраняет и закрывает документ.

Документы закрыты, можно закрыть приложение Word и очистить память от объекта Application. Используем метод Quit объекта Application и оператор W:=UnAssigned;. Реализуем это в виде процедуры-обработчика нажатия кнопки.

Закрытие приложения Word

```
procedure TForm1.Button16Click(Sender: TObject);
begin
  W.Quit;
  W:=UnAssigned;
end;
```

Обработка ошибок выполнения при работе с объектом Application

Вызов методов объекта Application и его дочерних объектов может приводить к ошибкам в процессе работы программы. Эти ошибки нужно обрабатывать, иначе они приведут к "зависанию" приложения или всей системы.

К тому же обработка ошибок — средство получения информации для анализа функционирования приложения. Как обычно, для обработки исключительной ситуации используем синтаксис try ...except ... end. Таким образом, каждое обращение к методам и объектам Application придется обрамлять синтаксисом обработки ошибок. Это, в конечном итоге, приведет к увеличению исходного текста программы и сопутствующим проблемам. Решением этих проблем может быть создание и использование пользовательской библиотеки процедур и функций с обработкой ошибок выполнения. Для начала можно создать библиотеку в виде модуля unit, а затем — в виде динамической библиотеки. Рассмотрим синтаксис библиотеки и процедур и функций для работы с Word.Application.

Синтаксис библиотеки процедур и функций для работы с Word.Application

```
unit MyWord;
interface
// Заголовки процедур и функций
Function CreateWord:boolean;
Function VisibleWord(visible:boolean):boolean;

Implementation
// Используемые библиотеки
uses ComObj;
// W - переменная, которая является ссылкой на объект Application
 var W:variant;

// Функция CreateWord создает объект Word.Application и помещает ссылку
// на него в переменную W:variant. Если объект создан без ошибок, то
// исключительная ситуация не возникает и функция возвращает True. Если
// возникает ошибка, например если приложение Word не установлено
// в системе, то вызывается исключительная ситуация и функция возвращает
// значение False.
Function CreateWord:boolean;
begin
  CreateWord:=true;
  try
    W:=CreateOleObject('Word.Application');
    except
    CreateWord:=false;
  end;
End;

// Функция VisibleWord показывает/скрывает окно приложения Word.
// Она основана на свойстве Visible объекта Application.
```

```
// В результате ошибки возникает исключительная ситуация
// и функция возвращает значение False.
Function VisibleWord(visible:boolean):boolean;
begin
  VisibleWord:=true;
  try
    W.visible:= visible;
    except
    VisibleWord:=false;
  end;
End;
End.
```

Полный текст библиотеки процедур и функций с описанием приведен в *Приложении 1.*

Далее в книге все исходные тексты примеров приложений приводятся с использованием этой библиотеки.

ГЛАВА 6

Создание простого документа

В этой главе мы рассмотрим некоторые свойства и методы объектов, принадлежащих объекту Application, необходимые для создания простого документа, например почтового конверта и бланка платежного поручения. Вы можете, используя и развивая описанную здесь методику, автоматизировать создание различных документов для нужд вашего предприятия.

Выделение текста

Из предыдущей главы вы узнали, что для работы с текстом используется объект Range, принадлежащий объекту-документу. В прошлый раз мы рассмотрели методы объекта Range, предназначенные для вставки фрагментов текста в начало или конец или в определенную позицию документа. Так можно создавать простой текст. Но чтобы создавать сложные документы с определенной структурой, заполняемой при формировании документа, требуются методы, позволяющие выделять фрагменты и работать с ними. Также необходимы функции поиска фрагментов и подстановки текста, методы для перемещения курсора в начало, конец или определенную позицию документа. Все эти методы реализованы в объектах Range и Selection. Selection — объект, ассоциированный с выделенным в данный момент времени объектом. Выделенным объектом может быть не только фрагмент текста, но и ячейки таблиц, таблицы целиком, надписи, рисунки и т. д. Исходя из этого, объект Selection может обладать свойствами любого визуального объекта, входящего в документ, и даже свойствами самого документа (если он выделен полностью). Но сейчас нас интересует только текст, и мы будем рассматривать Selection только применительно к нему.

Для того чтобы получить доступ к выделенному объекту, необходимо выделить весь текст документа или его фрагмент. Выделим фрагмент текста. Для этого можно использовать метод Select объекта Range. Рассмотрим два способа использования этого метода.

Выделение фрагмента текста в документе (первый способ)

```
var MyRange:variant;
      start, end:integer;
...
MyRange:=W.ActiveDocument.Range(statr, end);
MyRange.Select;
```

Выделение фрагмента текста в документе (второй способ)

```
var statr, end:integer;
...
W.ActiveDocument.Range(statr, end).Select;
```

Результаты применения этих способов абсолютно идентичны и основаны на функции Range(a, b), возвращающей усеченный объект Range. Иногда удобно использовать первый способ, особенно если в дальнейшем предполагается работа с усеченным объектом.

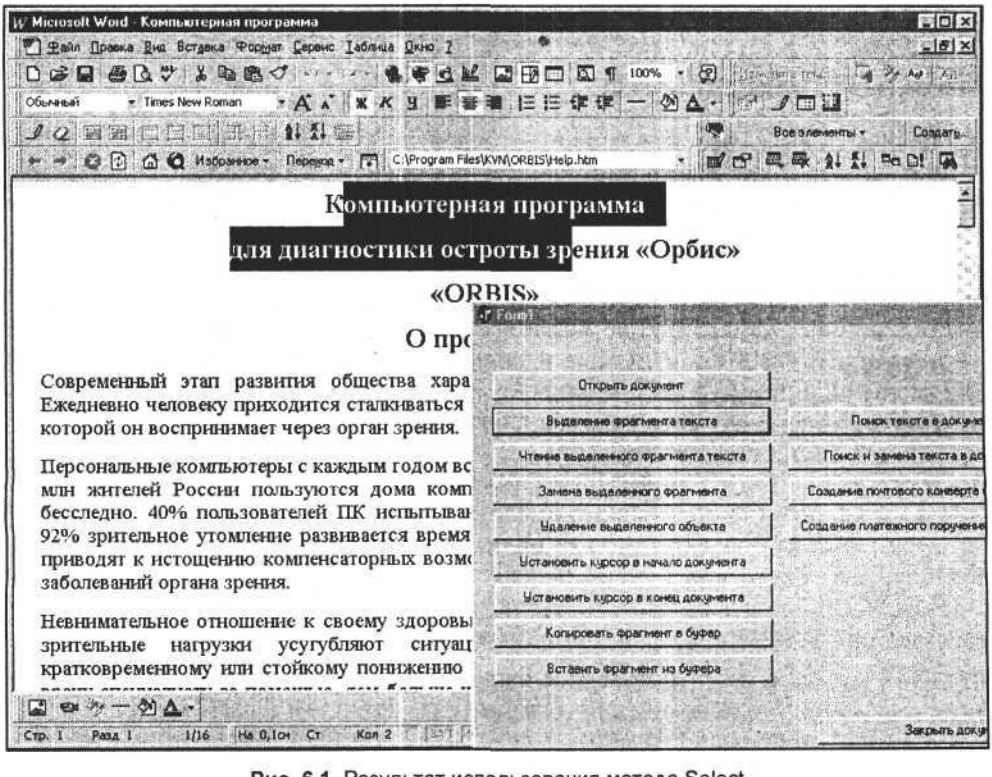


Рис. 6.1. Результат использования метода Select

В следующем примере используется второй способ. Создадим форму и кнопку, для которой в процедуре обработки нажатия разместим следующий программный текст:

Процедура выделения фрагмента текста

```
procedure TForm1.Button4Click(Sender: TObject);
begin
W.ActiveDocument.Range(1, 50).Select;
end;
```

Откроем любой документ и вызовем эту процедуру, нажав кнопку. Результат ее выполнения отображен на рис. 6.1.

Таким образом, мы получили объект Selection. Пусть он содержит только текст, но мы уже можем исследовать методы и свойства этого объекта.

Объект Selection

Исследуем объект Selection. Как уже сказано, этот объект может обладать свойствами любого выделяемого объекта. В нашем случае выделен текст, поэтому было бы интересно для начала прочитать этот текст. Весь выделенный текст содержится в свойстве Text объекта Selection.

Прочитаем этот текст, используя средства Delphi и ссылку W на объект Application. Разместим в нашей форме новую кнопку и напишем следующий фрагмент программы в процедуре обработки ее нажатия.

Чтение (импорт) выделенного текста из документа Word

```
procedure TForm1.Button5Click(Sender: TObject);
 var eee_:string;
begin
eee_:=W.Selection.Text;
messagebox(handle,pchar(eee_),'Чтение текста из выделенного фрагмента!',0);
end;
```

Оператор eee_:=W.Selection.Text; помещает выделенный текст в строковую переменную.

Результат выполнения этой процедуры представлен на рис. 6.2.

Для чтения текста из документа можно воспользоваться не только объектом Selection, но и свойствами и полями объекта Range. Например, оператор eee_:=W.ActiveDocument.Range(1, 23).Text; поместит в переменную eee_ текст с первого по двадцать третий символ документа. Это справедливо, по-

тому что объект Selection обладает свойствами и методами любых видимых элементов документа, в том числе и свойствами фрагмента текста. Конечно, можно напрямую работать с объектами документа, не выделяя их, но в некоторых случаях просто не обойтись без выделения, а также без свойств и методов объекта Selection.

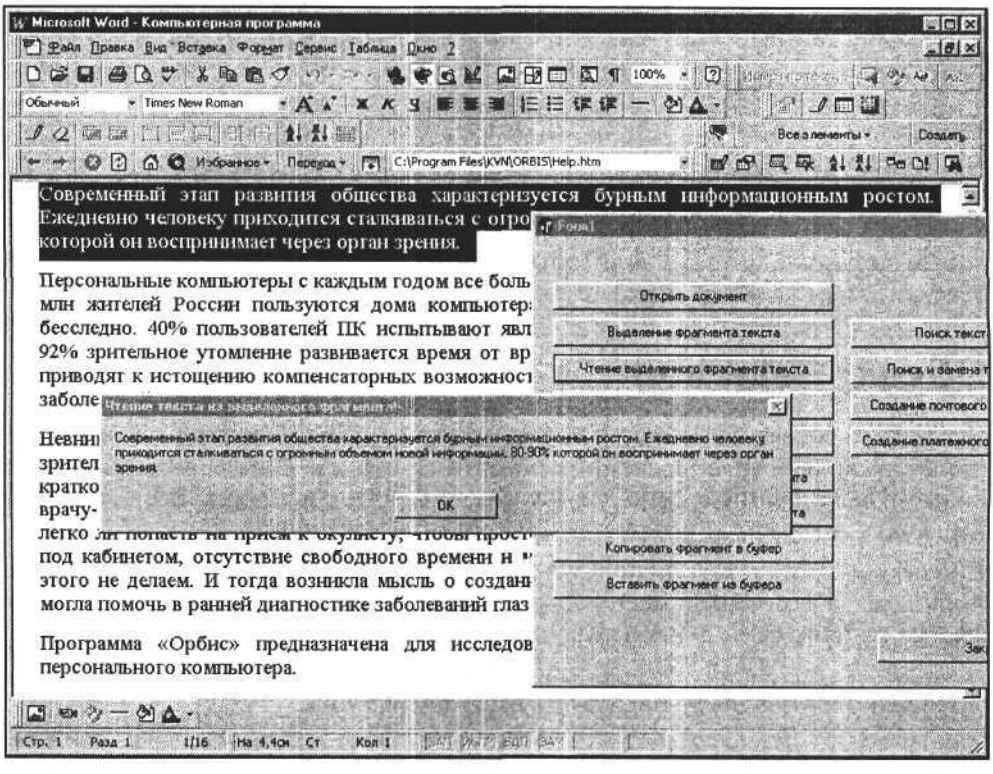


Рис. 6.2. Чтение выделенного фрагмента текста

Рассмотрим некоторые свойства и методы этого объекта, в том числе принадлежащие ему объекты. В табл. 6.1 перечислены наиболее часто используемые свойства и методы объекта Selection, их типы и краткое описание.

Таблица 6.1. *Свойства и методы объекта Selection*

Свойства и методы	Тип	Описание
Type	Число	Тип выделенного объекта (обычный текст Type =wdSelectionNormal=2)
StoryType	Число	Тип выделяемого объекта (может быть выделен основной текст, текст в заголовках, комментариях и т. п.)

Таблица 6.1 *(окончание)*

Свойства и методы	Тип	Описание
Text	Строка	Текстовое содержимое
Start	Число	Начальная позиция выделенного объекта
End	Число	Конечная позиция выделенного объекта
Characters	Коллекция	Символы выделенного объекта, их количество
ConvertToTable	Метод	Преобразование выделенного текста в таблицу
Copy	Метод	Копирование текста в буфер обмена
Paste	Метод	Вставка текста из буфера обмена
CopyAsPicture	Метод	Копирование выделенного текста (объекта) в буфер обмена с преобразованием его в графический объект в формате BMP
Cut	Метод	Вырезание выделенного фрагмента текста
Delete(a, b)	Метод	Удаление выделенного фрагмента текста (вызов без аргументов) или удаление фрагмента из b символов начиная с позиции a
Find	Объект	Поиск, поиск и замена в документе
Font	Объект	Шрифт выделенного объекта
InRange(MyRange)	Метод	Проверка вхождения выделенного объекта в объект MyRange
InsertAfter	Метод	Вставка текста после объекта Selection
InsertBefore	Метод	Вставка текста до объекта Selection
Move(Unit, Count)	Метод	Перемещение объекта Selection в документе (перемещение курсора)
SetRange(a, b)	Метод	Выделение текста между позициями a и b
Style	Число	Стиль выделенного текста
TypeText	Метод	Вставка текста на место выделенного объекта или с позиции курсора
Words	Коллекция	Слова в выделенном объекте

Рассмотрим подробней некоторые свойства объекта Selection.

При работе с текстом требуется уверенность в том, что это именно текст, чтобы не применять методы, свойственные тексту, к другим объектам, и

наоборот. Если применить к выделенному тексту методы, свойственные, например, ячейкам таблицы, будет получена ошибка выполнения. Определить тип выделенного объекта позволяет свойство Type. Если Type = wdSelectionNormal, это значит, что выделенный объект является текстом, при этом свойство Text объекта Selection содержит выделенный текст.

Свойства Start и End определяют начальную и конечную позицию (в символах) выделенного фрагмента текста. Если Selection.Start = Selection.End, то размер выделенного объекта равен нулю, т. е. ничего не выделено. Но при этом объект Selection все равно существует. Если Selection.Start = Selection.End = 0, это означает, что курсор находится в начале документа (при условии что наш документ содержит только текст). Используя свойства Start и End, можно установить курсор в любую позицию документа. Если Selection.Start <> Selection.End, то это означает, что выделен фрагмент текста между двумя позициями, значения которых записаны в эти поля. Присваивая им различные значения, можно манипулировать размерами выделенной области текста. Для примера откроем текст с помощью метода Open коллекции Documents и установим значения полей Start и End. Вот фрагмент исходного текста программы.

Задание диапазона выделенного объекта

```
procedure TForm1.Button6Click(Sender: TObject);
begin
W.Selection.Start:=13;
W.Selection.End:=25;
end;
```

Результат выполнения данной процедуры показан на рис. 6.3.

Рис. 6.3. Процедура устанавливает границы выделения

Также нетрудно убедиться, что положение курсора будет изменяться при изменении свойств Start и End объекта Selection. Если их значения установить в ноль, то курсор переместится в начало документа, и переместится в конец документа, когда их значения будут равны количеству символов документа. Следующий фрагмент программного кода показывает, как переместить курсор в начало и конец документа.

Перемещение курсора в начало и конец документа

```
W.Selection.Start:=0;
W.Selection.End:=0;

W.Selection.Start:=W.ActiveDocument.Characters.Count;
W.Selection.End:=W.ActiveDocument.Characters.Count;
```

Манипуляции с полями Start и End — не единственный способ перемещения курсора или выделения объекта.

Рассмотрим метод Move(Unit, Count) объекта Selection. Он позволяет перемещать курсор на определенное количество символов, слов, предложений или абзацев, в конец или начало документа, а также по ячейкам таблицы. Этот метод позволяет достичь начала или конца документа, если вызывать его неоднократно (до тех пор, пока возвращаемое им значение не равно 0). У метода есть два аргумента (табл. 6.2): первый определяет выполняемое действие, а второй — величину и направление перемещения курсора (при отрицательном значении курсор перемещается по направлению к началу текста).

Таблица 6.2. *Действия метода Move при разных значениях аргументов*

Значение аргумента Unit	Выполняемое действие
WdCharacter =1	Переход к следующему символу
WdWord=2	Переход к следующему слову
WdSentence=3	Переход к следующему предложению
WdParagraph=4	Переход к следующему абзацу
WdSection=8	Переход к следующему разделу
WdStory=6	Переход в следующую текстовую область документа
WdCell=12	Переход к следующей ячейки
WdColumn=9	Переход к следующему столбцу
WdRow=10	Переход к следующей строке
WdTable=15	Переход к следующей таблице
WdLine=5	Переход к следующей линии

Если вызвать метод Move без аргументов, то по умолчанию его выполнение переместит курсор вперед по тексту на один символ.

Рассмотрим использование метода Move в синтаксисе языка Delphi. Создадим в форме кнопку и поместим в процедуру обработки ее нажатия следующий программный код.

Перемещение курсора по тексту

```
procedure TForm1.Button15Click(Sender: TObject);
 const wdCharacter=1;
begin
W.Selection.Move(wdCharacter,3);
end;
```

Задавая разные значения первого аргумента (см. табл. 6.2), можно проверить работу этого метода и на более сложных документах, содержащих таблицы, рисунки и т. п.

Рассмотрим другие методы объекта Selection.

Когда курсор установлен в необходимое положение и выделен текст (как объект Range), можно поместить этот текст в буфер обмена или наоборот, вставить текст из буфера обмена на место выделенного текста. Для этого используются два метода объекта Selection — Copy и Paste. Метод Copy копирует выделенный текст в буфер обмена, метод Paste извлекает текст из буфера и вставляет его в текст, начиная от положения курсора. Если до выполнения метода Paste был выделен объект, то текст вставляется на место этого объекта. Эти методы можно применять к объекту любого типа (в данном случае в качестве объекта мы рассматриваем текст).

Синтаксис вызова этих методов в Delphi показан в следующем программном коде.

Использование методов Paste и Copy

```
procedure TForm1.Button8Click(Sender: TObject);
begin
W.Selection.Copy;
end;

procedure TForm1.Button9Click(Sender: TObject);
begin
W.Selection.Paste;
end;
```

Наряду с этими двумя методами можно упомянуть метод CopyAsPicture, который может быть очень полезен в некоторых задачах. Он помещает в буфер обмена графическое изображение (битовый рисунок) выделенного объ-

екта. Его также можно вызывать из приложений Delphi. Рассмотрим следующий пример.

Использование метода CopyAsPicture

```
procedure TForm1.Button16Click(Sender: TObject);
begin
W.Selection.CopyAsPicture;
end;
```

Выделим часть текста (рис. 6.4) и вызовем вышеописанную процедуру нажатием кнопки.

Рис. 6.4. Выделим текст перед тем, как вызвать метод CopyAsPicture

Запустив графический редактор Paint и выполнив операцию вставки из буфера обмена, получим результат, показанный на рис. 6.5.

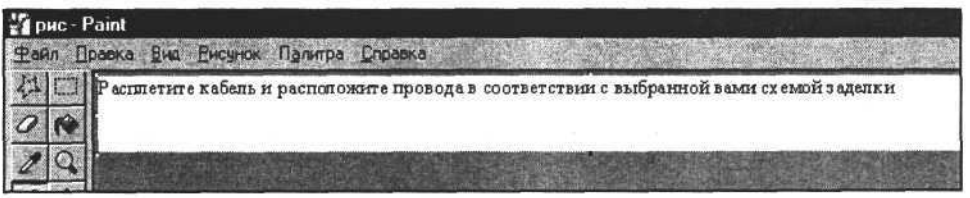


Рис. 6.5. Анализ скопированного в буфер как рисунок (методом CopyAsPicture) выделенного фрагмента документа

Выделенный текстовый фрагмент можно не только скопировать и преобразовать в графический объект, но и преобразовать в таблицу, как мы это часто делаем, редактируя текст. Используя автоматизацию в Word, в отличие от обычного редактирования, мы достигнем большей скорости и гибкости для преобразования таблиц, набранных в обычном текстовом режиме с разделителями, в таблицы Word.

Для этого предназначен метод ConvertToTable объекта Selection. У этого метода имеется несколько аргументов, которые задают форматы и режимы преобразования. Пожалуй, один из основных аргументов — Separator (символ-разделитель). В таблицах, набранных в DOS, таким символом был элемент псевдографики. Используя этот аргумент, можно без труда преобразовать таблицы, оформленные до этого в обычном текстовом редакторе. Другой не менее важный параметр — Format (формат представления таблицы). Этот аргумент представлен переменной типа integer и может принимать несколько значений. При преобразовании текста в таблицу можно также задать количество столбцов (NumColumns), строк (NumRows) и начальную ширину столбцов (InitialColumnWidth). Если не задавать все эти параметры, то преобразование произойдет в режиме "по умолчанию".

Рассмотрим спецификацию вызова метода ConvertToTable в Visual Basic.

Вызов метода ConvertToTable в Visual Basic

```
Selection.ConvertToTable(Separator, NumRows, NumColumns,
  InitialColumnWidth, Format, ApplyBorders, ApplyShading,
  ApplyFont, ApplyColor, ApplyHeadingRows, ApplyLastRow,
  ApplyFirstColumn, ApplyLastColumn, AutoFit)
```

Вызов этого метода в приложении Delphi будет выглядеть иначе в соответствии с принятой спецификацией языка программирования Pascal.

Например, требуется преобразовать выделенный текст в таблицу из пяти столбцов. В качестве разделителя текста используем символ пробела, формат таблицы зададим константой wdTableFormatGrid2=17 и используем значения выбранных характеристик в качестве аргументов метода ConvertToTable. Смотрите исходный текст.

Преобразование выделенного текста в таблицу (Delphi)

```
procedure TForm1.Button17Click(Sender: TObject);
 const wdTableFormatGrid2=17;
begin
W.Selection.ConvertToTable(Separator:=' ',NumRows:=5,NumColumns:=5,
  Format:=wdTableFormatGrid2);
end;
```

Откроем произвольный документ и выделим фрагмент текста, как на рис. 6.6.

После применения к выделенному тексту метода ConvertToTable получим результат, представленный на рис. 6.7.

Рис. 6.6. Выделяем фрагмент текста

Отказ	от	повторного	прочтения	текста
повышает	скорость	чтения	в	два
раза,	а	качество	понимания	прочитанного
текста	повышает	в	три	раза.

Рис. 6.7. Текст преобразован в таблицу

Возможно, нам придется удалять некоторые фрагменты текста. Это можно сделать несколькими способами. Поскольку мы рассматриваем работу с объектом Selection, то используем для этой цели его метод Delete или Cut.

Вызов методов Delete и Cut для удаления выделенного фрагмента текста

```
W.Selection.Delete;
W.Selection.Cut;
```

Отличие этих методов заключается в том, что первый метод удаляет выделенный объект целиком, а второй помещает его в буфер обмена, а только потом удаляет. Второе отличие этих методов определяется их действием на текст, если последний не выделен. В этом случае первый метод удаляет один символ после курсора, а второй просто вызовет ошибку выполнения.

Примечание

Для удаления выделенного фрагмента можно также воспользоваться объектом Range. Так как у объектов Range и Selection много общего, методы удаления фрагментов текста у них одноименные:

```
W.ActiveDocument.Range(1,20).Delete;
W.ActiveDocument.Range(1,20).Cut;
```

При создании документов не обойтись и без методов, позволяющих вставлять текст. В предыдущей главе были рассмотрены два метода объекта Range — InsertAfter и InsertBefore. Эти методы есть и у объекта Selection. Дополнительно можно рассмотреть метод TypeText, позволяющий вставлять текст начиная с позиции курсора (если нет выделенного текста) или замещать выделенный текст. У этого метода только один аргумент — текстовая строка.

Вызов этого метода в среде Delphi выглядит так:

Использование метода TypeText

```
procedure TForm1.Button18Click(Sender: TObject);
begin
W.Selection.TypeText('<-- Заменяем выделенный фрагмент данным текстом -->');
end;
```

Вставить текст в положение курсора или вместо выделенной области можно и более простым способом. Достаточно записать нужный текст в свойство Text объекта Selection, при этом результат будет таким же, как при выполнении метода TypeText.

Использование свойства Text

```
procedure TForm1.Button7Click(Sender: TObject);
begin
W.Selection.Text:= <-- Заменяем выделенный фрагмент данным текстом -->;
end;
```

Результат применения любой из этих двух процедур к выделенному фрагменту текста (см. рис. 6.2) представлен на рис. 6.8.

Иногда требуется получить размер выделенной области (фрагмента) в символах или словах, скопировать определенные символы или слова, а также задать или получить параметры шрифта для выделенной области текста. Выделенная область (определяемая объектом Selection), как и область текста (определяемая объектом Range), обладает аналогичными одноименными свойствами. Коллекция Characters, входящая в объект Selection, позволяет работать с отдельными символами области, в частности получить ее размер. Коллекция Words объекта Selection содержит отдельные слова области и определяет ее размер в словах. Объект Font, принадлежащий выделенной области, определяет или задает для нее параметры шрифта.

Еще один полезный параметр объекта Selection — Style типа integer, содержащий информацию о стиле выделенного текста. Если ему присвоить какое-либо значение из допустимых (существует около сотни стилей), то выделенный текст будет отображен в соответствующем стиле. Можно разработать и использовать свой стиль.

Разберем следующий пример: выделим текст и выполним оператор W.Selection.Style:=WdStyleHyperlinkFollowed; (значение константы WdStyleHyperlinkFollowed = −87). Выполнение данного оператора установит стиль текста, который представлен на рис. 6.9.

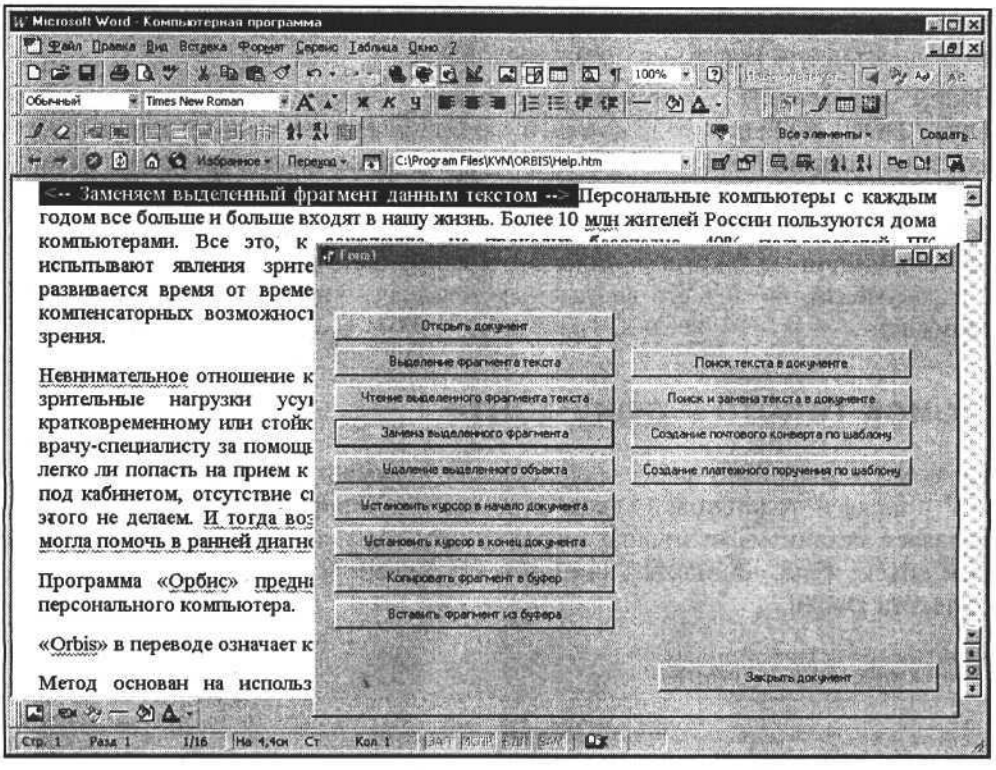


Рис. 6.8. Результат выполнения метода замены выделенного фрагмента текста в документе

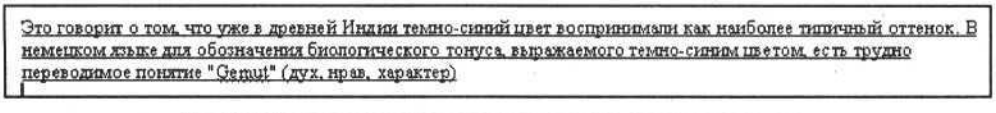


Рис. 6.9. Установка стиля для выделенного фрагмента текста

Шаблон документа

Мы рассмотрели, как выделить нужный фрагмент текста, удалить, скопировать, изменить его или вставить фрагмент текста в документ. Но этого еще недостаточно для того, чтобы сформировать простой документ по шаблону. Конечно, можно последовательно добавлять в документ фрагменты текста, изменять его стили и таким образом привести документ к нужному виду.

Но как быть с документами, у которых определенные записи должны находиться в определенных позициях документа? Очевидным решением этой задачи является создание документа на базе шаблона, где нужный текст будет подставляться вместо некоторых текстовых констант. Наша программа,

создающая такой документ, должна уметь находить эти текстовые константы и подставлять вместо них реальные значения.

Дадим определение. *Шаблон документа* это документ (файл с расширением DOT, DOC, RTF или др.), который может содержать произвольный текст, таблицы, надписи и другие объекты, а также некоторые определенные текстовые константы, которые будут использованы программой формирования документа для поиска и подстановки на их место реального текста (причем эти определенные текстовые константы могут быть расположены как в тексте документа, так и в его визуальных объектах — надписях, таблицах, диаграммах).

Поиск текста в документе

Ко всему описанному ранее для создания документа нам не хватает функций поиска и подстановки текста в документ. Хорошо, что объект Selection обладает механизмами поиска текста — ими мы и воспользуемся. Рассмотрим объект Find, принадлежащий объекту Selection. Вот фрагмент исходного текста Delphi.

Поиск текста в документе

```
procedure TForm1.Button11Click(Sender: TObject);
 var text_:string;
begin
text_:='';
text_:=InputBox('Введите текст для поиска',text_,text_);
W.Selection.Find.Forward:=true;
W.Selection.Find.Text:=text_;
if W.Selection.Find.Execute then messagebox(handle,'Поиск текста ↵
завершен успешно!','Внимание!',0);
end;
```

Запустим программу нажатием кнопки и введем в специальном окне фрагмент текста, который нужно найти в документе (рис. 6.10).

В результате выполнения процедуры искомый фрагмент текста будет выделен (рис. 6.11).

В данном примере мы задали текст в режиме поиска "от начала к концу документа", в поля объекта Find поместили нужные данные, которые установили необходимый режим, а затем вызвали метод Execute, который и выполнил поиск текста. Этот метод возвращает значение True/False (при успешном/неуспешном поиске).

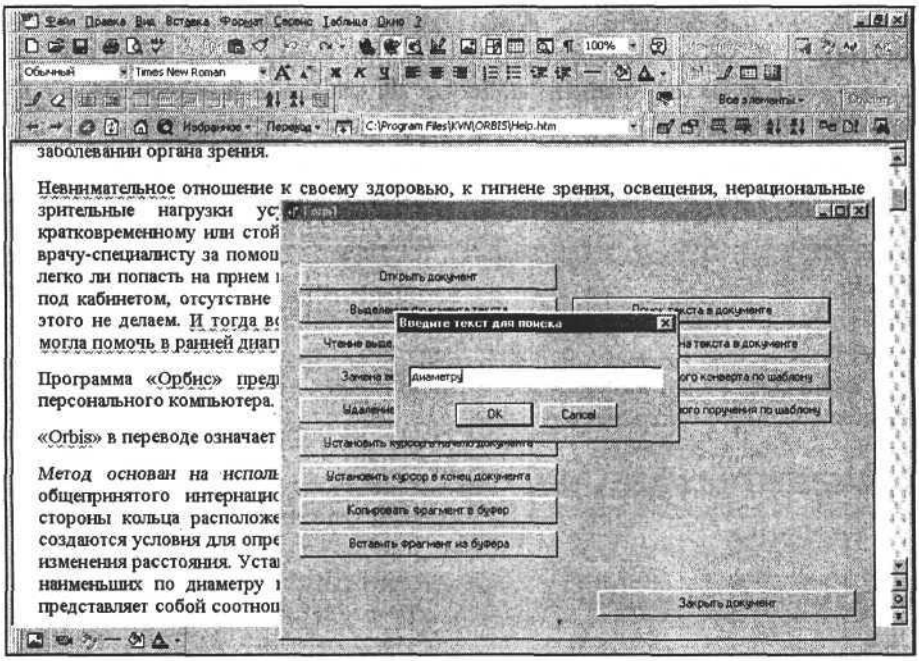


Рис. 6.10. При выполнении метода Find вводится фрагмент текста для поиска

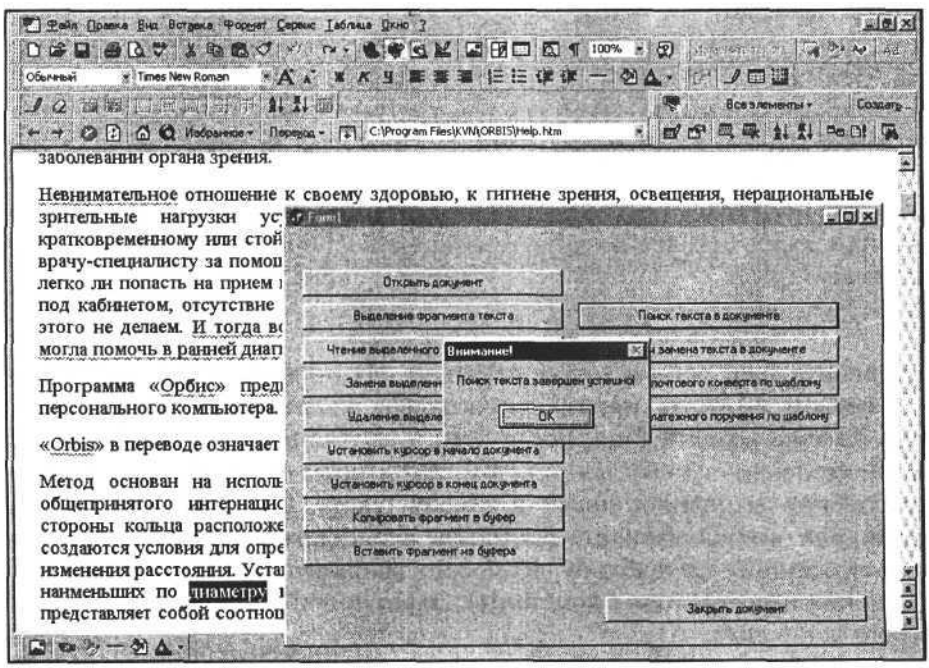


Рис. 6.11. Результат успешного поиска в виде выделенного текста в документе

Очевидно, что набора методов, позволяющих осуществлять поиск и подстановку текста на место выделенного фрагмента, будет достаточно для создания документов на основании шаблона. В качестве примера рассмотрим следующие процедуры.

Поиск и замена текста (вариант 1)

```
procedure TForm1.Button11Click(Sender: TObject);
 var text_:string;
begin
text_:='';
text_:=InputBox('Введите текст для поиска',text_,text_);
W.Selection.Find.Forward:=true;
W.Selection.Find.Text:=text_;
if W.Selection.Find.Execute then W.Selection.Text:='Фрагмент для замены';
end;
```

Поиск и замена текста (вариант 2)

```
procedure TForm1.Button12Click(Sender: TObject);
 const wdReplaceAll=2;
 var text_:string;
begin
text_:='';
text_:=InputBox('Введите текст для поиска',text_,text_);
W.Selection.Find.Forward:=true;
W.Selection.Find.Text:=text_;
if W.Selection.Find.Execute(Replace:=wdReplaceAll) then
messagebox(handle,pchar('Поиск и замена текста "'+text_+'" завершена!'),
  'Внимание!',0);
end;
```

Оба варианта дают один и тот же результат. Их отличия заключаются в том, что в первом варианте производится поиск фрагмента с использованием метода Execute объекта Find, после чего, в случае удачной попытки, выделенный фрагмент заменяется текстом. Во втором варианте метод Execute одновременно выполняет поиск и замену текста. Если данную процедуру (первый или второй вариант) повторять неоднократно, применяя к документу, созданному по шаблону, то можно заполнить его реальными данными, заменив ими текстовые константы. Этого достаточно для формирования простого документа.

Почтовый конверт

Возможно, наиболее распространенный и удачный пример использования описанных объектов и методов — небольшой фрагмент программы для формирования надписей на почтовых конвертах. Работая в среде Delphi с базами данных, вы с легкостью сможете разработать свою программу для их подготовки и получите массу благодарностей от людей, которым приходилось делать это вручную.

Сначала создадим документ (как на рис. 6.12). Сохраним его в формате "Шаблон документа", чтобы использовать для создания необходимого документа. Текст в этом документе (###обратный индекс& и т. п.) представляет собой текстовые константы, которые будут заменены текстом в процессе формирования документа.

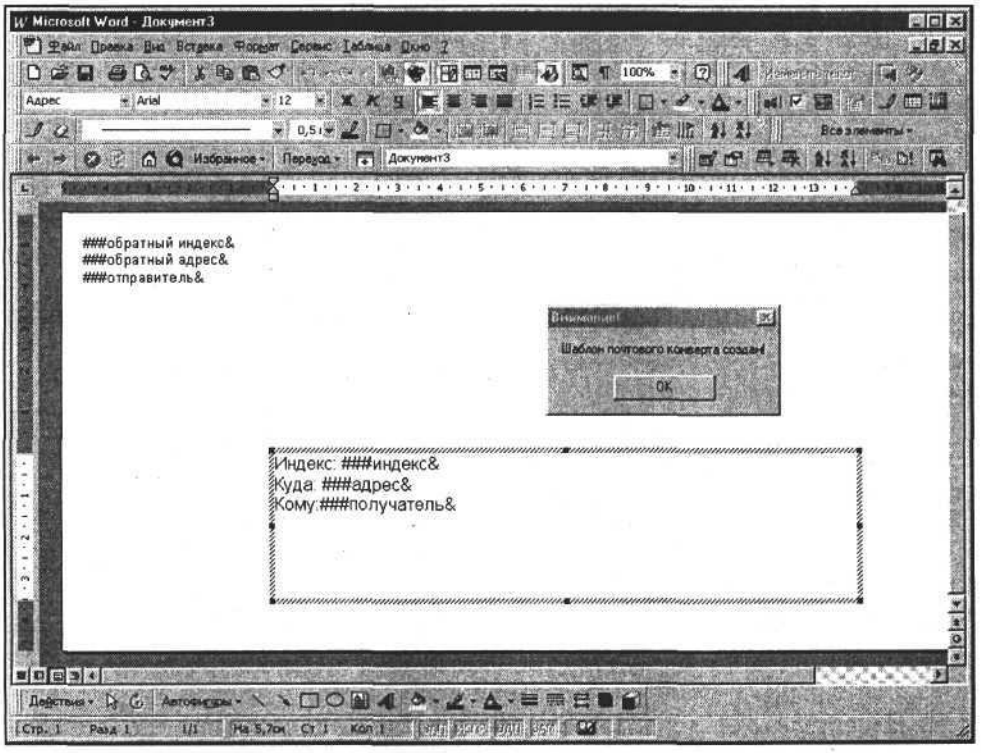


Рис. 6.12. Шаблон надписей для почтового конверта

После создания шаблона скопируем файл с расширением DOT (формат "Шаблон документа") в рабочую папку программы или папку MS Office. После этого можно приступить к формированию надписей на конверте. Рассмотрим следующий фрагмент программы.

Создание текста на конверте с использованием методов поиска и подстановки текста

```
procedure TForm1.Button13Click(Sender: TObject);
 const wdReplaceAll=2;
begin
// Создаем новый документ по шаблону
W.documents.Add(ExtractFileDir(Application.ExeName)+↵
   '\Шаблон простого конверта.dot');

// Прямой адрес
W.Selection.Find.Text:='###индекс&';
W.Selection.Find.Replacement.Text:='350049';
W.Selection.Find.Execute(Replace:=wdReplaceAll);
W.Selection.Find.Text:='###адрес&';
W.Selection.Find.Replacement.Text:='Краснодар, ул. Севастопольская, ↵
 д. 3, кв. 1230';
W.Selection.Find.Execute(Replace:=wdReplaceAll);

W.Selection.Find.Text:='###получатель&';
W.Selection.Find.Replacement.Text:='Иванов Иван Иванович';
W.Selection.Find.Execute(Replace:=wdReplaceAll);

// Обратный адрес
W.Selection.Find.Text:='###обратный индекс&';
W.Selection.Find.Replacement.Text:='198005';
W.Selection.Find.Execute(Replace:=wdReplaceAll);
W.Selection.Find.Text:='###обратный адрес&';
W.Selection.Find.Replacement.Text:='Санкт-Петербург, Измайловский пр., ↵
 д. 29, кв. '+'1112';
W.Selection.Find.Execute(Replace:=wdReplaceAll);

W.Selection.Find.Text:='###отправитель&';
W.Selection.Find.Replacement.Text:='Петрова Альбина Васильевна';
W.Selection.Find.Execute(Replace:=wdReplaceAll);
end;
```

После выполнения этой процедуры мы получим результат, как на рис. 6.13. Чтобы использовать этот результат, необходимо сохранить полученный документ и распечатать его. Можно выполнять данную процедуру в цикле, получая исходные данные из базы данных и сохраняя документы под уникальными именами или распечатывая их.

Пожалуй, данный пример — самый простой случай создания и использования шаблона. Рассмотрим еще один пример, основанный на использовании

шаблона документа. Его отличия заключаются в том, что для размещения текстовых констант используются ячейки таблицы. Этот пример может быть полезен вам в практическом плане, если вы создаете или поддерживаете разработанную в среде Delphi бухгалтерскую программу. Подобным образом, используя таблицы, надписи и текстовые константы для подстановки, можно изготовить шаблон документа практически любой сложности.

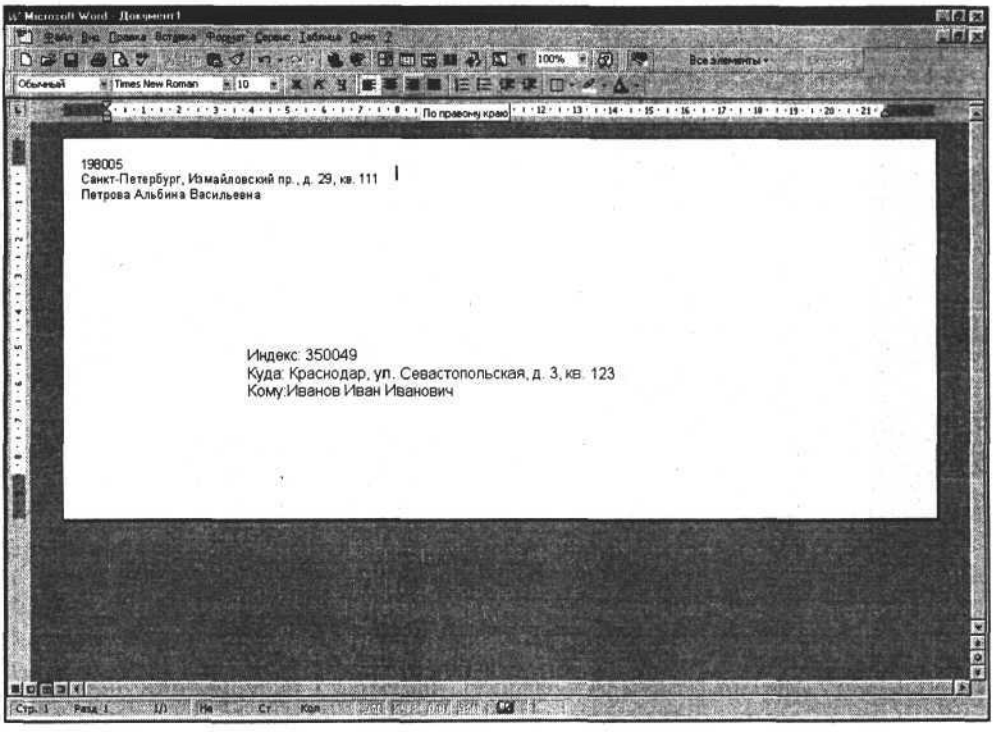


Рис. 6.13. Сформированные программой надписи для почтового конверта

Платежное поручение

Разработаем шаблон платежного поручения. Чтобы облегчить себе задачу, воспользуемся справочно-правовой системой "Гарант" и получим форму платежного поручения — документ в формате DOC. Отредактируем этот документ и приведем его к нужному виду (рис. 6.14). Далее сохраним его в формате DOT для последующего использования в процедуре формирования выходной формы документа "платежное поручение".

Как видно из рис. 6.14, преобразование документа в шаблон заключается в том, что в некоторые ячейки внесен текст типа "###Сумма прописью&".

Эти фрагменты текста и представляют собой текстовые константы, которые будут использованы для подстановки информации из программы.

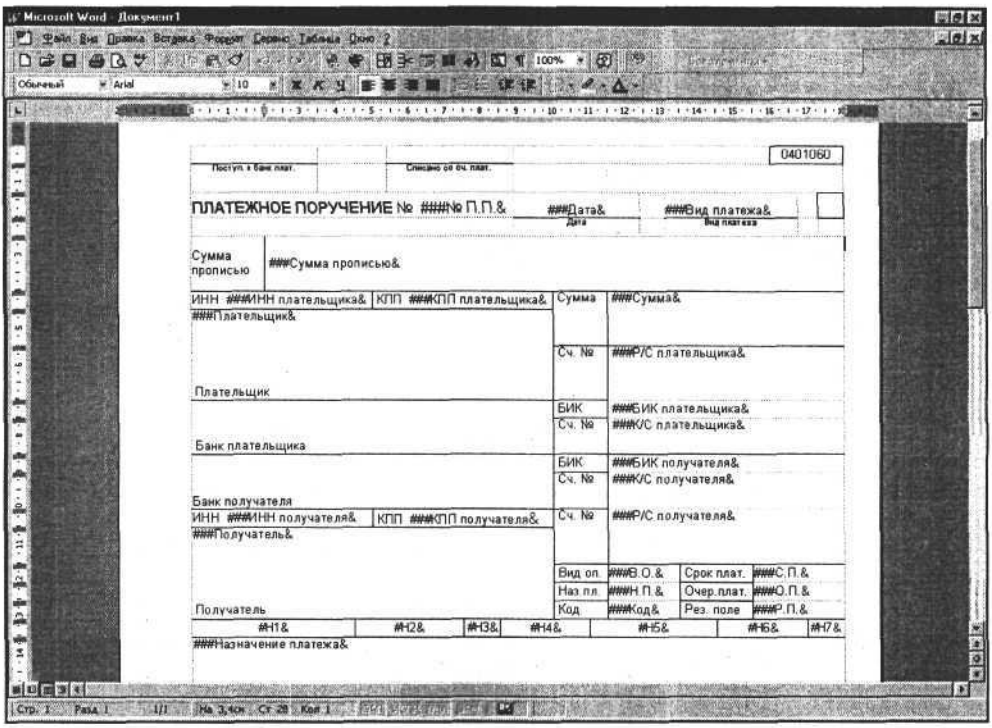


Рис. 6.14. Вид шаблона документа "Платежное поручение"

Используем этот шаблон для создания документа. Следующий фрагмент исходного текста демонстрирует, как это сделать.

Формирование платежного поручения

```
// Данная функция позволяет находить строку FindText
// и подставлять на ее место строку ReplacementText.
//
Function FindAndInsert(FindText,ReplacementText:string):boolean;
 const wdReplaceAll=2;
begin
W.Selection.Find.Text:=FindText;
W.Selection.Find.Replacement.Text:=ReplacementText;
FindAndInsert:=W.Selection.Find.Execute(Replace:=wdReplaceAll);
end;
```

```
// Данная процедура заполняет шаблон документа.
//
Pocedure TForm1.Button14Click(Sender: TObject);
begin
// Создаем новый документ по шаблону
W.documents.Add(ExtractFileDir(Application.ExeName)+'\Шаблон платежного'+' поручения.dot');
// Подставляем тексте
FindAndInsert('###№ П.П.&','1');
FindAndInsert('###Дата&',datetostr(date));
FindAndInsert('###Вид платежа&','почтой');
FindAndInsert('###Сумма прописью&','Двести пятьдесят рублей сорок копеек');
FindAndInsert('###Сумма&','250,40');
FindAndInsert('###ИНН плательщика&','0000000000');
FindAndInsert('###КПП плательщика&','000000000011');
FindAndInsert('###Плательщик&','ЗАО Селена');
FindAndInsert('###Р/С плательщика&','00000000000000000000');
FindAndInsert('###БИК плательщика&','000000');
FindAndInsert('###К/С плательщика&','00000000000000000000');

FindAndInsert('###ИНН получателя&','1111111111');
FindAndInsert('###КПП получателя&','111111111100');
FindAndInsert('###БИК получателя&','111111');
FindAndInsert('###К/С получателя&','11111111111111111111');
FindAndInsert('###Р/С получателя&','11111111111111111111');
FindAndInsert('###Получатель&','ЗАО Комета');

FindAndInsert('###В.О.&','');
FindAndInsert('###Н.П.&','');
FindAndInsert('###Код&','');
FindAndInsert('###С.П.&','');
FindAndInsert('###О.П.&','');
FindAndInsert('###Р.П.&','');

FindAndInsert('#Н1&','');
FindAndInsert('#Н2&','');
FindAndInsert('#Н3&','');
FindAndInsert('#Н4&','');
FindAndInsert('#Н5&','');
FindAndInsert('#Н6&','');
FindAndInsert('#Н7&','');
FindAndInsert('###Назначение платежа& ','Оплата за поставку товара');
end;
```

Результат выполнения данной процедуры отображен на рис. 6.15.

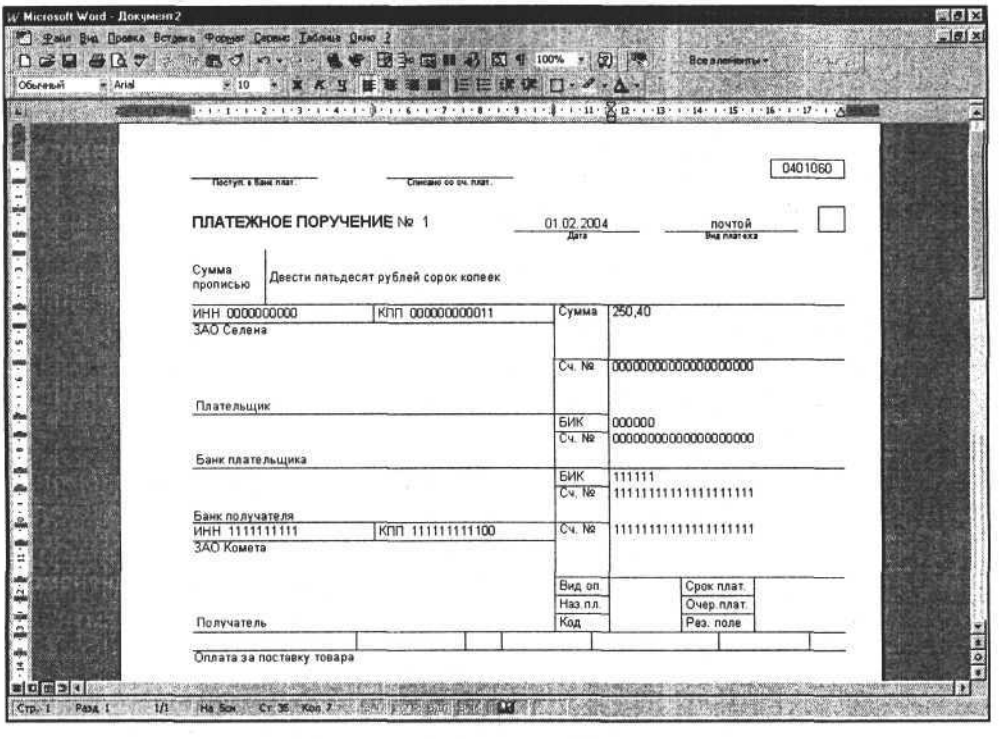


Рис. 6.15. Сформированный документ "Платежное поручение"

Далее можно переходить к печати документа.

Автоматизацию печати рассмотрим в следующей главе.

ГЛАВА 7

Создание таблиц и работа с ними

В этой главе рассмотрены следующие темы:

- создание, выделение и удаление таблиц в документе;
- форматы таблиц;
- изменение положения таблицы и ее строк;
- границы и заливка ячеек таблиц;
- добавление и удаление строк и столбцов таблицы;
- текст в ячейках таблицы;
- задание шрифта текста в документе и в таблице;
- направление текста;
- пример разработки табличного документа — бланка счета-фактуры.

Создание, выделение и удаление таблиц в документе

Документ Word может содержать таблицы, которые как объекты объединены в коллекцию Tables. Как вытекает из вышесказанного, обычно эта коллекция принадлежит объекту "документ". Рассмотрим ее основные свойства и методы.

Свойство Count:integer содержит информацию о количестве таблиц в коллекции. Если в документе нет ни одной таблицы, то Count содержит ноль.

Метод Add добавляет новую таблицу в коллекцию. При создании таблицы определяются область, где будет создана таблица, и ее основные параметры — количество столбцов и строк. Хотя коллекция таблиц и принадлежит документу, тем не менее сама таблица может быть создана на любом объекте или месте документа, на котором это возможно. Объект Item(tab) — эле-

мент коллекции и представляет собой таблицу с порядковым номером tab данной коллекции.

Для демонстрации свойств и методов коллекции Tables создадим новый документ и в нем создадим таблицу. Используем метод Add. Его спецификация в VB имеет следующий вид: expression.Add(Range, NumRows, NumColumns), где Range — область, на которой создается таблица, NumRows и NumColumns — количество строк и столбцов создаваемой таблицы.

Рассмотрим использование метода Add в среде Delphi на примере приведенного ниже фрагмента программного текста.

Создание таблицы

```
procedure TForm1.Button2Click(Sender: TObject);
begin
W.ActiveDocument.Tables.Add(Range:=W.ActiveDocument.Range, NumRows:=2,
                            NumColumns:=3);
end;
```

Данная процедура создаст таблицу непосредственно в документе. Первый аргумент метода определяет область, где будет создана таблица, — весь документ. Следующие аргументы определяют количество строк и столбцов. Если попробовать вторично выполнить эту процедуру, то будет сгенерирована ошибка (рис. 7.1).

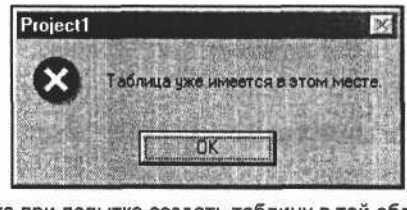


Рис. 7.1. Ошибка при попытке создать таблицу в той области документа, где она уже есть

Для корректного создания следующей таблицы необходимо в качестве первого аргумента метода Add указать область, которая свободна и не содержит таблицу. Для задания этой области можно использовать метод Range, который рассматривался ранее, или, например, свойство End объекта Range.

Рассмотрим следующую процедуру, которая создает новую таблицу в документе. В отличие от предыдущей, для создания новой таблицы она использует область в конце документа.

Создание таблицы в конце документа

```
procedure TForm1.Button3Click(Sender: TObject);
 var MyRange:variant;
```

```
begin
MyRange:=W.ActiveDocument.Range(W.ActiveDocument.Range.End-1,
                                W.ActiveDocument.Range.End-1);
W.ActiveDocument.Tables.Add(Range:=MyRange, NumRows:=2, NumColumns:=3);
W.ActiveDocument.Range.InsertAfter(' ');
end;
```

Данная процедура использует область в конце документа, в которой и создается новая таблица. За вновь созданной таблицей вставляется пробел, который разделяет таблицы между собой. Если разделения нет, то вновь создаваемые таблицы будут добавляться как строки к существующим таблицам.

Когда создано несколько таблиц, мы можем определить их количество. Это значение содержится в свойстве Count коллекции Tables. Приведенная ниже процедура выводит список таблиц в компонент ListBox.

Получение списка таблиц в документе

```
procedure TOKBottomDlg.FormShow(Sender: TObject);
 var a_:integer;
begin
ListBox1.Items.Clear;
for a_:=1 to Form1.W.ActiveDocument.Tables.Count do begin
    ListBox1.Items.Add('Таблица - '+inttostr(a_));
    end;
end;
```

В результате выполнения этой процедуры компонент ListBox1 будет содержать список таблиц (рис. 7.2).

Используя объект Item коллекции Tables и порядковый номер таблицы, можно выделить любую таблицу из всего списка таблиц документа для дальнейших манипуляций. Для выделения таблицы, которая содержится в коллекции, используется метод Select объекта Item. Этот объект содержит все методы, коллекции и объекты, присущие таблице как объекту.

Следующий фрагмент программы демонстрирует использование метода Select и метода Delete, удаляющего выбранную таблицу.

Выделение и удаление таблицы коллекции Tables

```
procedure TOKBottomDlg.Button1Click(Sender: TObject);
begin
Form1.W.ActiveDocument.Tables.Item(ListBox1.itemindex+1).Select;
end;
```

```
procedure TOKBottomDlg.Button2Click(Sender: TObject);
begin
Form1.W.ActiveDocument.Tables.Item(ListBox1.itemindex+1).Delete;
end;
```

Для демонстрации этих методов предназначено текстовое приложение, представленное на сопроводительном диске книги. Процесс выполнения данных процедур может выглядеть примерно так, как показано на рис. 7.2.

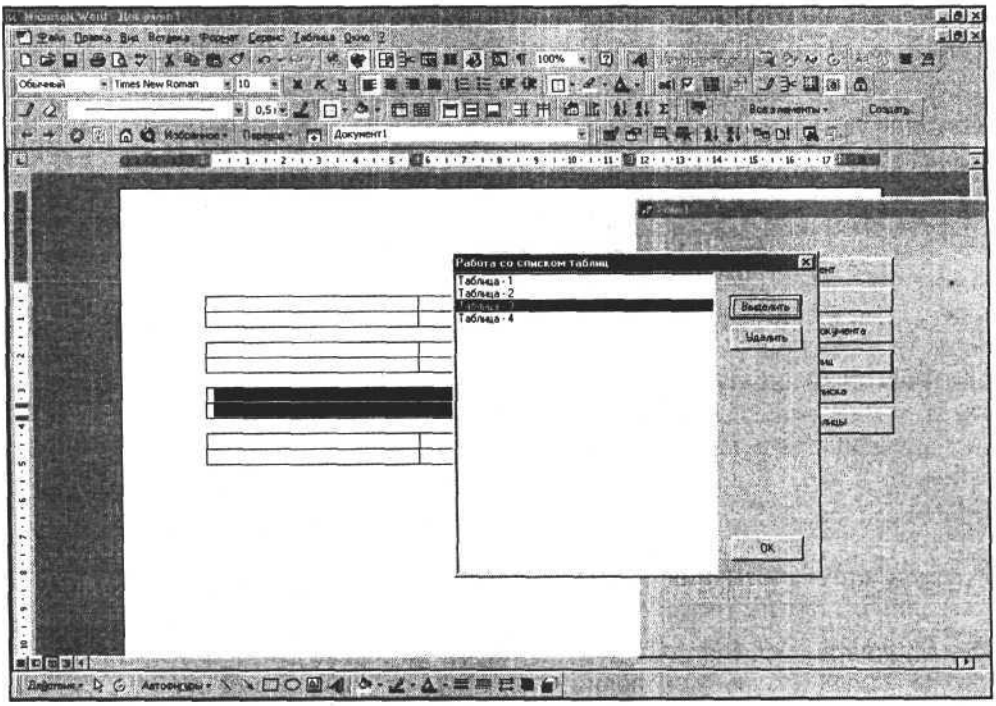


Рис. 7.2. Работа со списком таблиц в документе, выделение и удаление

Форматы таблиц

В результате использования метода Add мы создали простую таблицу, основными параметрами которой являются ее местонахождение, а также количество столбцов и строк. Но в документах таблицы могут иметь вид, далеко отличающийся от того, что мы получили. Чтобы изменить вид таблицы, достаточно изменить толщину, стиль и цвет линий, рисунок и цвет заливки ячейки, цвет ячейки, расположить текст определенным образом. Тогда мы можем получить таблицу, удовлетворяющую определенным требованиям отображения содержащейся в ней информации.

Есть методы, позволяющие настроить индивидуально каждую строку, столбец или ячейку. Их мы рассмотрим позже. В Word есть несколько заданных форматов таблиц, использование которых приведет к определенному виду все ячейки таблицы.

Формат таблицы задается с помощью метода AutoFormat объекта "таблица". Вот его спецификация в VB:

Метод AutoFormat

```
AutoFormat(Format, ApplyBorders, ApplyShading, ApplyFont, ApplyColor,
           ApplyHeadingRows, ApplyLastRow, ApplyFirstColumn,
           ApplyLastColumn, AutoFit)
```

Аргументы метода и их назначение перечислены в табл. 7.1.

Таблица 7.1. *Аргументы метода AutoFormat*

Аргумент	Тип	Назначение
Format	Число	Формат таблицы
ApplyBorders	Логический	True — задать границы согласно спецификации формата
ApplyShading	Логический	True — применить текстуру согласно спецификации формата
ApplyFont	Логический	True — применить заданный форматом шрифт
ApplyColor	Логический	True — применить заданный форматом цвет
ApplyHeadingRows	Логический	True — использовать свойства для верхней строки согласно спецификации формата
ApplyLastRow	Логический	True — использовать свойства для последней строки согласно спецификации формата
ApplyFirstColumn	Логический	True — использовать свойства для первого столбца согласно спецификации формата
ApplyLastColumn	Логический	True — использовать свойства для последнего столбца согласно спецификации формата
AutoFit	Логический	True — изменять ширину ячейки согласно ширине введенного текста

Первый аргумент метода Format:integer определяет формат, который нужно задать таблице. Он может принимать значения от 0 до 39 (в Office 97), соответствующие различным предопределенным форматам таблицы. Остальные

аргументы определяют применение или отказ от применения некоторых опций этого метода.

Используем метод AutoFormat для изменения внешнего вида созданных таблиц. Для этого создадим форму, расположим в ней элементы управления и следующий программный текст, который добавим в процедуру отклика на выбор в списке компонента ListBox1 и в процедуру отклика на нажатие кнопки Button1.

Задание формата таблицы

```
var table:variant;

procedure TOKBottomDlg3.ListBox1Click(Sender: TObject);
begin
table:=Form1.W.ActiveDocument.Tables.item(ListBox1.itemindex+1);
NumRows.Value:=table.Rows.Count;
NumColumns.Value:=table.Columns.Count;
end;

procedure TOKBottomDlg3.Button1Click(Sender: TObject);
begin
table.AutoFormat(Format:=format.value,
                 ApplyBorders:=ApplyBorders.Checked,
                 ApplyShading:=ApplyShading.Checked,
                 ApplyFont:=ApplyFont.Checked,
                 ApplyColor:=ApplyColor.Checked,
                 ApplyHeadingRows:= ApplyHeadingRows.Checked,
                 ApplyLastRow:=ApplyLastRow.Checked,
                 ApplyFirstColumn:= ApplyFirstColumn.Checked,
                 ApplyLastColumn:=ApplyLastColumn.Checked,
                 AutoFit:=AutoFit.Checked);
end;
```

Первая процедура выбирает таблицу, используя индекс компонента ListBox1, и записывает в переменную table:variant ссылку на эту таблицу. Вторая процедура работает с table как с объектом "таблица", изменяя ее формат согласно состоянию элементов управления (рис. 7.3). Заполним элементы формы, значения которых используем в качестве аргументов метода AutoFormat, и выполним этот метод, предварительно выбрав одну из таблиц документа. В итоге получим результат, показанный на рис. 7.3. Выбранный в данном примере формат соответствует значению аргумента Format=wdTableFormatColumns4 (wdTableFormatColumns4=13).

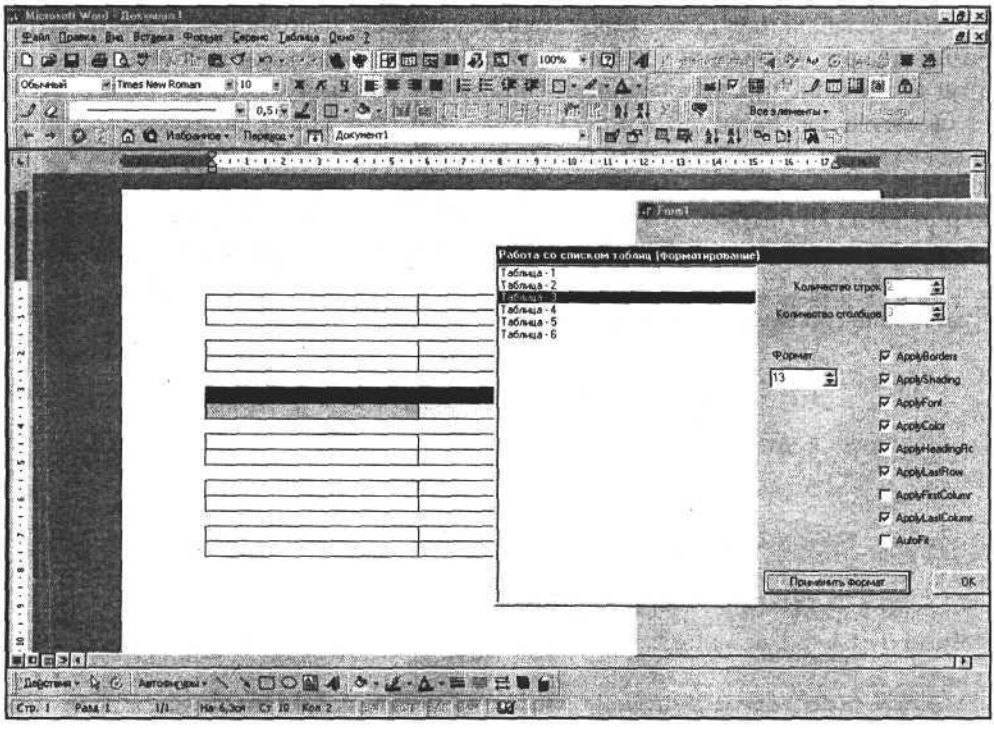


Рис. 7.3. Задание формата таблицы

Изменение положения таблицы и ее строк

Как ранее было сказано, таблица создается в определенной области, которая определяется как аргумент метода Add. Таблица вставляется в промежуток между символами текста или накладывается на область текста. Когда таблица создана, мы можем изменить ее положение относительно левой границы документа или по вертикальному направлению в документе. Положение таблицы по вертикали определяется текстом, расположенным в документе до таблицы. Положение таблицы относительно левой границы страницы документа определяется свойствами самой таблицы. Рассмотрим эти свойства подробней.

Таблица включает в себя коллекции, объекты и другие свойства. Коллекция Rows объекта "таблица" определяет набор ее строк и общие свойства, присущие им. Свойство LeftIndent этой коллекции определяет положение таблицы относительно левой границы листа. Это свойство имеет числовой тип и может принимать как положительные, так и отрицательные значения.

Используя значение LeftIndent, можно не только задать, но и изменить горизонтальное положение таблицы. Как и сама коллекция Rows, элемент этой коллекции, представляющий собой одну указанную строку, также имеет свойство LeftIndent. В данном случае оно определяет положение строки относительно левой границы документа. Остается добавить, что доступ к элементу коллекции Rows осуществляется через объект Item(i), где i — номер строки. Попробуем изменить горизонтальное положение таблицы и выбранной строки, для этого используем следующий программный текст:

Изменение горизонтального положения таблицы

```
procedure TOKBottomDlg4.ListBox1Click(Sender: TObject);
begin
table:=Form1.W.ActiveDocument.Tables.item(ListBox1.itemindex+1);
end;

procedure TOKBottomDlg4.LeftPosChange(Sender: TObject);
begin
table.Rows.LeftIndent:=LeftPos.value;
end;

procedure TOKBottomDlg4.LeftPosRow1Change(Sender: TObject);
begin
table.Rows.Item(1).LeftIndent:=LeftPosRow1.value;
end;
```

Результат выполнения программы представлен на рис. 7.4.

На рис. 7.4 также представлена форма, в которой задаются эти изменения. Компонент ListBox1 выбирает таблицу из списка и помещает ссылку на таблицу в переменную table типа variant. Компоненты LeftPos и LeftPosRow1 задают положение всей таблицы и первой строки соответственно.

Рассмотрим еще некоторые свойства коллекций и элементов коллекций строк и столбцов, которые в большинстве своем аналогичны. Есть только различия, определяемые расположением этих элементов таблицы — если строки имеют параметр "высота", то столбцы имеют параметр "ширина".

В состав коллекций Rows и Columns входит свойство, содержащее количество строк и столбцов таблицы Count.

Доступ к отдельным строкам и столбцам осуществляется посредством объекта Item(). Чтобы задать высоту строки, используем свойство Height элемента коллекции Rows.Item(). Например: table.Rows.Item(1).Height:=21;. Для задания высоты одновременно всех строк таблицы используйте оператор table.Rows.Height:=21;. Если попытаться задать слишком маленькую высоту

строки, получим ошибку выполнения. Для задания ширины столбца используется свойство Width элемента коллекции Columns.Item(). Например, для задания ширины первого столбца используем следующий оператор Delphi: table.Columns.Item(1).Width:=43;. Для задания одинаковой ширины всех столбцов используйте оператор table.Columns.Width:=43;.

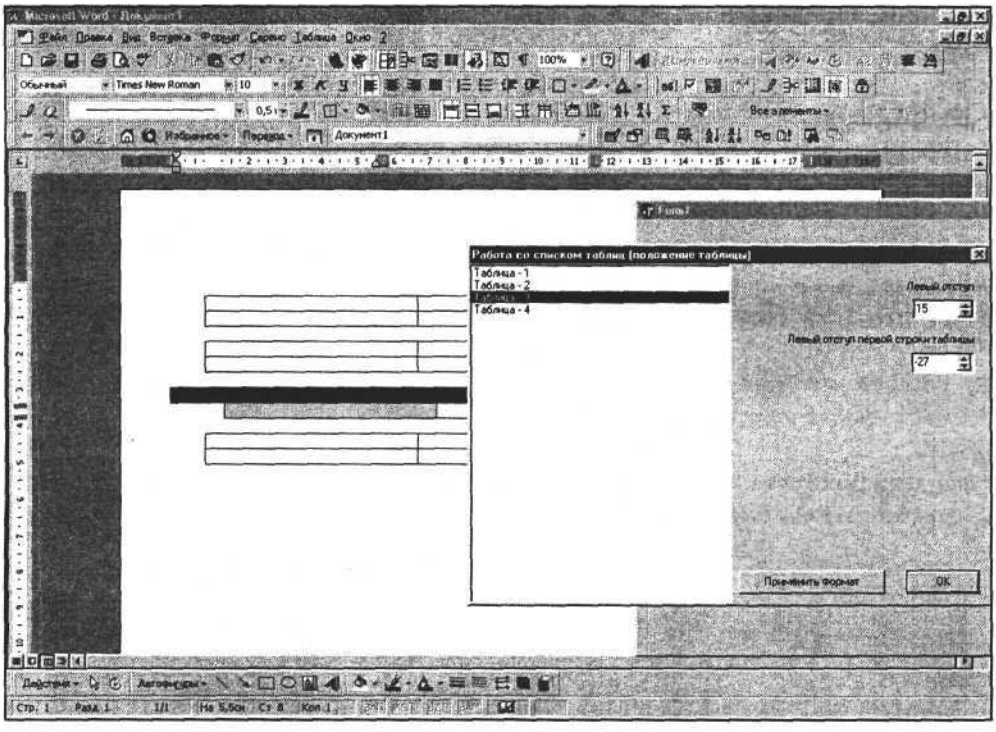


Рис. 7.4. Задание положения таблицы относительно левой границы документа

Границы и заливка ячеек таблиц

Таблица состоит из ячеек, которые объединены в строки и столбцы. Поэтому, как у строк и столбцов, так и у отдельных ячеек есть аналогичные между собой свойства, отличающиеся только областью применения этих свойств. Если мы, например, изменяем толщину границ для ячейки, то это изменение приведет к изменению отображения данной ячейки. Если изменить параметры строки (столбца), то это может привести к изменению всех границ всех ячеек строки (столбца) или только границы строки (столбца). Заливка определяется для всех ячеек выбранной строки (столбца). Рассмотрим это на следующем примере, в котором будем изменять толщину линий столбца, их цвет и заливку.

Изменение некоторых визуальных свойств столбца таблицы

```
var table:variant;
// Выбираем таблицу, для столбцов которой будем проводить изменения.
// В свойства компонента ColumnId записываем количество столбцов
// выбранной нами таблицы.
procedure TOKBottomDlg5.ListBox1Click(Sender: TObject);
begin
table:=Form1.W.ActiveDocument.Tables.item(ListBox1.itemindex+1);
NumColumns.Value:=table.Columns.Count;
ColumnId.MaxValue:=table.Columns.Count;
end;

// Толщина линий границ ячеек может принимать значения из набора
// определенных констант.
procedure TOKBottomDlg5.OutsideLineWidthChange(Sender: TObject);
const
   wdLineWidth025pt=2;
   wdLineWidth450pt=36;
begin
case OutsideLineWidth.ItemIndex of
0:table.Columns.Item(ColumnId.value).Borders.OutsideLineWidth:=
                                              wdLineWidth025pt;
1:table.Columns.Item(ColumnId.value).Borders.OutsideLineWidth:=
                                              wdLineWidth450pt;
end;
end;

// Цвет линий, рисунка текстуры и заливки определяется посредством
// цветового индекса.
procedure TOKBottomDlg5.OutsideColorIndexChange(Sender: TObject);
begin
table.Columns.Item(ColumnId.value).Borders.OutsideColorIndex:=
                                  OutsideColorIndex.ItemIndex;
end;

procedure TOKBottomDlg5.BackgroundPatternColorIndexChange(Sender:
                                                              TObject);
begin
table.Columns.Item(ColumnId.value).Shading.BackgroundPatternColorIndex:=
                                 BackgroundPatternColorIndex.ItemIndex;
end;
```

```
procedure TOKBottomDlg5.ForegroundPatternColorIndexChange(Sender: TObject);
begin
table.Columns.Item(ColumnId.value).Shading.ForegroundPatternColorIndex:=
                                    ForegroundPatternColorIndex.ItemIndex;
end;

// Текстура представляет собой определенный узор, заполняющий
// ячейки столбца. Его выбор определяется значением из набора
// определенных констант.

procedure TOKBottomDlg5.TextureChange(Sender: TObject);
const
   wdTextureNone=0;
   wdTextureSolid=1000;
   wdTexture30Percent=300;
begin
case Texture.ItemIndex of
0:table.Columns.Item(ColumnId.value).Shading.Texture:=wdTextureNone;
1:table.Columns.Item(ColumnId.value).Shading.Texture:=wdTextureSolid;
2:table.Columns.Item(ColumnId.value).Shading.Texture:=wdTexture30Percent;
end;
end;
```

Результаты применения описанных процедур к выбранной таблице и внешний вид формы представлены на рис. 7.5.

Рассмотрим еще одно важное свойство, применяемое как к отдельным ячейкам, так и ко всем ячейкам строки, столбца или таблицы. Это стиль линии границы. Особенностью его является то, что стиль линии задается не для всех линий границы ячейки, а индивидуально для каждой. Выбор определенной линии границы осуществляется выбором элемента коллекции Borders. Количество элементов коллекции определяется числом линий, ограничивающих ячейки или пересекающих ячейки или группу выделенных ячеек. Для простоты рассмотрим фрагмент программы, изменяющий стиль одной выбранной линии границы ячейки.

Задание стиля линии границы ячейки

```
procedure TOKBottomDlg8.LineStyleChange(Sender: TObject);
const
wdBorderTop=-1;
wdBorderLeft=-2;
wdBorderBottom=-3;
wdBorderRight=-4;
var Border:variant;
```

```
begin
case BorderId.itemindex of
0:Border:=table.Cell(2,1).Borders.Item(wdBorderTop);
1:Border:=table.Cell(2,1).Borders.Item(wdBorderLeft);
2:Border:=table.Cell(2,1).Borders.Item(wdBorderBottom);
3:Border:=table.Cell(2,1).Borders.Item(wdBorderRight);
end;
Border.LineStyle:=LineStyle.itemindex;
end;
```

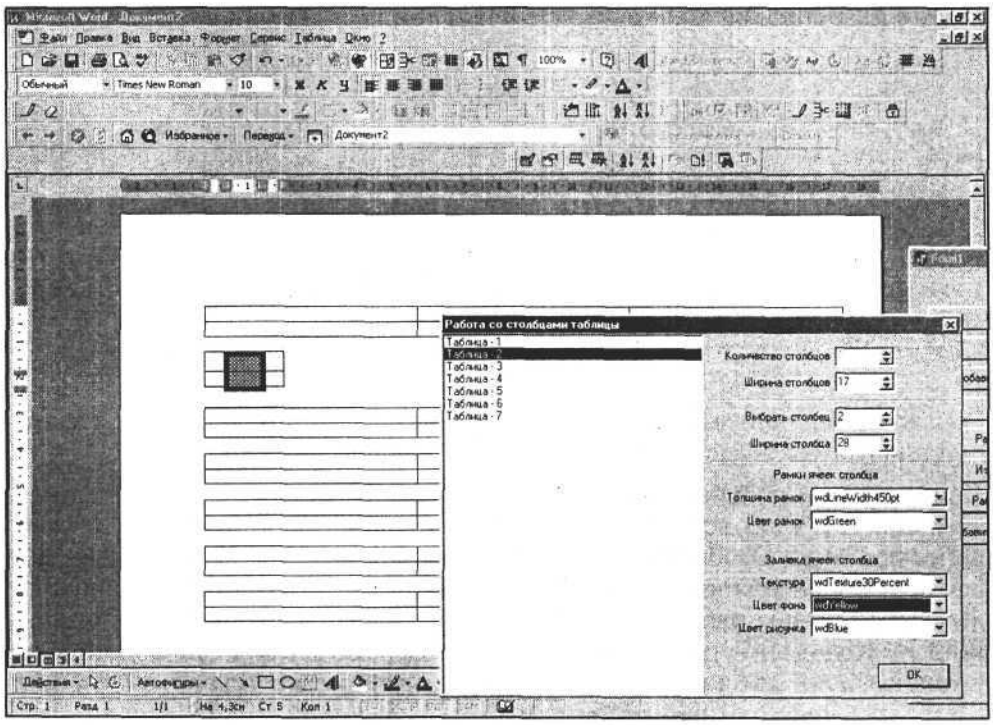


Рис. 7.5. Изменение формата столбца таблицы

С помощью этой процедуры мы можем изменить стиль разных линий границы и получить результат, представленный на рис. 7.6.

Добавление и удаление строк и столбцов таблицы

Рассмотрев общие свойства столбцов, строк и ячеек, перейдем к некоторым методам коллекций Rows и Columns.

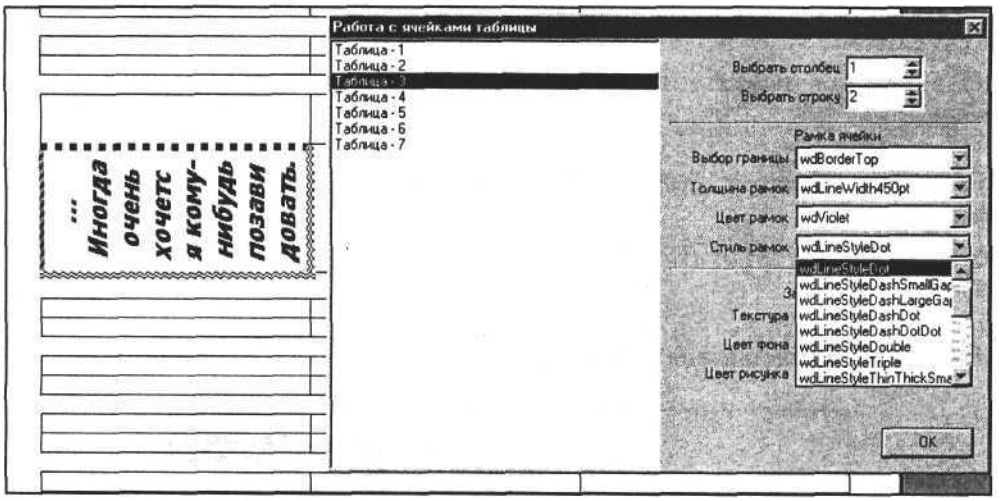


Рис. 7.6. Изменение стилей линий границ ячейки

Создавая таблицу, мы не можем быть уверенными в том, что нас устроит количество строк и столбцов, определенное при ее создании. Для увеличения таблицы предназначен метод Add коллекции Rows и Columns, а для удаления элементов этих коллекций — метод Delete. Первый метод добавляет строку или столбец в конец таблицы, а второй удаляет строку или столбец с указанным индексом.

Создадим новую форму и расположим в ней элементы управления. В процедуру нажатия кнопки Button1 добавим следующий программный текст.

Изменение количества строк и столбцов таблицы

```
var table:variant;
...
procedure TOKBottomDlg6.ListBox1Click(Sender: TObject);
begin
table:=Form1.W.ActiveDocument.Tables.item(ListBox1.itemindex+1);
end;

procedure TOKBottomDlg6.Button1Click(Sender: TObject);
begin
table.Columns.Add;
end;

procedure TOKBottomDlg6.Button2Click(Sender: TObject);
begin
table.Rows.Add;
end;
```

```
procedure TOKBottomDlg6.Button3Click(Sender: TObject);
begin
table.Columns.Item(1).Delete;
end;

procedure TOKBottomDlg6.Button4Click(Sender: TObject);
begin
table.Rows.Item(1).Delete;
end;
```

На рис. 7.7 представлена таблица до внесения изменений.

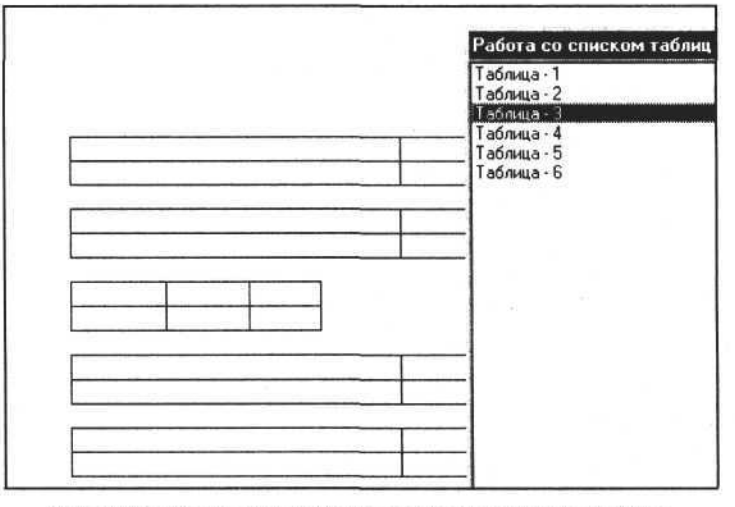


Рис. 7.7. Таблица до изменения количества строк и столбцов

На рис. 7.8 представлена таблица, в которую добавлено несколько столбцов и строк. Поскольку строки и столбцы добавляются в конец таблицы, они имеют такой же стиль, как и те, к которым они добавлены.

Для того чтобы добавить строку или столбец в произвольное место таблицы, нужно вызывать метод Add с аргументом, которым является ссылка на строку (столбец), перед которой требуется вставить новую строку (столбец). Процедура, вставляющая новую строку между первой и второй строкой с помощью метода Add, выглядит так:

Вставка новой строки между первой и второй строками таблицы

```
procedure TOKBottomDlg6.Button2Click(Sender: TObject);
 var row:variant;
```

```
begin
row:=table.Rows.Item(2);
table.Rows.Add(row);
end;
```

Можно переработать эту процедуру для вставки столбца в заданное место таблицы.

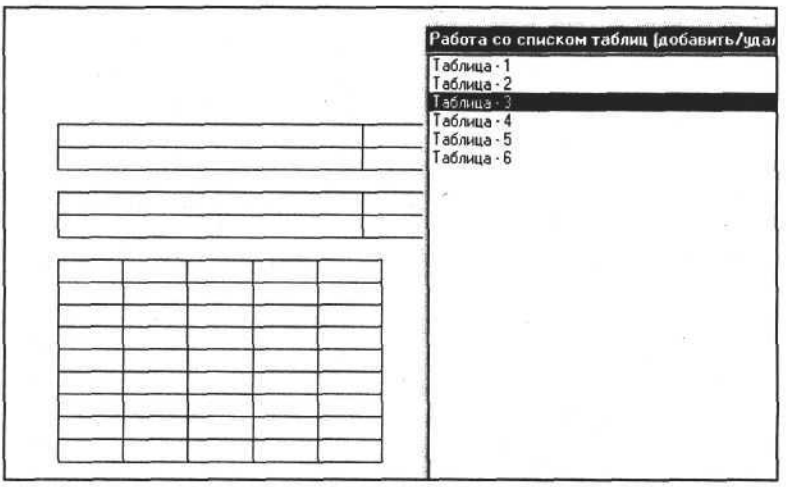


Рис. 7.8. Таблица после добавления строк и столбцов

Текст в ячейках таблицы

Когда таблица создана, можно заполнять ее текстом. Записывать информацию непосредственно в ячейку таблицы позволяет объект Cell, который и представляет собой ячейку. Ссылку на конкретную ячейку получают, обращаясь к объекту Cell с параметрами (номер строки и столбца). Например, если выполнить оператор mycell:=table.Cell(1, 2);, то переменная mycell:variant будет содержать ссылку на ячейку в первой строке и во втором столбце таблицы. В состав объекта "ячейка" входит объект Range. Нам известно, что Range может содержать текст. Используя свойства ячейки и объекта Range, запишем текст в конкретную ячейку и зададим для него шрифт и расположение.

Запись текста в ячейку таблицы

```
// Компоненты RowId и ColumnId позволяют пользователю выбрать ячейку,
// а Range_Text - ввести в нее текст.
//
```

```
procedure TOKBottomDlg7.Range_TextChange(Sender: TObject);
begin
table.Cell(RowId.value, ColumnId.value).Range.Text:=Range_Text.text;
end;

// Выравниваем текст в ячейке по вертикали, используя свойство
// VerticalAlignment, которое может принимать значения – 0, 1, 2.
procedure TOKBottomDlg7.VerticalAlignmentChange(Sender: TObject);
begin
table.Cell(RowId.value, ColumnId.value).VerticalAlignment:=
                                VerticalAlignment.itemindex;
end;

// Горизонтальное выравнивание осуществляется с помощью
// свойства Alignment объекта ParagraphFormat, принадлежащего
// объекту Range.
procedure TOKBottomDlg7.AlignmentChange(Sender: TObject);
begin
table.Cell(RowId.value, ColumnId.value).Range.ParagraphFormat.Alignment:=
                                    Alignment.itemindex;
end;
```

Задание шрифта текста в документе и в таблице

Шрифт в Word представляет собой объект, имеющий свойства и методы, часть которых аналогична свойствам и методам объекта Tfont в Delphi. Отличия заключаются в следующем: в Delphi можно задать цвет шрифта как сочетание красного, зеленого и синего, а в Word цвет выбирается из палитры возможных значений. В Word шрифт имеет больше дополнительных визуальных параметров, чем шрифт в Delphi. Сравним диалоги выбора и настройки шрифта в Word (рис. 7.9) и Delphi (рис. 7.10).

Мы будем задавать шрифт из программ, разработанных в Delphi, поэтому нам нужно будет преобразовать параметры объекта Tfont Delphi в параметры объекта Font Word. Поскольку аналогичные по назначению свойства этих двух объектов имеют разные типы, нам придется использовать следующую специальную процедуру.

Преобразование параметров объекта Tfont (Delphi) в параметры объекта Font (Word)

```
Function  FontToWFont(font:Tfont;WFont:variant):boolean;
begin
FontToWFont:=true;
```

```
try
WFont.Name:=font.Name;
if fsBold in font.Style then WFont.Bold:=True        // Полужирный
                        else WFont.Bold:=False;      // Светлый
if fsItalic in font.Style then WFont.Italic:=True    // Курсив
                          else WFont.Italic:=False; // Прямой
WFont.Size:=font.Size;                               // Размер
if fsStrikeOut in font.Style
               then WFont.StrikeThrough:=True        // Зачеркнутый
               else WFont.StrikeThrough:=False;      // Незачеркнутый
if fsUnderline in font.Style
           then  WFont.Underline:=wdUnderlineSingle // Подчеркивание
           else WFont.Underline:=wdUnderlineNone;   // Нет подчеркивания
except
FontToWFont:=false;
end;
end;
```

Далее используем эту процедуру для преобразования шрифта и рассмотрим фрагмент демонстрационной программы.

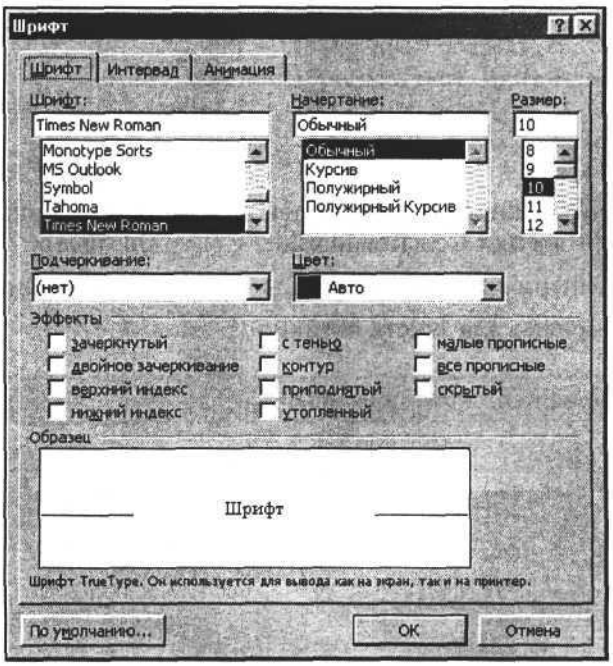


Рис. 7.9. Диалог выбора шрифта в Word

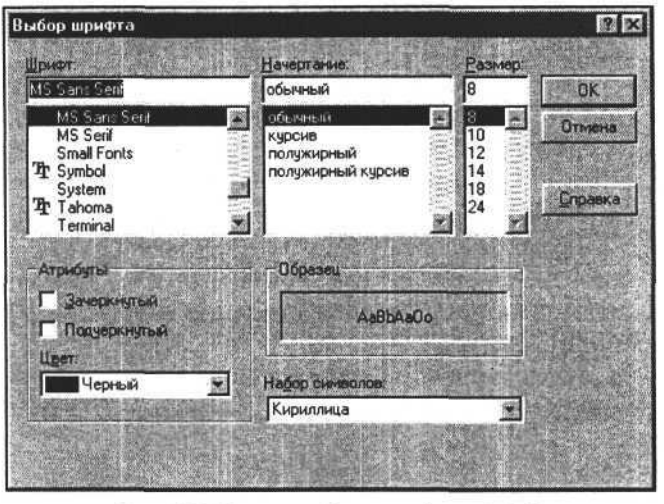


Рис. 7.10. Диалог выбора шрифта в Delphi

Выбор шрифта для текста ячейки

```
procedure TOKBottomDlg7.Button1Click(Sender: TObject);
begin
if not FontDialog1.Execute then exit;
FontToWFont(FontDialog1.font,table.Cell(RowId.value,
                                        ColumnId.value).Range.font);
end;
```

Данная процедура, используя возвращаемый диалогом FontDialog1 шрифт, изменяет шрифт ячейки Cell(RowId.value, ColumnId.value).

Результат выполнения данной процедуры представлен на рис. 7.11.

Зададим цвет шрифта для текста ячейки

Выбор цвета шрифта для текста ячейки

```
const
wdAuto              =0;
wdBlack             =1;
wdBlue              =2;
wdTurquoise         =3;
wdBrightGreen       =4;
...
wdGray50            =14;
wdGray25            =15;
```

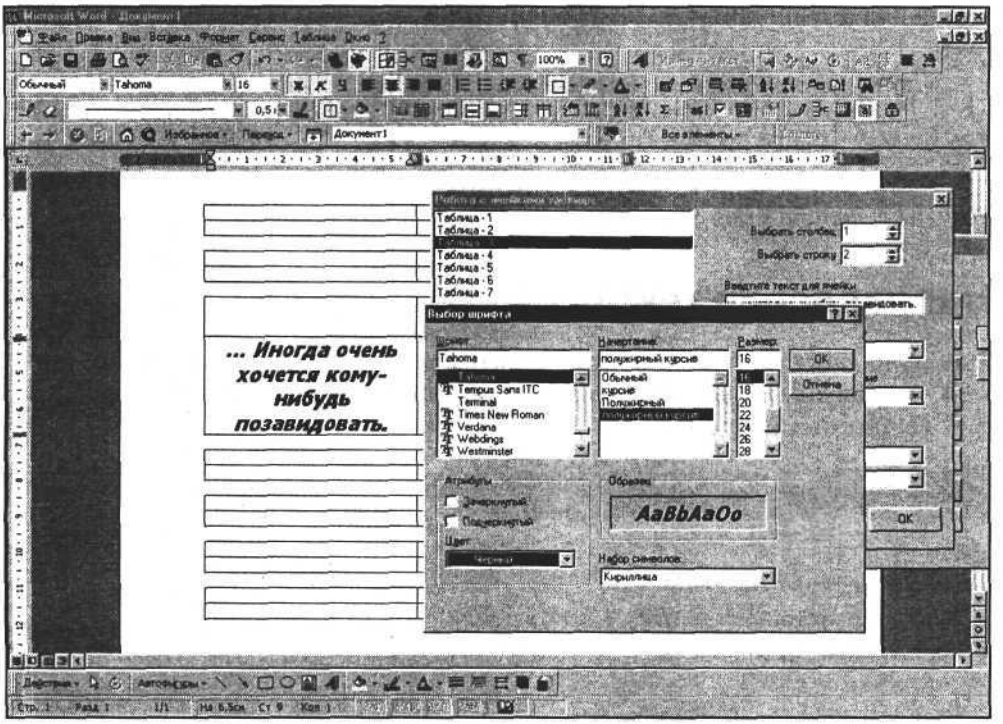


Рис. 7.11. Устанавливаем шрифт для текста ячейки таблицы

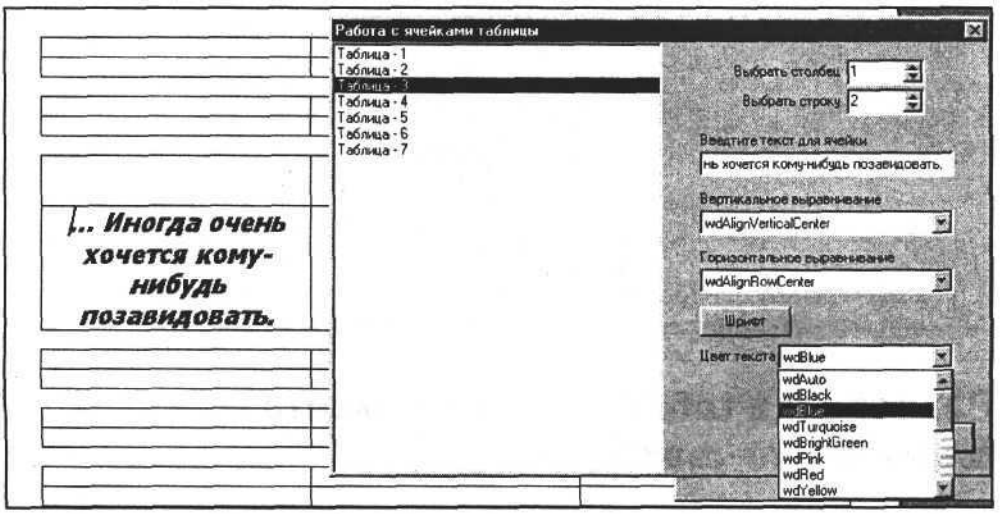


Рис. 7.12. Устанавливаем цвет шрифта для текста ячейки

```
procedure TOKBottomDlg7.TextColorIndexChange(Sender: TObject);
begin
table.Cell(RowId.value, ColumnId.value).Range.font.ColorIndex:=
                                        TextColorIndex.itemindex;
end;
```

Результат выполнения процедуры представлен на рис. 7.12.

Направление текста

Еще одно свойство текста в Word — направление. Направление текста задается установкой свойства Orientation объекта Range в значение одной из трех возможных констант.

Задание направления текста

```
const
wdTextOrientationUpward          =2;
wdTextOrientationHorizontal      =0;
wdTextOrientationDownward        =3;

procedure TOKBottomDlg7.OrientationChange(Sender: TObject);
begin
case Orientation.itemindex of
0:table.Cell(RowId.value,ColumnId.value).Range.Orientation:=
                                    wdTextOrientationUpward;
1:table.Cell(RowId.value,ColumnId.value).Range.Orientation:=
                                  wdTextOrientationHorizontal;
2:table.Cell(RowId.value, ColumnId.value).Range.Orientation:=
                                    wdTextOrientationDownward;
end;
end;
```

Используем данную процедуру и установим направление текста в ячейке. Результат выполнения данной процедуры представлен на рис. 7.13.

Разработка табличного документа — бланк счета-фактуры

Печатная форма счета-фактуры состоит из трех частей. Первая часть представляет собой заголовок формы, количество записей и размеры которой постоянны. Вторая часть представляет собой таблицу, состоящую из заго-

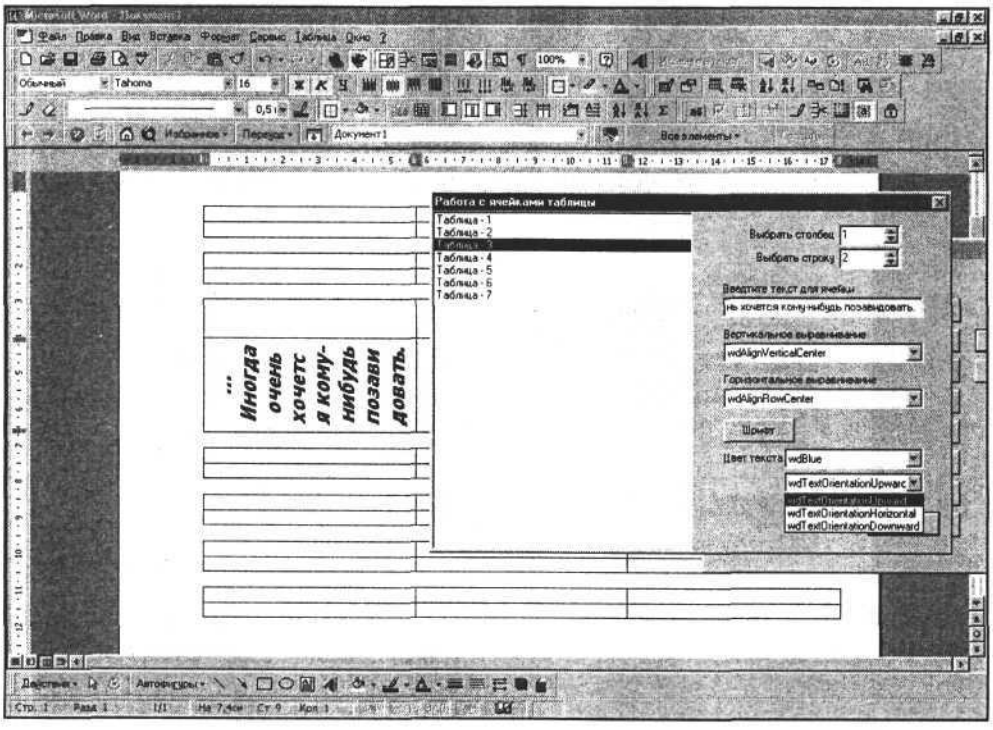


Рис. 7.13. Устанавливаем направление шрифта для текста ячейки

ловка, табличной части и оконечной части, которая содержит итоговые суммы. Третья часть формы счета-фактуры представляет собой записи, количество которых является константой, а расположение определяется размерами табличной части документа.

Для формирования документа будем использовать его шаблон, который создадим и сохраним в формате шаблона документа Word. Данный шаблон, как и шаблон простого (без таблиц) документа, должен содержать постоянный текст, который не будет меняться при формировании новых документов, и текстовые константы, служащие для подстановки реального текста при формировании документа. Кроме этого в шаблоне счета-фактуры должна присутствовать табличная часть, ячейки которой содержат как постоянные записи, так и текстовые константы для заполнения из программы. Дополнительно табличная часть должна иметь строки, по подобию которых будут добавляться новые строки в таблицу.

Внешний вид возможного шаблона документа бланка счета-фактуры представлен на рис. 7.14.

Для формирования выходного документа будем использовать методы поиска фрагмента текста в документе, поиска и замены фрагмента текста на задан-

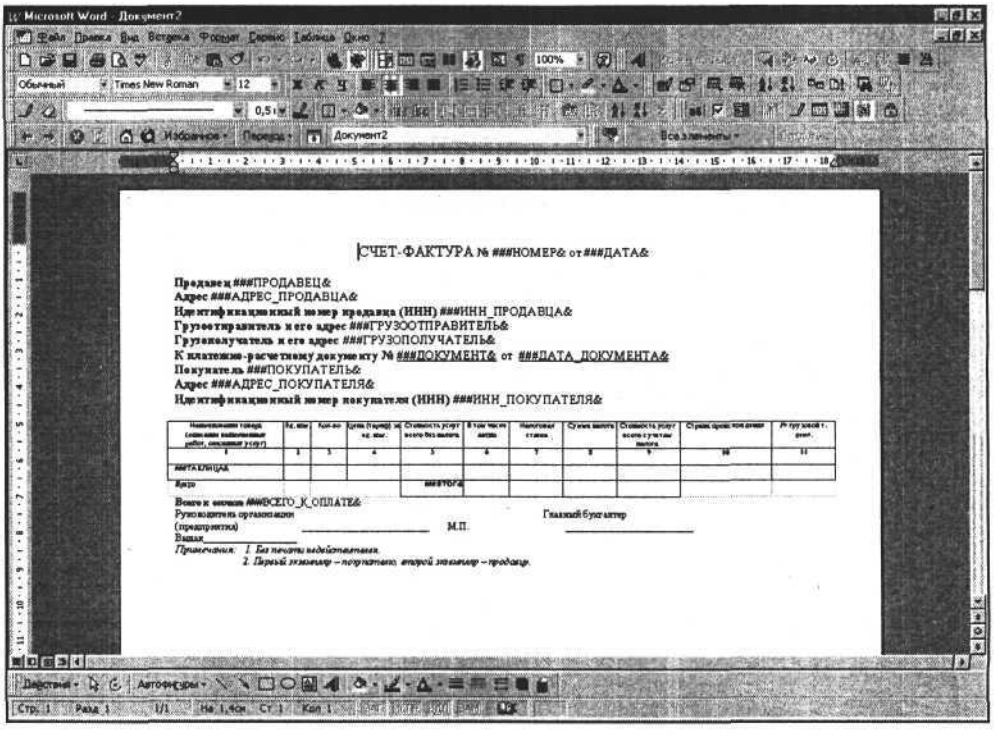


Рис. 7.14. Шаблон документа бланка счета-фактуры

ный текст. Для работы с табличной частью документа нам придется использовать свойства и методы для обеспечения перехода к заданной ячейке, определения текущей таблицы и положения в ней ячейки, записи текста в ячейку. Рассмотрим фрагменты исходного текста:

Формирование табличного документа с использованием шаблона документа

```
// Функция FindAndInsert находит текстовую константу и подставляет
// на ее место текст
function FindAndInsert(FindText,ReplacementText:string):boolean;
 const wdReplaceAll=2;
begin
Form1.W.Selection.Find.Text:=FindText;
Form1.W.Selection.Find.Replacement.Text:=ReplacementText;
FindAndInsert:=Form1.W.Selection.Find.Execute(Replace:=wdReplaceAll);
end;

// Функция FindRowColumnInTable находит текстовую константу и возвращает
// таблицу, а также номер строки и столбца, где находится эта константа
```

```
Function FindRowColumnInTable(FindText:string;var tab:variant; var
                                    Row,Column:integer):boolean;
begin
FindRowColumnInTable:=false;
try
Form1.W.Selection.Find.Text:=FindText;
if Form1.W.Selection.Find.Execute then begin
   Column:=Form1.W.Selection.Cells.Item(1).ColumnIndex;
   Row:=Form1.W.Selection.Cells.Item(1).RowIndex;
   tab:=Form1.W.Selection.Tables.Item(1);
   FindRowColumnInTable:=true;
                                        end;
except
FindRowColumnInTable:=false;
end;
end;

// Данная процедура формирует документ счет-фактуру
procedure TForm1.Button11Click(Sender: TObject);
 var table_:variant;
 Row_,Column_:integer;
begin
// Создаем новый документ по шаблону
W.documents.Add(ExtractFileDir(Application.ExeName)+
  '\Шаблон счета-фактуры.dot');

// Подставляем текст в заголовок документа
FindAndInsert('###НОМЕР&','1');
FindAndInsert('###Дата&',datetostr(date));
FindAndInsert('###ПРОДАВЕЦ&','ЗАО Сокол');
FindAndInsert('###АДРЕС_ПРОДАВЦА&',
              'г. Санкт-Петербург, ул. Кузнечная 5');
FindAndInsert('###ИНН_ПРОДАВЦА&','1234567890');
FindAndInsert('###ГРУЗООТПРАВИТЕЛЬ&','ЗАО Сокол');
FindAndInsert('###ГРУЗОПОЛУЧАТЕЛЬ&','ЗАО Селена');
FindAndInsert('###ДОКУМЕНТ&','1');
FindAndInsert('###ДАТА_ДОКУМЕНТА&',datetostr(date));
FindAndInsert('###ПОКУПАТЕЛЬ&','ЗАО Селена');
FindAndInsert('###АДРЕС_ПОКУПАТЕЛЯ&','г. Москва, ул. Комаровского 41');
FindAndInsert('###ИНН_ПОКУПАТЕЛЯ&','0987654321');

// Если нашли текстовую константу, то переходим к формированию табличной
// части.
if FindRowColumnInTable('###ТАБЛИЦА&',table_,Row_,Column_)then begin
```

```
// Перед началом формирования курсор находится в таблице table_
// в ячейке Cell(Row_,Column_). Добавляем в таблицу необходимое
// количество строк, уменьшенное на 1 (поскольку одна строка
// в шаблоне уже есть).
W.Selection.InsertRows(2-1);

// Формируем первую строку табличной части документа.
table_.Cell(Row_,Column_+0).Range.Text:='KOVRIC TRIVEL RUG';
table_.Cell(Row_,Column_+1).Range.Text:='шт';
table_.Cell(Row_,Column_+2).Range.Text:='1';
table_.Cell(Row_,Column_+3).Range.Text:='342,00';
table_.Cell(Row_,Column_+4).Range.Text:='342,00';
table_.Cell(Row_,Column_+6).Range.Text:='20,00';
table_.Cell(Row_,Column_+7).Range.Text:='68,40';
table_.Cell(Row_,Column_+8).Range.Text:='410,40';


// Формируем вторую строку табличной части документа.
Inc(Row_);
table_.Cell(Row_,Column_+0).Range.Text:=
                            'НИТЬ Climax Braided 100m 0,16mm';
table_.Cell(Row_,Column_+1).Range.Text:='шт';
table_.Cell(Row_,Column_+2).Range.Text:='2';
table_.Cell(Row_,Column_+3).Range.Text:='309,08';
table_.Cell(Row_,Column_+4).Range.Text:='618,16';
table_.Cell(Row_,Column_+6).Range.Text:='20,00';
table_.Cell(Row_,Column_+7).Range.Text:='123,63';
table_.Cell(Row_,Column_+8).Range.Text:='741,79';

// Формируем подпись табличной части документа.
if FindRowColumnInTable('###ИТОГ&',table_,Row_,Column_)then begin
   table_.Cell(Row_,Column_+0).Range.Text:='960,16';
   table_.Cell(Row_,Column_+1).Range.Text:='192,03';
   table_.Cell(Row_,Column_+2).Range.Text:='1152,19';
   end;
end; // Заполняем текст в окончании документа.
FindAndInsert('###ВСЕГО_К_ОПЛАТЕ&',
              'Одна тысяча сто пятьдесят два рубля 19 копеек');
end;
```

Результат выполнения процедуры представлен на рис. 7.15.

Если доработать представленный фрагмент программы так, чтобы вносить данные из таблицы базы данных, то разработчики Delphi смогут с успехом

использовать его в своих проектах не только для формирования счетов-фактур.

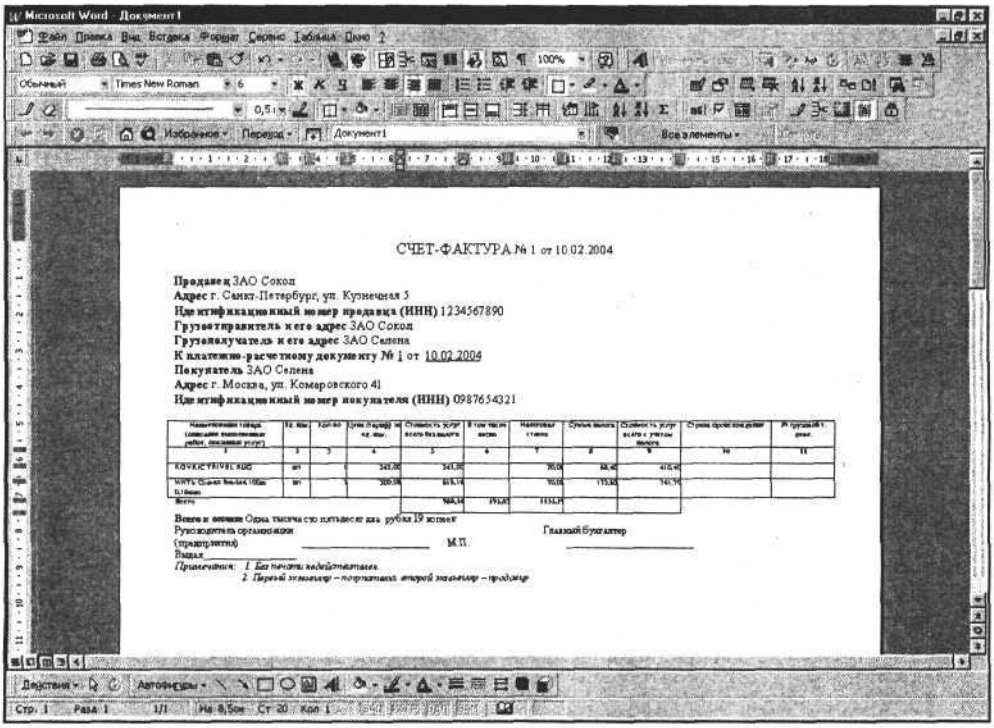


Рис. 7.15. Сформированный бланк счета-фактуры

Как известно, документы Word наряду с текстом и таблицами могут содержать рисунки, надписи, линии, геометрические фигуры и другие графические объекты. Возможно, вы захотите украсить документ логотипом своей фирмы. О том, как использовать эти возможности Word в программах Delphi, читайте в следующей главе. Там же вы найдете информацию о том, как подготовить документ для печати и выполнить печать.

ГЛАВА 8

Работа с объектами в документе Word

В этой главе рассматриваются следующие темы:

- коллекция объектов Shapes;
- надписи;
- выноски;
- линии;
- геометрические фигуры;
- внешние объекты (OLE);
- настройка страницы;
- печать документа;
- пример программы — формирование товарного ярлыка.

Коллекция объектов Shapes

Рассмотрим объекты документа Word, расположение которых, в отличие от таблиц и текста, можно задавать произвольно с помощью координат. Такими объектами являются надписи, простые геометрические фигуры (линии, прямоугольники и овалы), автофигуры (предопределенные геометрические фигуры), рисунки, внешние объекты (OLE-объекты) и др. Все эти разнородные объекты, находящиеся в документе, объединены в коллекцию Shapes. Рассмотрим некоторые общие свойства и методы коллекции Shapes. Свойства: Count — содержит количество элементов коллекции, SelectAll — выделяет все элементы коллекции. Методы AddCallout, AddCurve, AddLabel, AddLine, AddOLEControl, AddOLEObject, AddPicture, AddPolyline, AddShape, AddTextbox, AddTextEffect добавляют в коллекцию различные объекты, рисунки, указатели, элементы управления и т. д.

Надписи

Рассмотрим некоторые свойства коллекции Shapes и ее элементов на примере надписи (объекта TextBox). Для этого создадим в документе такой объект. Рассмотрим следующий фрагмент программы:

Создание надписи

```
procedure TForm1.Button2Click(Sender:TObject);
const msoTextOrientationHorizontal=1;
 var left_, top_:Extended;
begin
  left_:=10.15;
  top_:=10.15;
  W.ActiveDocument.Shapes.AddTextbox(msoTextOrientationHorizontal,
                left_, top_, 200.25, 100, W.ActiveDocument.Range);
end;
```

Примечание

Для задания координат надписи можно использовать переменные типа real, но для Office XP необходимо использовать только тип Extended (var left_, top_:Extended).

Для создания нового объекта мы используем метод AddTextbox коллекции Shapes. Первый аргумент этого метода имеет тип integer и задает ориентацию расположения текста этого объекта, последний аргумент имеет тип Range и определяет область, где будет создан объект, — в нашем случае это активный документ. Остальные аргументы типа Extended определяют координаты и размеры объекта. Для создания надписи с вертикальным размещением текста необходимо в качестве первого аргумента использовать константу msoTextOrientationVertical=5. С помощью константы msoTextOrientationDownward=3 или msoTextOrientationUpward=2 можно задать направление текста при вертикальной ориентации (сверху вниз или снизу вверх).

Если описанную процедуру видоизменить так, чтобы при создании нового объекта учитывались координаты и размеры последнего, а также размеры страницы (см. приложение на сопроводительном диске книги), то ее выполнение может привести к результату, представленному на рис. 8.1.

Рассмотрим методы элементов коллекции на примере коллекции объектов TextBox, которую мы до этого создали. Для доступа к элементам коллекции используем свойство Item(i), которое возвращает объект, где i — индекс или имя объекта. Например:

```
WordTextBox:=W.ActiveDocument.Shapes.Item(1);
```

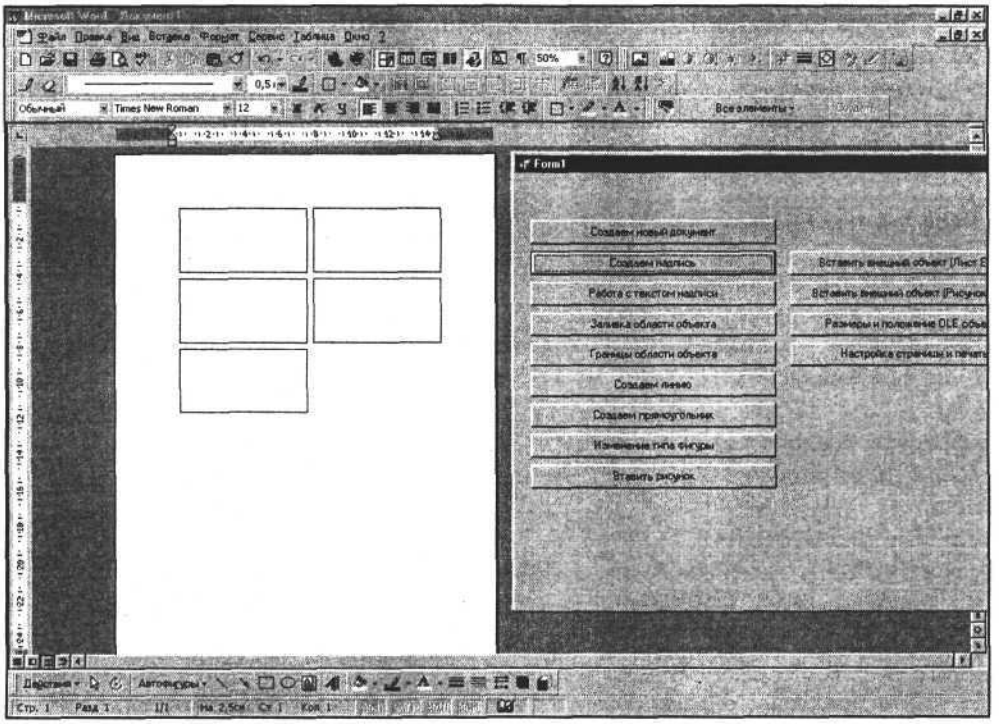


Рис. 8.1. Создание надписей

Данный оператор запишет в переменную WordTextBox:variant ссылку на первый объект коллекции.

Общими свойствами и методами для объектов коллекции Shapes являются размеры, координаты размещения объектов и некоторые методы, например Delete и Select. Следующие операторы соответственно выполняют выделение объекта, выбранного с помощью предыдущего оператора, изменение его координаты и удаление:

```
WordTextBox.Select;
WordTextBox.Left:=10;
WordTextBox.Delete;
```

Поскольку коллекция Shapes может содержать разнородные объекты, эти объекты могут обладать и особыми свойствами. Объекты TextBox, созданные методом AddTextBox и рассмотренные в предыдущем примере, предназначены для отображения текста, поэтому для них специфичны свойства и методы для работы с текстом. Рассмотрим их.

Основным объектом, принадлежащим объекту TextBox и содержащим текст и его характеристики, является объект TextFrame. Он определяет границы

области текста внутри рамки надписи, направление текста и некоторые другие характеристики. В свою очередь объект TextFrame включает в себя объект Range, который содержит текст и характеристики шрифта. Рассмотрим следующую форму и исходный текст Delphi для манипуляций с текстом объекта TextBox.

Работа с текстом надписи

```
var WordTextBox:variant;
//
// Загружаем в ListBox1 имена объектов коллекции Shapes
// из документа Word.
//
procedure TOKBottomDlg2.FormCreate(Sender:TObject);
 var a_:integer;
begin
ListBox1.Items.Clear;
for a_:=1 to Form1.W.ActiveDocument.Shapes.Count do
    ListBox1.Items.Add(Form1.W.ActiveDocument.Shapes.Item(a_).Name);
end;
//
// При активизации строки объекта ListBox1, используя имя объекта,
// выделяем его из коллекции Shapes и записываем ссылку на этот объект
// в переменную WordTextBox. Затем в компонент Text записываем текстовое
// содержимое этого объекта.
//

procedure TOKBottomDlg2.ListBox1Click(Sender: TObject);
begin
Form1.W.ActiveDocument.Shapes.Item(
                ListBox1.Items.Strings[ListBox1.ItemIndex]).Select;
WordTextBox:=Form1.W.ActiveDocument.Shapes.Item(
                ListBox1.Items.Strings[ListBox1.ItemIndex]);
Text.Text:=WordTextBox.TextFrame.TextRange.Text;
end;
// При активизации компонента Text типа TEdit записываем введенный
// в него текст непосредственно в объект TextBox документа Word.
//

procedure TOKBottomDlg2.TextChange(Sender:TObject);
begin
WordTextBox.TextFrame.TextRange.Text:=Text.Text;
end;
```

```
//
// Следующие две процедуры регулируют величину отступа левой границы
// текста от левой границы надписи.
//
procedure TOKBottomDlg2.SpinButton1DownClick(Sender:TObject);
begin
WordTextBox.TextFrame.MarginLeft:=WordTextBox.TextFrame.MarginLeft-0.25;
end;

procedure TOKBottomDlg2.SpinButton1UpClick(Sender: TObject);
begin
WordTextBox.TextFrame.MarginLeft:=WordTextBox.TextFrame.MarginLeft+0.25;
end;
```

Результат выполнения, описанного выше исходного текста, показан на рис. 8.2.

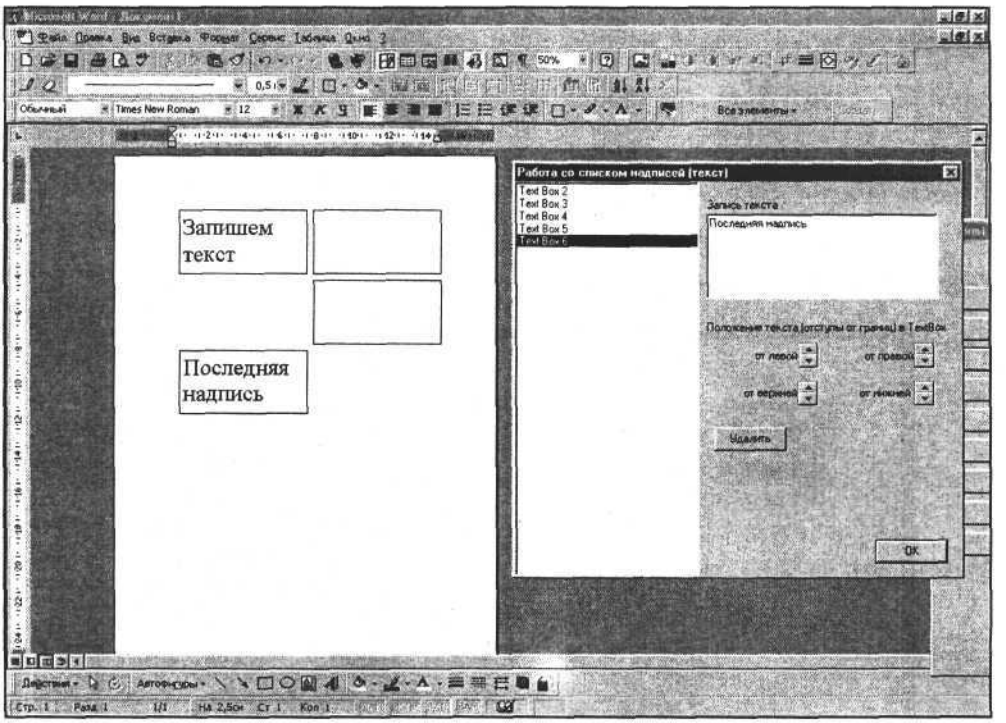


Рис. 8.2. Работа с текстом надписи

Надпись, наряду с текстом и его свойствами, имеет параметры, определяющие другие ее визуальные элементы — заливку (фоновый цвет) и линию

границы. Заливка определяется цветом, текстурой, рисунком или узором. Линия границы определяется толщиной, цветом, узором и стилем. Рассмотрим формирование заливки надписи.

Заливка надписи

Заливка надписи полностью определяется объектом Fill и его свойствами. Этот объект принадлежит родительскому объекту TextBox, а доступ к нему осуществляется следующим образом: WordTextBox.Fill, где WordTextBox — ссылка на объект TextBox. В самом простом случае в качестве заливки можно использовать цветовой фон, который задается комбинацией из трех констант, соответствующих цветам модели RGB (Red — красный, Green — зеленый, Blue — синий), например:

```
ForeColor.RGB:=RGB(100,150,220);
```

С помощью следующей процедуры можно в диалоговом режиме изменять заливку области нашей надписи.

Изменение заливки надписи

```
procedure TOKBottomDlg3.Button1Click(Sender:TObject);
begin
  if not ColorDialog1.Execute then exit;
  WordTextBox.Fill.ForeColor.RGB:=ColorDialog1.Color;
end;
```

Результат выполнения данной процедуры представлен на рис. 8.3.

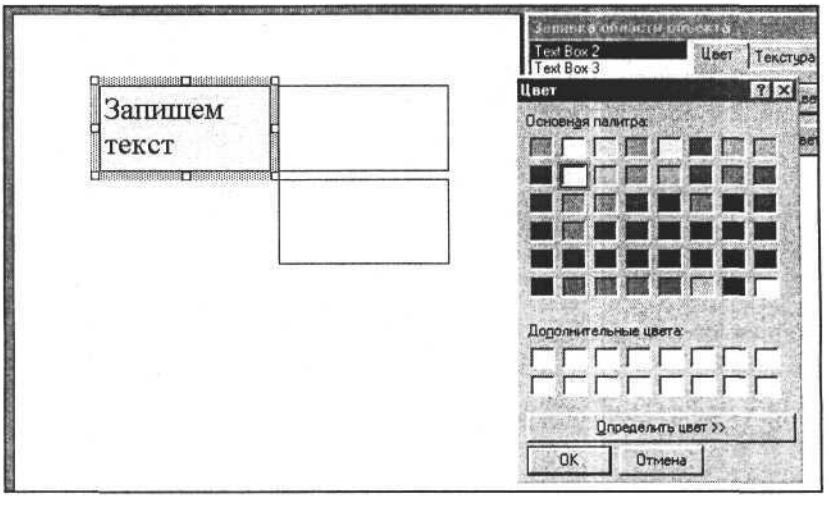


Рис. 8.3. Изменение заливки надписи

Если для заливки использовать два цвета, то такой способ заливки называется *градиентным* и определяется, в дополнение к обычному способу одноцветной заливки, еще тремя свойствами — значением второго цвета в формате RGB (свойство BackColor.RGB), способом задания и направлением градиента (метод TwoColorGradient). Рассмотрим следующий пример:

Использование градиентной заливки для надписи

```
procedure TOKBottomDlg3.Button2Click(Sender:TObject);
begin
  if not ColorDialog1.Execute then exit;
  WordTextBox.Fill.BackColor.RGB:=ColorDialog1.Color;
end;

procedure TOKBottomDlg3.CheckBox1Click(Sender:TObject);
begin
  if CheckBox1.Checked then
    WordTextBox.Fill.TwoColorGradient(Gradient.ItemIndex+1,
                                      napravlenie.ItemIndex+1)
  else WordTextBox.Fill.Solid;
end;
```

Первая процедура задает второй цвет, а вторая — стиль и направление градиента. Внешний вид формы приложения и результат представлены на рис. 8.4.

Для градиентной заливки более сложной формы можно выбрать предопределенный стиль градиентной заливки из предоставленного набора. Вид такой заливки определяется константой, задаваемой в качестве третьего аргумента метода Fill.PresetGradient (первый и второй аргументы — стиль и направление градиента). Это можно сделать с помощью следующей процедуры:

Использование предопределенного градиента для заливки надписи

```
procedure TOKBottomDlg3.PresetGradientChange(Sender: TObject);
begin
  WordTextBox.Fill.PresetGradient(Gradient.ItemIndex+1,
                napravlenie.ItemIndex+1, PresetGradient.ItemIndex+1);
end;
```

Результат выполнения процедуры представлен на рис. 8.5.

Еще один способ заливки надписи — задание текстуры. Текстура представляет собой многократно повторяемый рисунок, заполняющий прямоугольную область надписи. Рисунок текстуры определяется выбором в списке определенных числовых целочисленных констант и задается вызовом метода:

```
Fill.PresetTextured(textured:integer);
```

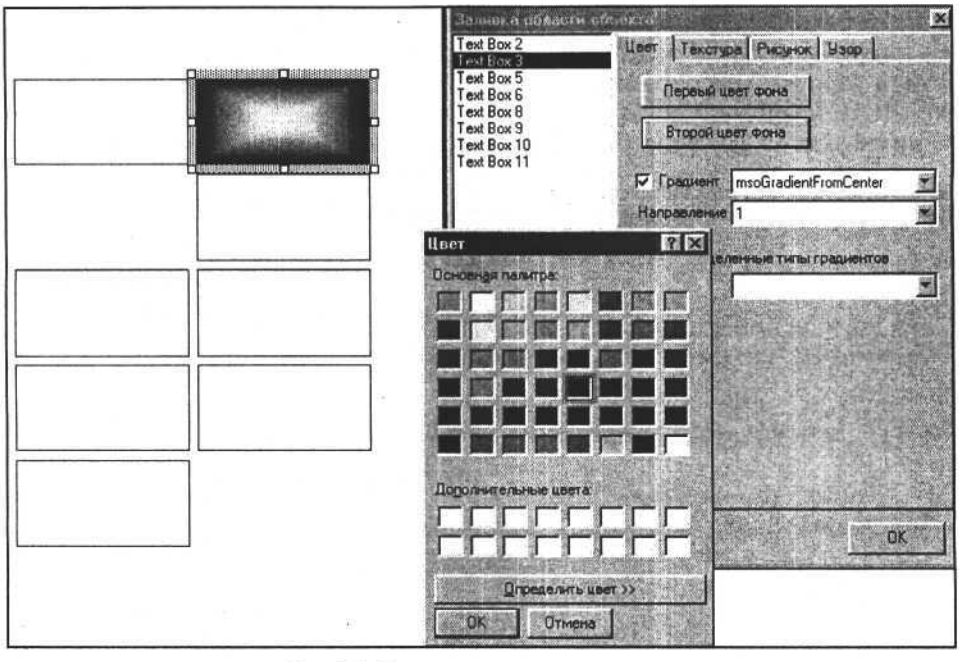


Рис. 8.4. Градиентная заливка надписи

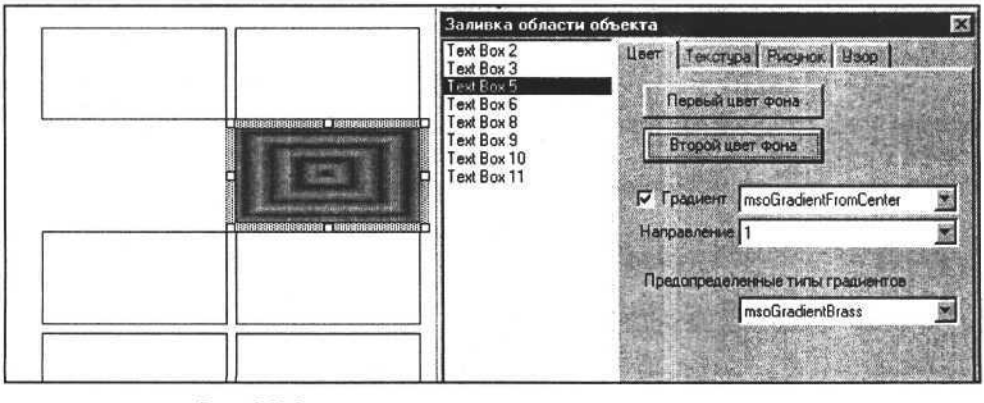


Рис. 8.5. Заливка надписи предопределенным градиентом

Использование предопределенной текстуры для заливки надписи

```
procedure TOKBottomDlg3.TextureChange(Sender:TObject);
begin
  WordTextBox.Fill.PresetTextured(Texture.ItemIndex+1);
end;
```

Создадим форму Delphi с элементами управления и разместим в ее модуле представленную процедуру. Внешний вид формы и результат работы программы показаны на рис. 8.6.

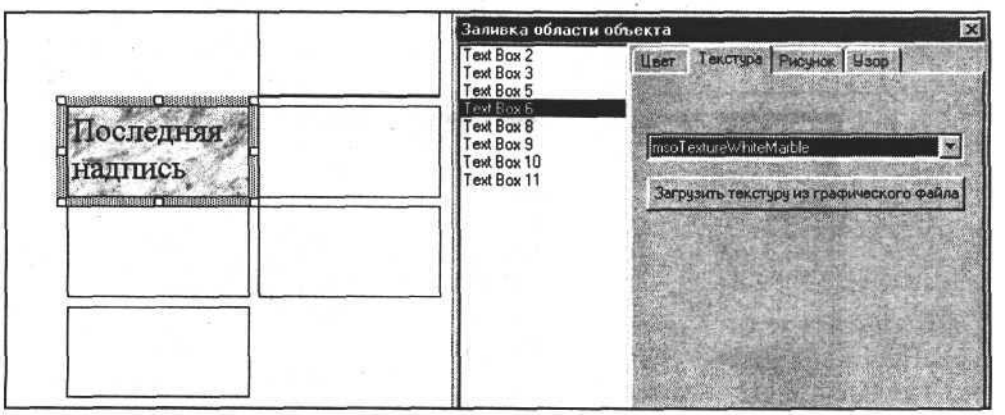


Рис. 8.6. Текстурная заливка надписи

В качестве текстуры можно также использовать рисунок, загружаемый из файла. Для этого применим метод UserTextured объекта Fill. Единственный аргумент этого метода — строка — ссылка на путь и имя графического файла.

Использование рисунка в качестве текстуры для надписи

```
procedure TOKBottomDlg3.Button3Click(Sender:TObject);
begin
  WordTextBox.Fill.UserTextured('c:\рыбак.bmp');
end;
```

Результат выполнения процедуры представлен на рис. 8.7.

Рисунок, загружаемый из файла, можно использовать не только в качестве текстуры, но и как фоновый рисунок (заливку), который сжимается или растягивается, вписываясь в размеры области объекта TextBox. Метод UserPicture, аргументом которого является строка — путь и имя графического файла, позволяет использовать рисунок в качестве фона объекта TextBox. Рассмотрим следующую процедуру.

Использование рисунка в качестве фона для надписи

```
procedure TOKBottomDlg3.Button4Click(Sender: TObject);
 var dir_:string;
```

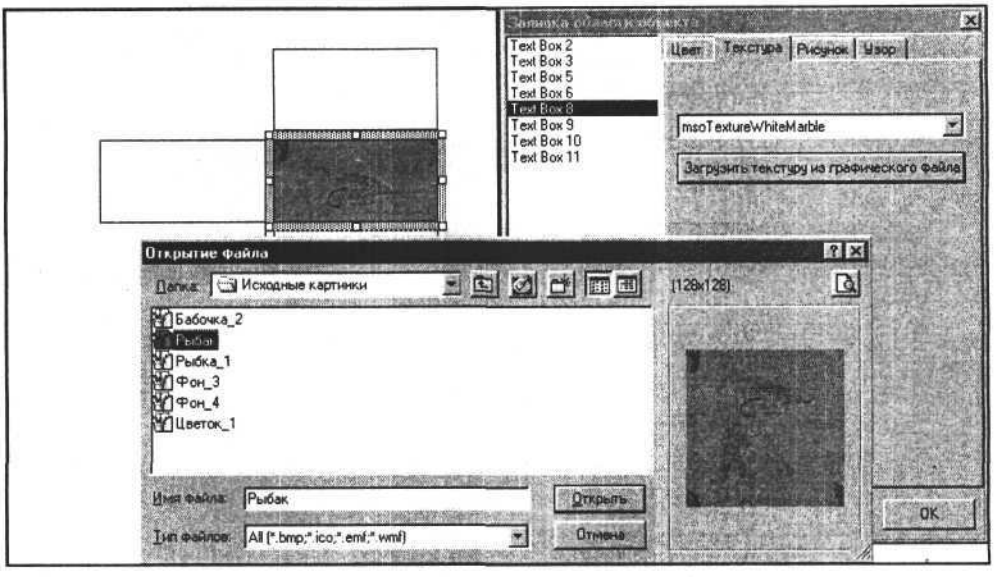


Рис. 8.7. Использование рисунка в качестве текстуры для надписи

```
begin
  GetDir(0,dir_);
  if not OpenPictureDialog1.Execute then begin
    ChDir(dir_);
    exit;
  end;
  ChDir(dir_);
  WordTextBox.Fill.UserPicture(OpenPictureDialog1.FileName);
end;
```

Результат выполнения представлен на рис. 8.8.

Еще одним и последним способом задания фона текстовой области надписи является использование узора. Узор определяется собственно рисунком (штриховкой), а также цветом фона и цветом штриховки. Узор определяется константой, выбираемой более чем из сотни возможных. Для выбора узора используется метод Patterned(Pattern:integer) объекта Fill, где Pattern определяет используемую штриховку. Для задания цветов фона и штриховки в формате RGB предназначены свойства BackColor.RGB и ForeColor.RGB. Рассмотрим следующий фрагмент исходного текста.

Использование штриховки в качестве фона надписи

```
procedure TOKBottomDlg3.Button1Click(Sender: TObject);
begin
  if not ColorDialog1.Execute then exit;
```

```
  WordTextBox.Fill.ForeColor.RGB:=ColorDialog1.Color;
end;

procedure TOKBottomDlg3.Button2Click(Sender: TObject);
begin
  if not ColorDialog1.Execute then exit;
  WordTextBox.Fill.BackColor.RGB:=ColorDialog1.Color;
end;

procedure TOKBottomDlg3.PatternChange(Sender: TObject);
begin
  WordTextBox.Fill.Patterned(Pattern.ItemIndex+1);
end;
```

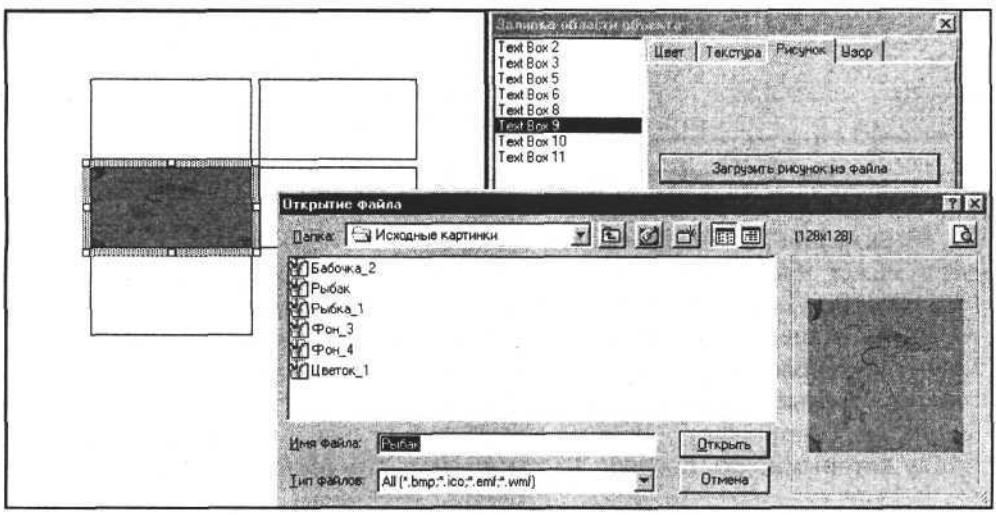


Рис. 8.8. Пример использования рисунка в качестве фона для надписи

Первая процедура определяет цвет штриховки, вторая — цвет фона. Цвета выбираются с помощью диалога ColorDialog1, возвращающего значение цвета как комбинацию значений в формате RGB. Третья процедура определяет вид штриховки с помощью элемента управления Pattern:TComboBox (комбинированный список). Внешний вид формы с элементами управления и результат выполнения программы представлены на рис. 8.9.

Линия границы надписи

Линии границы объекта коллекции Shapes в общем и надписи (объекта TextBox) в частности могут выглядеть по-разному и определяются свойства-

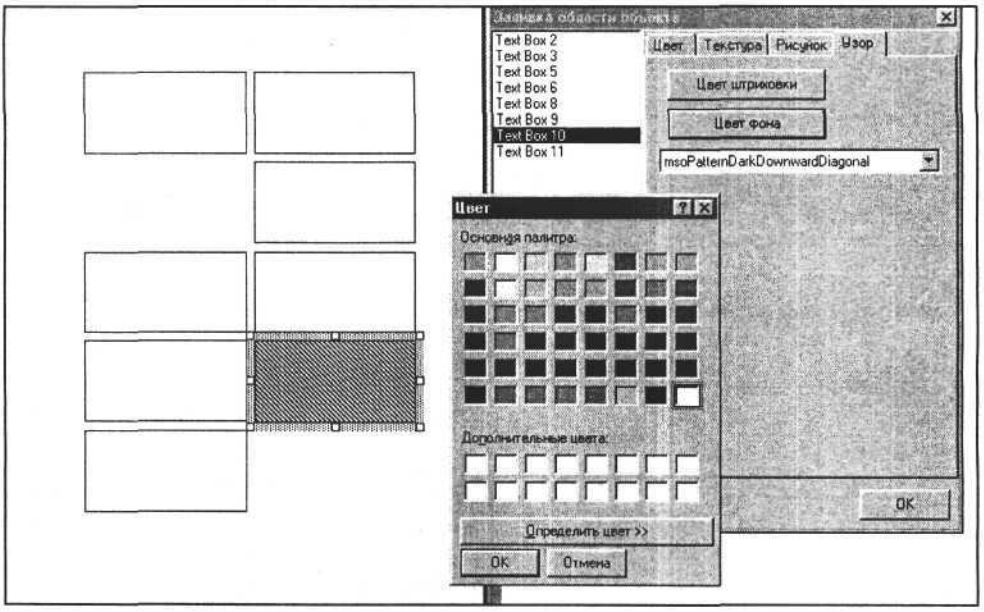


Рис. 8.9. Пример использования штриховки в качестве фона надписи

ми объекта Line, принадлежащего элементу коллекции Shapes. Граница объекта может быть невидимой, если значение свойства Visible равно 0 (WordTextBox.Line.Visible:=0). Если свойству Visible присвоить значение −1, то линия станет видимой. Наиболее часто используются свойства объекта Line, определяющие толщину и цвет линий. Если линия границы имеет толщину больше одного пиксела, то можно задать прозрачность цвета (свойство Transparency). В этом случае линия будет выглядеть как точки, между которыми есть просвет. Рассмотрим следующий исходный текст.

Задание видимости, толщины и цвета линии границы надписи

```
procedure TOKBottomDlg4.CheckBox2Click(Sender: TObject);
begin
  if CheckBox2.Checked then WordTextBox.Line.Visible:=-1
  else WordTextBox.Line.Visible:=0;
end;

procedure TOKBottomDlg4.Button1Click(Sender: TObject);
begin
  if not ColorDialog1.Execute then exit;
  WordTextBox.Line.ForeColor.RGB:=ColorDialog1.Color;
end;
```

```
procedure TOKBottomDlg4.SpinButton1DownClick(Sender: TObject);
begin
  WordTextBox.Line.Weight:=WordTextBox.Line.Weight-0.25;
end;

procedure TOKBottomDlg4.SpinButton2DownClick(Sender: TObject);
begin
  if WordTextBox.Line.Transparency<1 then
    WordTextBox.Line.Transparency:=WordTextBox.Line.Transparency+0.01;
end;
```

Первая процедура задает видимость границы объекта, вторая определяет ее цвет, присваивая значение цвета в формате RGB свойству ForeColor.RGB. Третья процедура определяет толщину линии границы, присваивая ее значение свойству WordTextBox.Line.Weight, которое имеет тип Extended. Последняя процедура определяет прозрачность линии и задает для свойства Line.Transparency значение от 0 до 1, имеющее тип Extended. Применяя эти процедуры, мы можем получить результат, который может быть, например, таким, как показано на рис. 8.10.

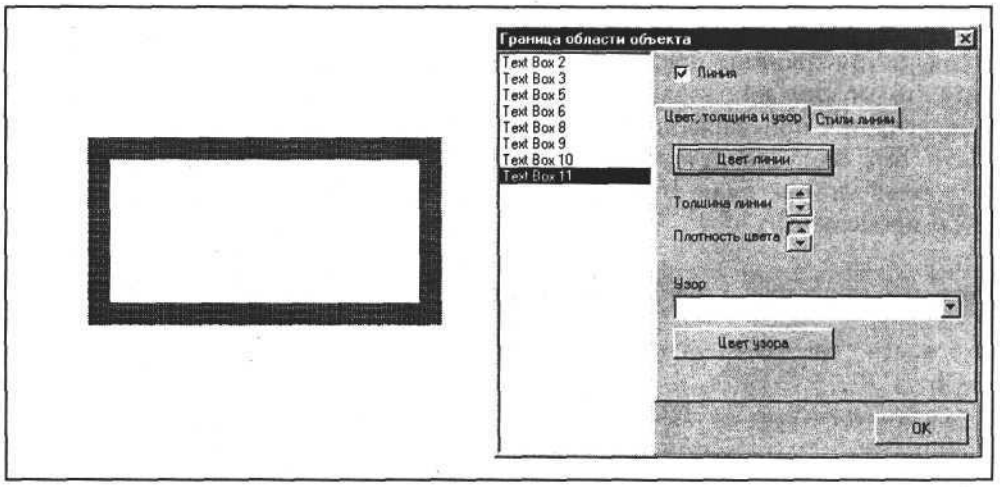


Рис. 8.10. Выбор толщины и цвета линии границы надписи

Для задания вида линии можно использовать узор из набора возможных вариантов. Узор определяется целочисленной константой, которая записывается в поле Line.Pattern. Используя это свойство, запишем значение константы, выбрав его как индекс элемента управления Pattern:TComboBox (комбинированный список).

Выбор узора для линии границы надписи

```
WordTextBox.Line.Pattern:=Pattern.ItemIndex+1;
```

Результат изменений представлен на рис. 8.11.

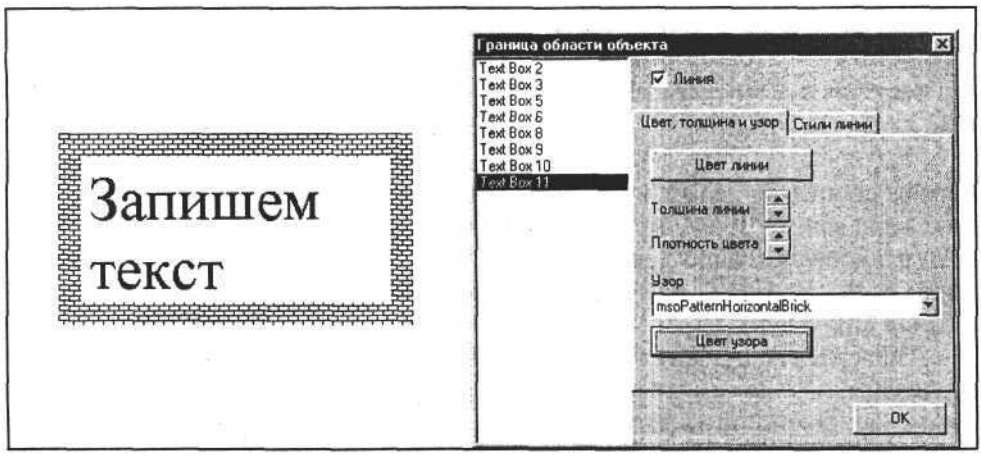


Рис. 8.11. Выбор узора для линии границы надписи

Если линия границы представлена не в виде узора, а в виде сплошной линии, то можно дополнительно задать тип линии и ее шаблон. Тип линии определяется присваиванием целочисленной константы свойству Line.Style. Если это свойство установить в значение msoLineSingle=0, то получится сплошная линия. Если этот стиль нас не устраивает, то используем следующую процедуру.

Задание стиля линии границы надписи

```
procedure TOKBottomDlg4.StyleChange(Sender: TObject);
begin
  WordTextBox.Line.Style:=Style.ItemIndex+1;
end;
```

С помощью этой процедуры мы можем изменить стиль линии, например, как показано на рис. 8.12.

Шаблон линии определяется присваиванием целочисленной константы свойству Line.DashStyle надписи. Если это свойство установить в значение msoLineSolid=0, то линия будет сплошной, без разрывов. Если нужно выбрать другой шаблон линии, то данному свойству можно присвоить значение другой константы из списка возможных.

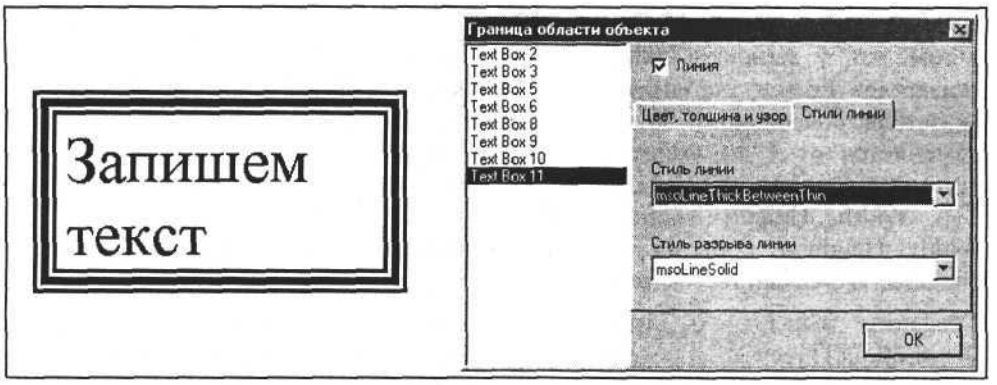


Рис. 8.12. Выбор стиля линии границы надписи

Выбор вида разрыва для линии границы надписи

```
procedure TOKBottomDlg4.DashStyleChange(Sender: TObject);
begin
  WordTextBox.Line.DashStyle:=DashStyle.ItemIndex+1;
end;
```

Пример результата выполнения этой процедуры показан на рис. 8.13.

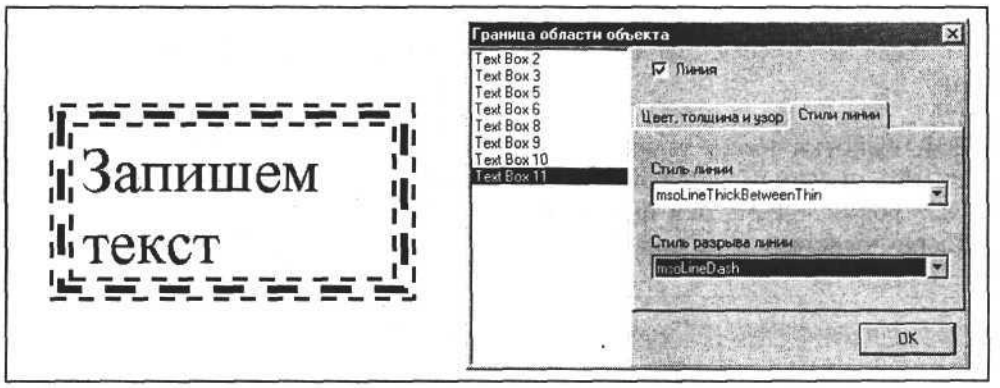


Рис. 8.13. Выбор стиля разрыва линии границы надписи

Выноски

Выноска (объект Callout) является разновидностью надписи. Ее отличие заключается в том, что к прямоугольной области, где размещается текст, добавлена линия-указатель. Общими с надписью для выноски являются свой-

ства заливки и линии для области текста, а также свойства текста. Отличие в том, что у выноски свойства линии применяются и к ломаной линии-указателю. Есть и дополнительный объект Adjustments для описания линии-указателя, содержащий описание параметров линии Adjustments.Item(i), где i изменяется от 1 до Adjustments.Count (общее количество точек перелома линии). Создается объект-выноска с помощью метода AddCallout(Type, Left, Top, Width, Height, Anchor), где Type:integer — тип выноски, Left, Top, Width, Height — координаты и размеры (имеют тип Extended), Anchor — область, где создается выноска.

Создадим выноску с линией-указателем, описываемой тремя точками (msoCalloutThree=3). Для этого используем следующую процедуру, записанную в Delphi. Затем изменим горизонтальную координату внешней точки линии-указателя.

Создание и настройка выноски

```
procedure TOKBottomDlg8.Button1Click(Sender:TObject);
begin
  Form1.W.ActiveDocument.Shapes.AddCallout(msoCalloutThree,
                                    100,10.15,200.25,100);
end;

procedure TOKBottomDlg8.SpinButton1DownClick(Sender: TObject);
begin
  Callout.Adjustments.item(1):=Callout.Adjustments.item[1]-0.01;
end;
```

Текст выноски имеет те же характеристики, что и текст надписи (объекта TextBox), и записывается теми же методами, поэтому процесс записи текста и настройки параметров шрифта здесь не описан. Внешний вид созданной сноски представлен на рис. 8.14.

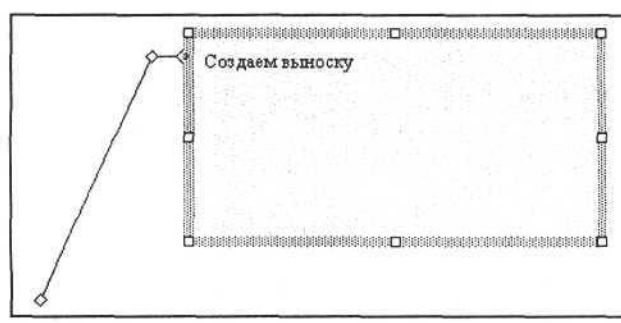


Рис. 8.14. Создание и настройка выноски

Линии

Линия (объект Line) в документе Word создается с помощью метода AddLine коллекции Shapes. Аргументами метода AddLine(BeginX, BeginY, EndX, EndY, Anchor) являются начальные и конечные координаты линии, имеющие тип Extended. Последний аргумент метода (типа Range) определяет область, где и создается линия. Толщина, цвет и другие характеристики линии задаются так же, как для линии границы надписи (объекта TextBox). Так выглядит оператор в среде Delphi, создающий прямую линию (результат его выполнения показан на рис. 8.15):

Добавление и настройка линии

```
W.ActiveDocument.Shapes.AddLine(100,100,200,150);
```

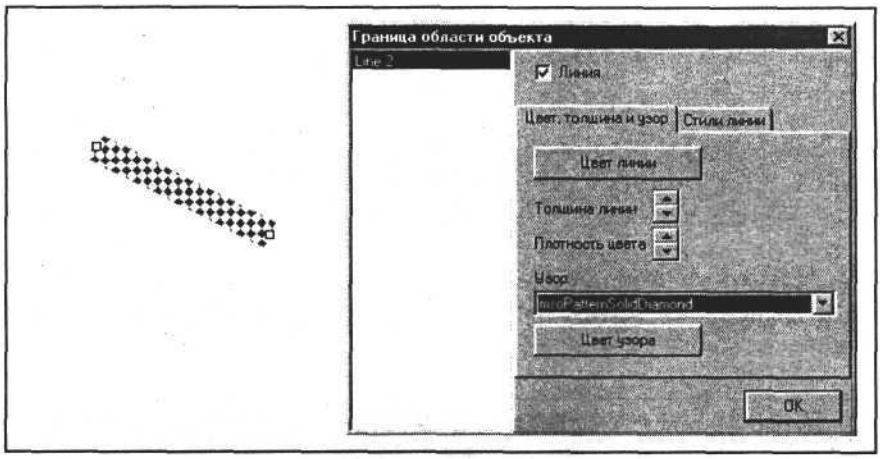


Рис. 8.15. Задание параметров линии

Геометрические фигуры

Метод AddShape коллекции Shapes используется для создания геометрических фигур. Прямые, ломаные и кривые линии создаются другими методами. Вызов этого метода выглядит так:

```
AddShape(Type, Left, Top, Width, Height, Anchor);
```

Первый аргумент представляет собой целое число и определяет тип создаваемого объекта, например Type=1 соответствует прямоугольнику, набор всех возможных типов геометрических фигур составляет более 130. Аргу-

менты Left, Top, Width, Height — числа, имеющие тип Extended и определяющие положение и размеры создаваемого объекта. Аргумент Anchor — объект типа Range, определяющий область, где будет создан объект. Для записи текста в область объекта применяются те же методы и свойства, что и к объекту TextBox (надписи). Создадим прямоугольник с помощью следующих процедур и запишем в него текст.

Создание геометрической фигуры (прямоугольника) и добавление текста

```
procedure TForm1.Button7Click(Sender:TObject);
begin
  W.ActiveDocument.Shapes.AddShape(1,100,100,130,100)
end;

procedure TOKBottomDlg5.ListBox1Click(Sender:TObject);
begin
  Shape:=Form1.W.ActiveDocument.Shapes.Item(
       ListBox1.Items.Strings[ListBox1.ItemIndex])
end;

procedure TOKBottomDlg5.Button1Click(Sender: TObject);
begin
  Shape.TextFrame.TextRange.Text:=Text.Text
end;
```

Результат выполнения этих процедур может выглядеть, как показано на рис. 8.16.

Рис. 8.16. Создание прямоугольника и добавление в него текста

Создав однажды геометрическую фигуру, можно изменить ее внешний вид, изменив значение ее стиля. При этом остальные параметры, например толщина и цвет линий, а также текст, останутся без изменений, если новый тип предполагает их наличие. Для изменения типа фигуры достаточно из-

менить свойство AutoShapeType объекта, записав в него новое значение. Эффект будет таким же, как при создании нового объекта.

Изменим тип и текст созданной ранее геометрической фигуры (прямоугольника) с помощью следующей процедуры.

Изменение типа геометрической фигуры

```
procedure TOKBottomDlg5.AutoShapeTypeChange(Sender:TObject);
begin
  Shape.AutoShapeType:=AutoShapeType.Value;
end;
```

Текст, размещенный в полученной фигуре, может автоматически изменяться, чтобы вписаться в размеры новой фигуры.

Возможные результаты выполнения этой процедуры показаны на рис. 8.17 и 8.18.

Рис. 8.17. Восьмиугольник

Рис. 8.18. "Куб"

Внешние объекты (OLE)

Коллекция Shapes позволяет добавлять в документ объекты, созданные и отображаемые с помощью программ-серверов OLE (OLE-серверов). OLE-объекты в документах Word могут отображаться так же, как они отображаются в приложениях, предназначенных для их создания и редактирования. Если приложение, сопоставленное объекту, не является OLE-сервером, то объект отображается в виде ярлыка. Возможности Word в управлении такими объектами скудны и ограничиваются заданием координат и размеров. Кроме того, можно активизировать данные объекты.

Примечание

OLE (Object Linking and Embedding) — технология связывания и внедрения объектов.

Чтобы управлять внутренними свойствами OLE-объекта, внедренного в документ, необходимо получить доступ к объектам и методам этого объекта.

OLE-объект создается с помощью метода AddOLEObject. Спецификация вызова объекта включает ряд аргументов, передаваемых при вызове метода. Аргументами метода являются имя класса, имя файла, признак связи с файлом на диске и т. д. Есть два синтаксиса вызова метода AddOLEObject. При этом создаются объекты, имеющие некоторые отличия (вы можете разобраться с этими отличиями самостоятельно). Здесь мы рассмотрим создание OLE-объекта и его активизацию в документе Word на примере следующих процедур.

Создание, выбор и активизация OLE-объекта в документе Word

```
procedure TForm1.Button10Click(Sender: TObject);
begin
  W.ActiveDocument.Shapes.AddOLEObject(ClassType:='Excel.Sheet',
                    Left:=10, Top:=10, Width:=100, Height:=100);
end;

procedure TForm1.Button11Click(Sender: TObject);
begin
  if not OpenPictureDialog1.Execute then exit;
  W.ActiveDocument.Shapes.AddOLEObject(ClassType:='Paint.Picture',
                          FileName:=OpenPictureDialog1.FileName,
                      Left:=10, Top:=10, Width:=100, Height:=100);
end;

procedure TOKBottomDlg6.ListBox1Click(Sender: TObject);
begin
MyOleObject:=Form1.W.ActiveDocument.Shapes.Item(ListBox1.ItemIndex+1);
end;

procedure TOKBottomDlg6.Button1Click(Sender: TObject);
begin
  MyOleObject.Activate;
end;
```

Первая процедура создает новый лист Excel, вторая создает OLE-объект — Paint.Picture и загружает реальный рисунок из файла. Третья процедура позволяет выбрать нужный объект из списка коллекции Shapes. Последняя процедура активизирует OLE-объект, что приводит к запуску программы — OLE-сервера. Создадим несколько OLE-объектов различных типов как с помощью приведенного исходного текста, так и обычным путем — как это делают пользователи. Запустим приложение (см. сопроводительный диск книги) и получим список OLE-объектов. После чего мы можем выбрать и активизировать любой из них (рис. 8.19).

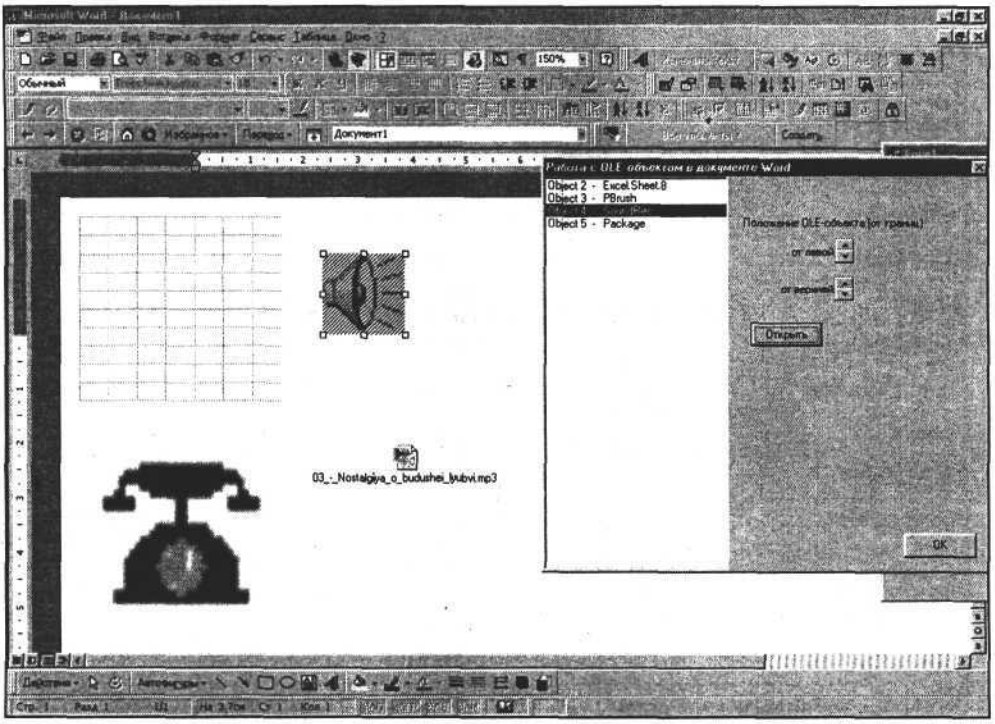


Рис. 8.19. Создание и активизация OLE-объектов в документе Word

Настройка страницы

Настройка страницы определяется ее высотой, шириной, размерами полей, ориентацией и другими параметрами, определяемыми через свойства объекта PageSetup, который, в свою очередь, принадлежит объекту-документу. Для доступа к параметрам страницы используем следующий оператор, возвращающий ссылку на объект:

```
PageSetup:=ActiveDocument.PageSetup;
```

Далее можно считывать или устанавливать параметры страницы документа, но сначала рассмотрим краткую характеристику объекта PageSetup (табл. 8.1).

Таблица 8.1. *Свойства и методы объекта PageSetup*

Свойство или метод	Тип	Краткое описание
LineNumbering	Объект	Нумерация строк
Orientation	Integer	Ориентация

Таблица 8.1 *(окончание)*

Свойство или метод	Тип	Краткое описание
TopMargin	Integer	Верхнее поле
BottomMargin	Integer	Нижнее поле
LeftMargin	Integer	Левое поле
RightMargin	Integer	Правое поле
Gutter	Integer	Переплет
HeaderDistance	Integer	Поле от края до верхнего колонтитула
FooterDistance	Integer	Поле от края до нижнего колонтитула
PageWidth	Integer	Ширина страницы
PageHeight	Integer	Высота страницы
FirstPageTray	Integer	Выбор источника бумаги для первой страницы
OtherPagesTray	Integer	Выбор источника бумаги для следующих страниц
SectionStart	Integer	Начало раздела
OddAndEvenPagesHeaderFooter	Boolean	Различать колонтитулы четных и нечетных страниц
DifferentFirstPageHeaderFooter	Boolean	Различать колонтитул первой страницы
VerticalAlignment	Integer	Выравнивание по вертикали
SuppressEndnotes	Boolean	Запретить концевые сноски
MirrorMargins	Boolean	Зеркальные поля
TextColumns	Объект	Определяет колонки текста в документе и их параметры
PaperSize	Integer	Размер бумаги
SetAsTemplateDefault	Метод	Устанавливает данные настройки в качестве настроек по умолчанию для активного документа и вновь создаваемых документов
TogglePortrait	Метод	Задает всем выделенным страницам (или части выделенных страниц) книжную ориентацию

Создадим новый документ и с помощью объекта PageSetup определим его основные параметры. Для этого используем форму в виде диалога (рис. 8.20). После создания диалога его элементы управления будут содер-

жать информацию об ориентации страницы, ее размерах, полях и стиле вертикального выравнивания текста. Используя эти же элементы управления, мы можем изменить основные параметры страницы, например, ориентацию (книжная/альбомная), стиль вертикального выравнивания, размер бумаги и величины левого и верхнего полей. Рассмотрим следующую процедуру.

Определение параметров страницы

```
procedure TOKBottomDlg7.FormCreate(Sender: TObject);
begin
  PageSetup:=Form1.W.ActiveDocument.PageSetup;
  Orientation.ItemIndex:=PageSetup.Orientation;
  VerticalAlignment.ItemIndex:=PageSetup.VerticalAlignment;
  CheckBox1.Checked:=PageSetup.LineNumbering.Active;
  PageWidth.Value:=PageSetup.PageWidth;
  PageHeight.Value:=PageSetup.PageHeight;
  TopMargin.Value:=PageSetup.TopMargin;
  LeftMargin.Value:=PageSetup.LeftMargin;
end;
```

На рис. 8.20 представлена форма, в элементах которой отображается считанная из свойств объекта PageSetup активного документа информация о параметрах страницы активного документа.

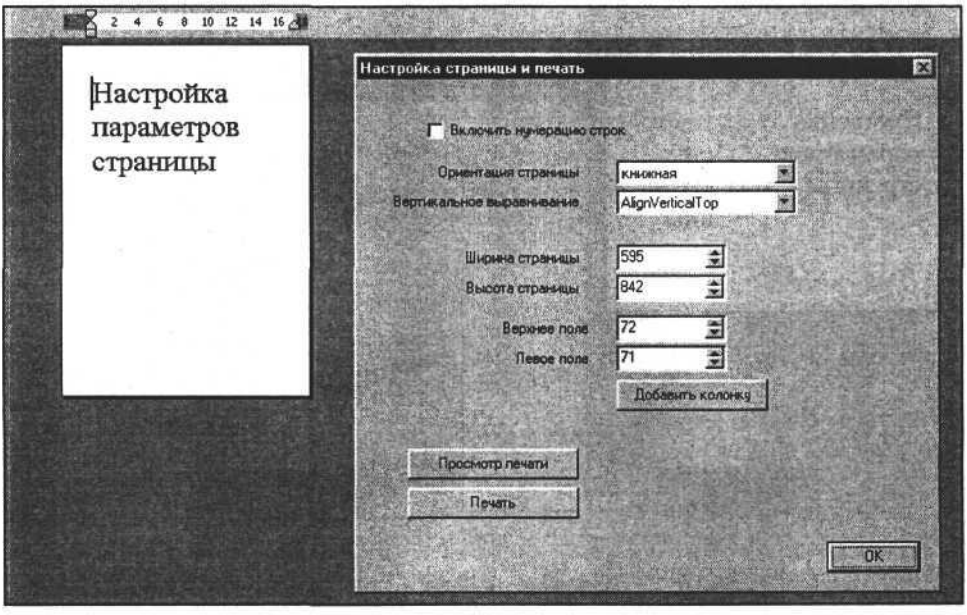


Рис. 8.20. Параметры страницы документа Word

Изменим некоторые параметры страницы. Например, установим альбомную ориентацию и добавим вторую колонку для вывода текста, установив ее ширину в 200 пикселов. Для этой цели используем свойства и методы коллекции TextColumns объекта PageSetup. После изменения ориентации листа с книжной на альбомную или наоборот, считаем новые значения ширины и высоты страницы, а также новые величины полей.

Изменение некоторых параметров страницы

```
procedure TOKBottomDlg7.OrientationChange(Sender: TObject);
begin
  PageSetup.Orientation:=Orientation.ItemIndex;
  PageWidth.Value:=PageSetup.PageWidth;
  PageHeight.Value:=PageSetup.PageHeight;
  TopMargin.Value:=PageSetup.TopMargin;
  LeftMargin.Value:=PageSetup.LeftMargin;
end;

procedure TOKBottomDlg7.Button1Click(Sender: TObject);
begin
  if PageSetup.TextColumns.count<2 then
                    PageSetup.TextColumns.Add(200);
end;
```

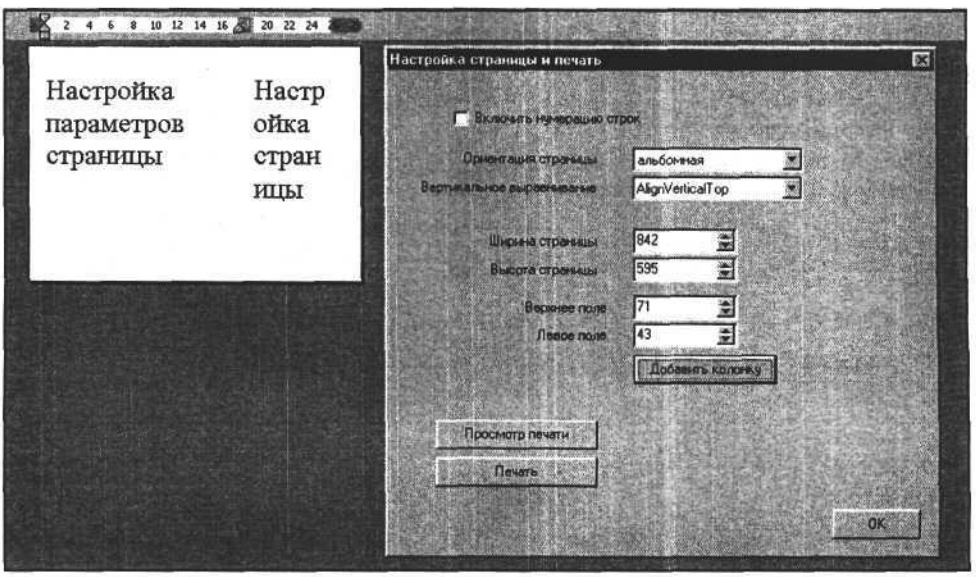


Рис. 8.21. Измененная страница документа Word

Вторая процедура добавляет новую текстовую колонку, если число колонок меньше двух. После выполнения этих манипуляций страница будет выглядеть так, как показано на рис. 8.21.

Печать документа

Печать документа осуществляется с помощью метода PrintOut объекта "документ". Следующие процедуры позволяют выполнить печать активного документа, печать любого из открытых документов (NameDoc — имя или индекс документа) и печать двух копий активного документа:

Печать документа

```
procedure TOKBottomDlg7.Button2Click(Sender: TObject);
begin
  Form1.W.ActiveDocument.PrintOut;
end;

procedure TOKBottomDlg7.Button2Click(Sender: TObject);
begin
  Form1.W.Documents.Item(NameDoc).PrintOut;
end;

procedure TOKBottomDlg7.Button3Click(Sender: TObject);
begin
  Form1.W.ActiveDocument.PrintOut(Copies:=2);
end;
```

Для предварительного просмотра активного документа перед печатью можно использовать следующую процедуру.

Предварительный просмотр документа

```
procedure TOKBottomDlg7.Button2Click(Sender: TObject);
begin
  Form1.W.ActiveDocument.PrintPreview;
end;
```

Мы рассмотрели основные типы объектов, входящих в коллекцию Shapes, после чего можно перейти к практическому использованию полученных знаний.

Рассмотрим пример программы.

Пример программы — формирование товарного ярлыка

Разработаем простую процедуру для формирования товарного ярлыка — объекта TextBox (надпись), содержащего таблицу 2 × 8, ячейки которой заполнены текстом (описанием товара). Внешний вид такого ярлыка представлен на рис. 8.22.

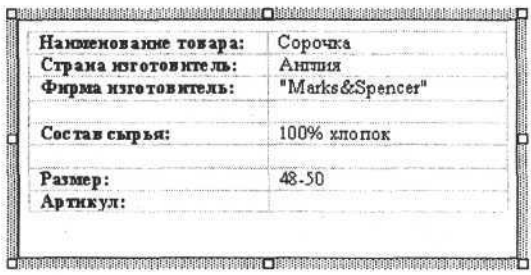


Рис. 8.22. Ярлык на товар, выполненный в виде документа Word

Рассмотрим фрагмент исходного текста программы, который сформирует данный документ.

Создание товарного ярлыка

```
procedure TForm1.Button15Click(Sender: TObject);
 var TextBox:variant;
       Table:variant;
begin
// Создаем новый документ
W.Documents.add;
// Создаем объект TextBox
TextBox:=W.ActiveDocument.Shapes.AddTextbox(msoTextOrientationHorizontal,
                                  10,10,250,120,W.ActiveDocument.Range);
// В области объекта TextBox создаем таблицу
Table:=W.ActiveDocument.Tables.Add(TextBox.TextFrame.TextRange,8,2);
// Линии таблицы делаем невидимыми
Table.Borders.Enable:=False;
// Заполняем таблицу
Table.Cell(1,1).Range:='Наименование товара:';
Table.Cell(1,2).Range:='Сорочка';
Table.Cell(2,1).Range:='Страна изготовитель:';
Table.Cell(2,2).Range:='Англия';
```

```
Table.Cell(3,1).Range:='Фирма-изготовитель:';
Table.Cell(3,2).Range:='"Marks&Spencer"';
Table.Cell(5,1).Range:='Состав сырья:';
Table.Cell(5,2).Range:='100% хлопок';
Table.Cell(7,1).Range:='Размер:';
Table.Cell(7,2).Range:='48-50';
Table.Cell(8,1).Range:='Артикул:';
// Выделяем первый столбец таблицы и изменяем в нем шрифт
Table.Columns.Item(1).Select;
W.Selection.Font.Bold:=True;
TextBox.Select;
end;
```

Результат выполнения данной процедуры показан на рис. 8.22.

Если потребуется создать ярлыки для нескольких товаров, например, указанных в таблице базы данных, то придется изменить эту программу таким образом, чтобы следующий ярлык располагался за текущим, — для этого придется учитывать размеры страницы, а также размеры и координаты расположения ярлыков.

ГЛАВА 9

Работа с объектом Word.Basic

В этой главе рассматриваются следующие темы:

- объектная модель WordBasic;
- загрузка объекта Word.Basic и визуализация окна приложения Word;
- создание документа Word;
- открытие существующего документа Word;
- поиск и редактирование текста в документе Word;
- создание и редактирование таблиц в документе Word;
- рисунки и другие внешние объекты;
- печать документа Word;
- запись документа Word на диск и окончание работы;
- пример программы — платежное поручение.

Объектная модель WordBasic

В предыдущих главах мы рассмотрели архитектуру объекта Word.Application, а также использование этого объекта для работы с документами. Для этого рассматривались внутренние объекты Application, их свойства и методы. Доступ к этим объектам, их свойствам и методам в среде Delphi обеспечивается аналогично доступу в среде программирования Visual Basic для приложения Word. Но приложение Word имеет в своем арсенале еще одно средство для работы с документами — WordBasic.

Word.Basic и Word.Application — разные объекты, но они имеют общие черты. Их сходство заключается в том, что они оба входят в состав приложения Word и предназначены для решения аналогичных задач автоматизации создания документов. Word.Basic, как и Word.Application, обладает инструмен-

том предоставления своих возможностей внешним программам через механизм OLE Automation. Основное отличие WordBasic от Visual Basic состоит в том, что первый предоставляет не структуру связанных объектов со своими свойствами и методами, а набор процедур и функций, количество которых превышает 900.

Между процедурами и функциями WordBasic и Visual Basic можно провести аналогию и сравнить их возможности, что мы и сделаем в этой главе. Сначала рассмотрим объектную модель WordBasic. Структура объекта Word.Basic представлена на рис. 9.1. Она проста и представляет собой набор функций и процедур, принадлежащих корневому объекту, через который внешняя программа может получить доступ к ним.

Рис. 9.1. Объектная модель WordBasic

Для доступа к объекту Word.Basic из приложений, созданных в Deplhi, используем библиотеку ComObj и функцию CreateOleObject, которая возвращает ссылку на объект. В качестве аргумента этой функции используется строка — идентификатор объекта 'Word.Basic'. Рассмотрим следующий фрагмент исходного текста:

Создание объекта Word.Basic

```
uses ComObj
   var WB:variant;
...
procedure TForm1.Button1Click(Sender: TObject);
begin
  WB:=CreateOleObject('Word.Basic');
end;
```

Получив доступ к объекту Word.Basic, можно переходить к решению вопросов автоматизации создания документов Word. В отличие от Visual Basic мы имеем дело с библиотекой процедур и функций, а не с четкой структурой объекта Application. Как быть в данном случае и с чего начать? Лучше всего начать с изучения библиотеки WordBasic и понять, как функции и процеду-

ры этой библиотеки соотносятся со структурой объекта Application. Откроем раздел "Справка по Visual Basic" справки по MS Word. В этом разделе откроем подраздел "Visual Basic Equivalents for WordBasic Commands" (рис. 9.2).

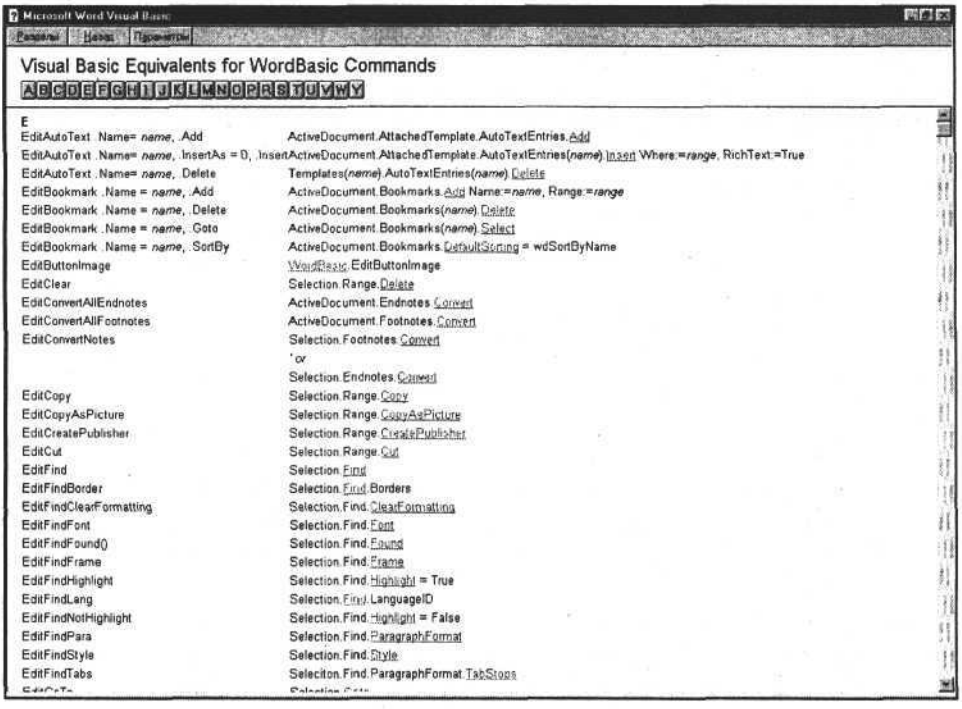


Рис. 9.2. Набор команд WordBasic

На рис. 9.2 слева представлены команды WordBasic, справа — объекты и методы объекта Application, выполняющие аналогичные функции. Для детального анализа возможностей WordBasic сравним несколько его процедур и функций, позволяющих создать или отредактировать простой документ.

Загрузка объекта WordBasic и визуализация окна приложения Word

Загрузка объекта Word.Basic для доступа к его процедурам и функциям производится посредством вызова функции CreateOleObject (см. пример кода "Создание объекта Word.Basic"). После этой процедуры приложение Word будет загружено в память, и мы можем работать с ним и с документами. В начале работы разберемся, как сделать видимым окно приложения Word или убрать его с экрана монитора. В фоновом режиме документы обрабатываются значительно быстрее, а после создания или редактирования можно

отобразить готовый документ. При работе с Word.Application данное действие производилось установкой свойства Visible объекта Application в значение True или False. Кроме того, считывая значение этого свойства, мы могли определить, в каком (видимом/невидимом) состоянии находится окно приложения. В WordBasic для этой цели применяется несколько процедур. Рассмотрим следующие процедуры.

Отображение/скрытие окна приложения Word

```
procedure TForm1.Button2Click(Sender: TObject);
begin
WB.AppShow;
end;

procedure TForm1.Button11Click(Sender: TObject);
begin
WB.AppHide;
end;
```

Первая процедура делает видимым окно приложения Word на экране компьютера. Она аналогична оператору W.Application.Visible=True; в Visual Basic. Вторая процедура убирает с экрана (скрывает) окно приложения Word.

В набор функций, изменяющих визуализацию главного окна Word, входят также функции, позволяющие изменять размеры и положение окна на рабочем столе: AppSize(width, height:integer) — изменяет размеры главного окна приложения Word, AppWindowHeight(height:integer) — изменяет только высоту главного окна приложения Word, AppWindowWidth(width:integer) — изменяет только его ширину, AppWindowPosLeft(horizpos:integer) — изменяет только отступ от левой границы рабочего стола для главного окна приложения Word, AppWindowPosTop(vertpos:integer) — изменяет только отступ от верхней границы рабочего стола для главного окна приложения Word.

Изменение размеров и положения главного окна приложения Word

```
procedure TForm1.Button19Click(Sender: TObject);
 var width_,height_,horizpos_,vertpos_:integer;
begin
width_:=300; height_:=200;
WB.AppSize(width_,height_);
horizpos_:=10; vertpos_:=10;
WB.AppWindowPosLeft(horizpos_);
WB.AppWindowPosTop(vertpos_);
end;
```

В результате выполнения данной процедуры окно переместится в положение с новыми координатами, а размеры окна будут установлены в новые значения (рис. 9.3).

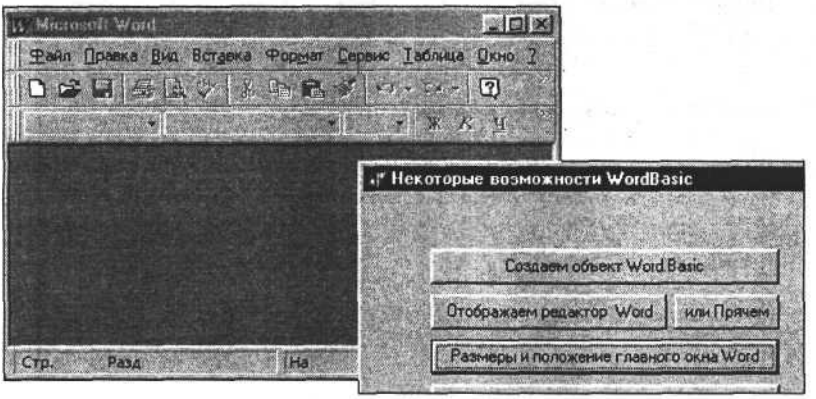


Рис. 9.3. Изменение размеров и координат главного окна MS Word

Видим, что с помощью команд WordBasic можно добиться такого же результата, как и в случае применения Visual Basic для Word.Application. Но наша цель — не только манипуляции с главным окном приложения Word, а гораздо шире — создание и редактирование простых документов в формате MS Word. Для решения такой задачи нужно научиться создавать новый документ, открывать существующий документ, вставлять, удалять и копировать текст и выполнять другие действия, связанные с созданием и редактированием документа.

Приступим к решению этой задачи.

Создание документа Word

Создание нового документа Word в WordBasic в общем случае выполняется процедурой FileNewDefault без аргументов. Данная процедура создает документ, тип которого определяется по умолчанию (обычно — Document Word). Если сохранить такой документ, то получим файл с расширением DOC. Рассмотрим следующую процедуру.

Создание документа по умолчанию

```
procedure TForm1.Button3Click(Sender: TObject);
begin
WB.FileNewDefault;
end;
```

Выполнив данную процедуру, мы получим в окне приложения Word окно нового документа (рис. 9.4).

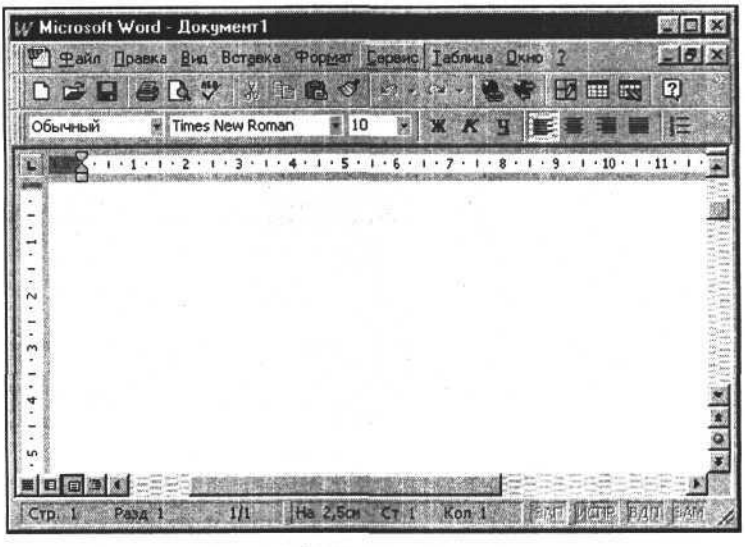


Рис. 9.4. Создан новый документ

Для создания нового документа по шаблону используем команду WordBasic FileNew(FileName), где FileName — путь и имя файла-шаблона (файл с расширением DOT). Рассмотрим следующую процедуру.

Создание документа по шаблону

```
procedure TForm1.Button20Click(Sender: TObject);
begin
if not OpenDialog2.Execute then exit;
WB.FileNew(OpenDialog2.FileName);
end;
```

С помощью данной процедуры мы можем выбрать нужный шаблон и на его основе создать новый документ. Создадим новый документ, используя шаблон "Изысканный отчет" из списка шаблонов MS Word (рис. 9.5).

Из сказанного следует, что WordBasic обладает необходимыми ресурсами для создания документа и эти ресурсы можно задействовать из приложений, создаваемых в среде Delphi. Разница между созданием документа в WordBasic и Visual Basic заключается в том, что последний для этой цели использует коллекцию Documents, что удобнее для дальнейшей работы с созданным документом.

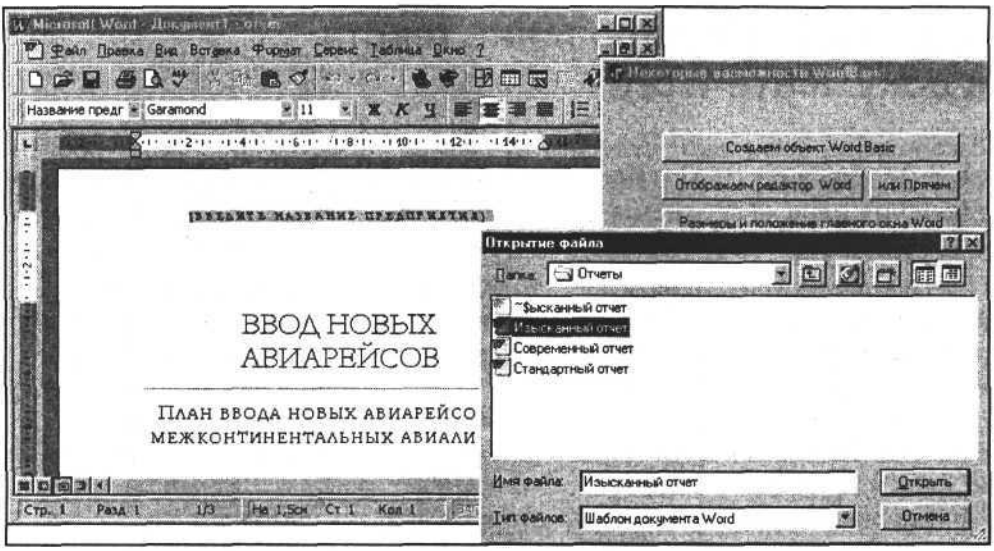


Рис. 9.5. Создание документа на основе шаблона "Изысканный отчет"

Открытие существующего документа Word

Открытие существующего (созданного ранее) документа в WordBasic осуществляется посредством функции FileOpen(FileName), где FileName — имя файла открываемого документа. Рассмотрим следующую процедуру.

Открытие существующего документа

```
procedure TForm1.Button5Click(Sender: TObject);
begin
  if not OpenDialog1.Execute then exit;
  WB.FileOpen(OpenDialog1.FileName);
end;
```

И здесь основное отличие от Visual Basic заключается в том, что просто используется отдельная процедура. Преимущества Visual Basic заключаются в применении методов коллекции Documents, где после открытия или создания документа всегда можно получить доступ к этому или нескольким документам коллекции одновременно.

После создания или открытия документа мы переходим к его редактированию.

Поиск и редактирование текста в документе Word

Рассмотрим несколько простых процедур, позволяющих редактировать текст в документе. Нам потребуются возможности: перемещения по тексту документа, поиска фрагмента текста, копирования и вставки текста, вставки текста из буфера обмена.

Сначала рассмотрим процедуры, позволяющие перемещаться по тексту документа. Их нужно вызывать перед выполнением поиска — для позиционирования курсора. Для поиска фрагмента в документе, скорее всего, дополнительно понадобятся две процедуры — перемещение в начало документа (для того чтобы начать поиск) и перемещение в конец документа (для добавления новых записей). Процедура StartOfDocument перемещает курсор в начало документа, а процедура EndOfDocument — в конец документа. Пример использования этих процедур в среде Delphi выглядит так:

Перемещение курсора в начало и конец документа

```
procedure TForm1.Button8Click(Sender: TObject);
begin
  WB.StartOfDocument;
end;

procedure TForm1.Button9Click(Sender: TObject);
begin
  WB.EndOfDocument;
end;
```

После того как мы переместили курсор в начало документа, можно начать поиск фрагмента текста. Поиск текста производит процедура EditFind(text:string). В случае успешного поиска данная процедура выделяет искомый фрагмент текста. Для продолжения поиска по документу необходимо повторно вызвать эту процедуру. Чтобы убедиться в том, что искомый фрагмент существует и был выделен в результате выполнения процедуры поиска, следует вызвать функцию EditFindFound — она возвращает значение True в результате успешного поиска. Рассмотрим следующий фрагмент исходного текста:

Поиск фрагмента текста в документе

```
procedure TForm1.Button10Click(Sender: TObject);
 var text_:string;
```

```
begin
  text_:=InputBox('Введем фрагмент текста для поиска','','');
  WB.EditFind(text_);
  if WB.EditFindFound
  then messagebox(handle,'Фрагмент текста найден!','Внимание!',0)
  else messagebox(handle,'Фрагмент текста не найден!','Внимание!',0);
end;
```

Используя приведенную процедуру, попробуем найти фрагмент текста в документе. Для этого откроем документ и введем искомое слово (рис. 9.6).

Рис. 9.6. Введем слово для поиска в документе

После того как слово для поиска введено, процедура WB.EditFind(text_) выполнит поиск. В случае успеха она выделит фрагмент текста, а функция WB.EditFindFound возвратит значение True. Когда заданный текст не найден, то никакой фрагмент не будет выделен, а WB.EditFindFound возвратит значение False. Если выполнить представленную выше процедуру дважды, то поиск продолжится с теми же параметрами и будет выделен следующий фрагмент, удовлетворяющий условиям поиска (рис. 9.7 и 9.8).

После того как курсор перемещен в заданное место документа, туда можно вставить текст. Если перед этим был выделен фрагмент текста, то вставляемый текст заменит этот фрагмент. Для этой цели используем функцию WB.insert(text_:string), где text_:string — вставляемый фрагмент текста. Рассмотрим следующую процедуру.

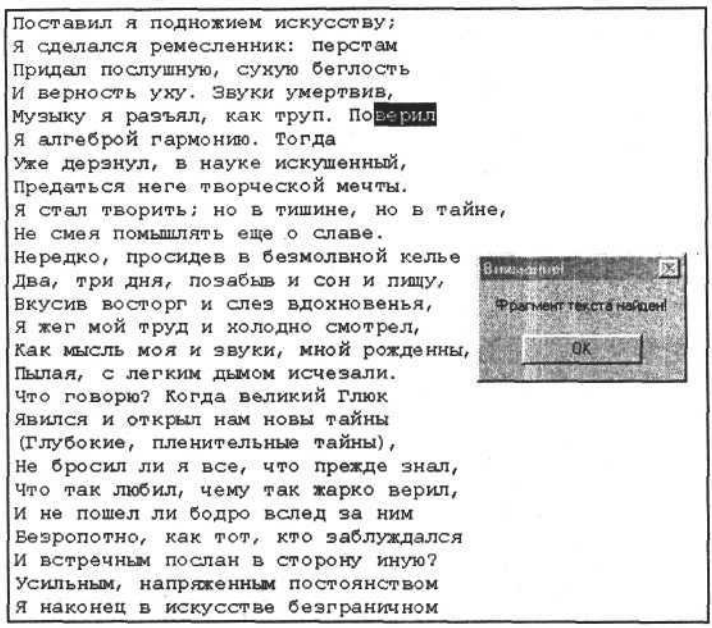


Рис. 9.7. Результатом поиска является выделенный фрагмент текста в документе

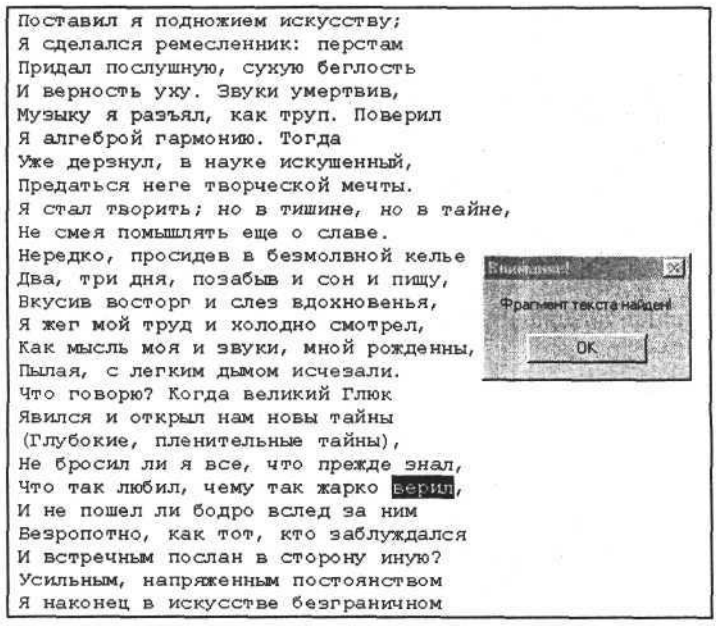


Рис. 9.8. Повторный поиск выделяет следующий фрагмент текста

Вставка текста в документ

```
procedure TForm1.Button7Click(Sender: TObject);
 var text_:string;
begin
  text_:=InputBox('Вставляем текст в документ','','');
  WB.insert(text_);
end;
```

Если комбинировать функции поиска и вставки текста, то можно заполнить подготовленный заранее шаблон документа. Можно заметить, что методы объектов Word.Application, в отличие от процедур WordBasic, имеют более гибкие возможности для редактирования текста. Но все же WordBasic позволяет создать простой документ, пример которого мы рассмотрим в конце этой главы.

Можно упомянуть еще несколько функций, которые пригодятся любому программисту Delphi, решившему использовать WordBasic в своей работе. Это функции выделения всего документа, копирования в буфер обмена, вставки из буфера обмена в документ и разрыва страницы. Рассмотрим следующие процедуры.

Процедуры работы с буфером обмена

```
procedure TForm1.Button12Click(Sender: TObject);
begin
  WB.EditSelectAll;
  WB.EditCopy;
end;

procedure TForm1.Button13Click(Sender: TObject);
begin
  WB.EditPaste;
end;
```

Первая процедура выделяет весь текст документа и копирует его в буфер обмена, вторая — вставляет текст из буфера обмена на место положения курсора или вместо выделенного текста.

Вставка разрыва страницы

```
procedure TForm1.Button14Click(Sender: TObject);
 const wdPageBreak=7;
begin
  WB.InsertBreak(Type:=wdPageBreak);
end;
```

Данная процедура вставляет в текст символ конца страницы, после которого начинается следующая страница.

Мы исследовали основные методы работы с текстом. Далее исследуем возможности WordBasic, которые будем использовать для создания и редактирования таблиц в документе Word.

Создание и редактирование таблиц в документе Word

Таблицы в документах Word представляют собой объекты, являющиеся продолжением основного текста в документе. Например, если мы создаем таблицу, то она создается в месте нахождения курсора и, таким образом, разрывает текст. То есть если требуется создать таблицу в определенном месте документа, то необходимо предварительно установить в это место курсор.

Создадим таблицу в нашем документе, для этого используем функцию TableInsertTable(NumColumns, NumRows), где NumColumns и NumRows — количества столбцов и строк в создаваемой таблице. Есть и другие синтаксисы данной функции (см. справку по WordBasic).

Рассмотрим, как выглядит в Delphi фрагмент программы, создающей таблицу.

Создание таблицы

```
procedure TOKBottomDlg3.Button15Click(Sender: TObject);
 var NumColumns, NumRows:integer;
begin
// Создаем таблицу размером 2 x 5
NumColumns:=2; NumRows:=5;
Form1.WB.TableInsertTable(NumColumns, NumRows);
// Отображаем линии таблицы
Form1.WB.TableGridlines(true);
// Переходим в конец документа
Form1.WB.EndOfDocument;
// Вставляем пробел
Form1.WB.insert(' ');
// Переходим в конец документа
Form1.WB.EndOfDocument;
end;
```

Итак, данная процедура включает в себя несколько операторов. Первые два пропускаем. Следующий оператор создает таблицу. Следующий за ним оператор представляет собой вызов процедуры, устанавливающей свойства ли-

ний таблицы. Если единственный аргумент процедуры TableGridlines установлен в значение True, то линии границ таблицы отображаются. Чтобы скрыть линии границ, вызовем метод WB.TableGridlines(False). Далее переходим в конец документа и вставляем символ пробела, чтобы следующая вставляемая таблица не слилась с этой таблицей.

Если между существующей таблицей и вновь создаваемой, следующей за первой, нет ни одного символа, то вторая таблица просто сольется с первой. В этом случае создание новой таблицы будет иметь эффект добавления новых строк к существующей таблице. Добавить или вставить новую строку в таблицу можно другим способом — с помощью процедуры TableInsertRow. Если вызвать эту процедуру без аргумента, то будет вставлена одна строка; если при вызове в качестве аргумента задать целое число, то оно определит количество вставляемых строк. Эта процедура выполняется, если курсор находится в области таблицы, иначе генерируется ошибка.

Рассмотрим следующую процедуру.

Добавление строки в таблицу

```
procedure TOKBottomDlg3.Button16Click(Sender: TObject);
begin
  Form1.WB.TableInsertRow;          // Добавляем одну строку
  Form1.WB.TableInsertRow(3);       // Добавляем три строки
end;
```

В результате выполнения этой и предыдущей процедур (см. пример кода "Создание таблицы") получаем две таблицы с заданными параметрами (рис. 9.9).

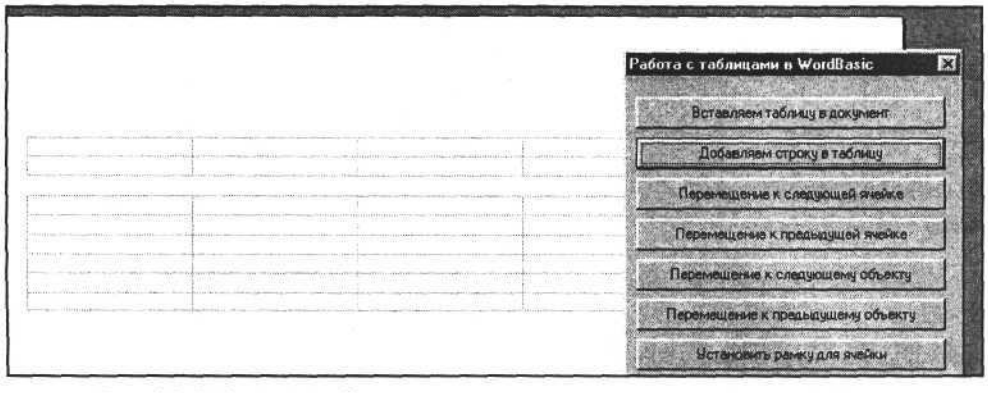


Рис. 9.9. Добавление таблиц в документ и вставка строк в последнюю таблицу

Создав таблицу, можно приступить к заполнению ее ячеек текстом. Вставка текста в ячейки таблицы производится так же, как обычная вставка текста

в документ. Для этого достаточно установить курсор в необходимую позицию. Последовательное перемещение по ячейкам таблицы можно выполнять несколькими способами.

Используем для этого процедуры NextCell и PrevCell, позволяющие двигаться последовательно от первой ячейки к последней или наоборот. Дополнительно к перемещению курсора от первой к последней ячейке процедура NextCell добавляет новую строку в конец таблицы, если до этого курсор находился в последней ячейке таблицы.

Рассмотрим применение этих процедур в Delphi на следующем примере программного кода.

Перемещение курсора в таблице

```
procedure TOKBottomDlg3.Button17Click(Sender: TObject);
begin
  Form1.WB.NextCell;
end;

procedure TOKBottomDlg3.Button18Click(Sender: TObject);
begin
  Form1.WB.PrevCell;
end;
```

Организовать перемещение курсора в таблице можно и с помощью другой пары процедур — NextObject и PrevObject. В отличие от уже рассмотренных процедур, они позволяют перемещаться и между таблицами, не изменяя при этом саму структуру таблицы. Пример их использования в приложениях Delphi:

Перемещение от объекта к объекту

```
procedure TOKBottomDlg3.Button19Click(Sender: TObject);
begin
  Form1.WB.NextObject;
end;

procedure TOKBottomDlg3.Button20Click(Sender: TObject);
begin
  Form1.WB.PrevObject;
end;
```

Для перемещения непосредственно в конец или начало строки или столбца (и в начало или конец таблицы) применяют следующие четыре команды WordBasic.

Перемещение в начало или конец строки или столбца

```
// Перемещаемся в начальную ячейку строки
procedure TOKBottomDlg3.Button3Click(Sender: TObject);
begin
  Form1.WB.StartOfRow;
end;

// Перемещаемся в последнюю ячейку строки
procedure TOKBottomDlg3.Button4Click(Sender: TObject);
begin
  Form1.WB.EndOfRow;
end;

// Перемещаемся в верхнюю ячейку столбца
procedure TOKBottomDlg3.Button5Click(Sender: TObject);
begin
  Form1.WB.StartOfColumn;
end;

// Перемещаемся в нижнюю ячейку столбца
procedure TOKBottomDlg3.Button6Click(Sender: TObject);
begin
  Form1.WB.EndOfColumn;
end;
```

Понятно, что, комбинируя представленные процедуры, можно перемещаться в начало или конец таблицы, а также перемещаться между таблицами в документе.

Перемещаясь по документу, используя функции поиска текстовых фрагментов или команды перемещения по объектам или по ячейкам таблицы, можно вставлять текст в любое место документа.

Рисунки и другие внешние объекты

Кроме возможности вставить текст, WordBasic предоставляет функции для добавления в документ рисунков и других внешних объектов (OLE-объектов). Рассмотрим, как можно вставить рисунок.

Для того чтобы вставить рисунок или другой внешний объект, воспользуемся, например, следующим набором процедур — InsertObject, InsertSound и InsertPicture. Первые две вставляют OLE-объекты, отображение которых обеспечивается программами — OLE-серверами, третья процедура вставляет рисунок. Рассмотрим следующий фрагмент исходного текста.

Вставка рисунка в документ Word

```
procedure TForm1.Button18Click(Sender: TObject);
begin
  if not OpenPictureDialog1.Execute then exit;
  WB.InsertPicture(OpenPictureDialog1.FileName, False, False);
end;
```

Процедура InsertPicture имеет три аргумента: первый — строка (имя графического файла); второй — логический (связь файла с документом); третий — логический (признак внедрения рисунка в документ).

После формирования документа переходим к его печати.

Печать документа Word

Для печати документа достаточно использовать две команды WordBasic — процедуру просмотра FilePrintPreview и процедуру печати документа FilePrint. Последнюю процедуру можно вызывать без аргументов или с аргументами, например для задания числа копий печати.

Примеры использования данных процедур в приложениях Delphi:

Печать документов Word

```
procedure TForm1.Button17Click(Sender: TObject);
begin
  WB.FilePrintPreview;
end;

procedure TForm1.Button16Click(Sender: TObject);
begin
  WB.FilePrint;
end;
```

В процедуре для печати вместо команды FilePrint можно использовать команду FilePrintDefault или команду FilePrint(c:integer), где c — количество печатаемых копий. Для предварительного просмотра печати также можно использовать команду FilePrintPreviewFullScreen, которая разворачивает окно предварительного просмотра печати во весь экран. Для просмотра в окне нескольких печатаемых страниц применяется команда FilePrintPreviewPages(p:integer), где p — количество одновременно отображаемых страниц в одном окне.

Следующий пример программного кода иллюстрирует использование данных функций.

Печать документов Word

```
procedure TForm1.Button16Click(Sender: TObject);
begin
  WB.FilePrint(2);
end;

procedure TForm1.Button17Click(Sender: TObject);
begin
  WB.FilePrintPreviewPages(2);
end;
```

Запись документа Word на диск и окончание работы

Записать документ в файл можно различными способами — все зависит от выбора пользователя и от конкретной реализации документа. Для сохранения документа служат следующие команды WordBasic.

Процедура FileSave сохраняет документ под тем же именем, под которым он был открыт. Если документ создан вновь, то данная процедура неприемлема.

Для сохранения нового документа предназначена процедура FileSaveAs(file:string), где file — имя файла, в котором сохраняется документ.

Когда открыто несколько документов и требуется сохранить их, используется процедура FileSaveAll.

Рассмотрим, как можно применять их в проектах Delphi на следующем примере.

Сохранение документа Word в файле

```
procedure TForm1.Button15Click(Sender: TObject);
begin
  WB.FileSave;
end;

procedure TForm1.Button15Click(Sender: TObject);
begin
  WB.FileSaveAs('1.doc');
end;

procedure TForm1.Button15Click(Sender: TObject);
begin
  WB.FileSaveAll;
end;
```

Как показал наш краткий обзор, WordBasic обладает всеми возможностями для создания и редактирования простого документа. В качестве примера рассмотрим процедуру создания платежного поручения. В *главе 6* мы создавали этот документ с помощью Visual Basic. Теперь у нас появилась возможность сравнить решения одной и той же задачи разными способами.

Пример программы — платежное поручение

Разработаем прототип шаблона платежного поручения. Чтобы облегчить себе задачу, воспользуемся справочно-правовой системой "Гарант" и получим форму платежного поручения в формате DOC. Отредактируем этот документ и приведем его к виду, показанному на рис. 9.10. Далее сохраним его в формате шаблона (файл с расширением DOT) для последующего использования в процедуре формирования выходной формы документа "Платежное поручение".

Как показано на рис. 9.10, преобразование документа в шаблон заключается в том, что в некоторые ячейки внесен текст типа "###Сумма прописью&". Эти фрагменты текста и представляют собой текстовые константы, используемые для подстановки информации из программы.

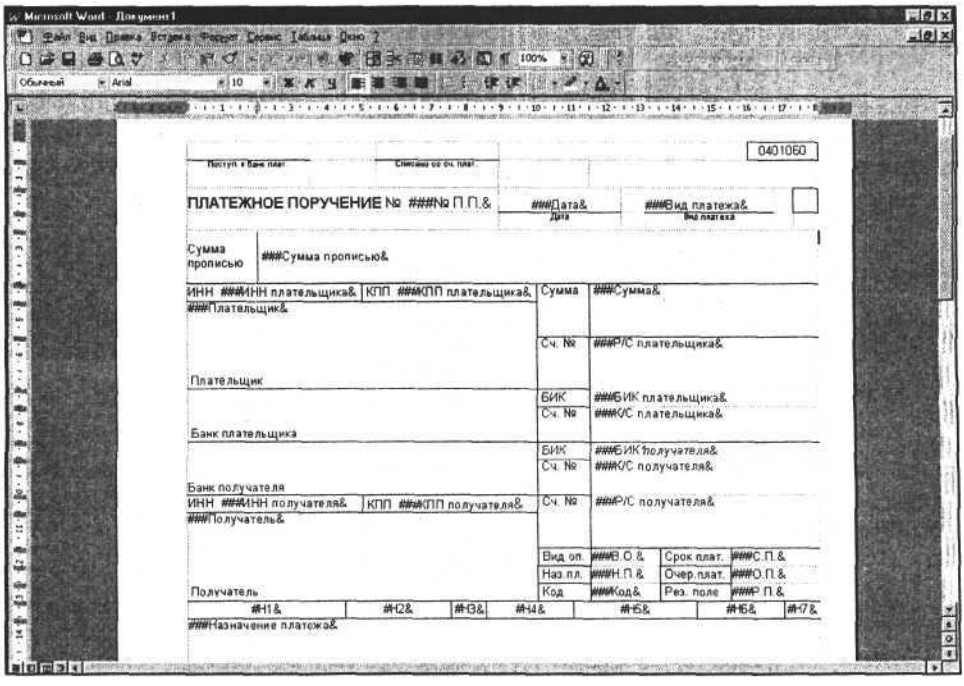


Рис. 9.10. Вид шаблона документа "Платежное поручение"

Используем этот шаблон для создания документа. Следующий фрагмент программы демонстрирует, как это сделать.

Формирование платежного поручения

```
// Данная функция позволяет находить строку FindText
// и подставлять на ее место строку ReplacementText.
function FindAndInsert(FindText,ReplacementText:string):boolean;
 label 0,1,2;
begin
Form1.WB.StartOfDocument;
1: Form1.WB.EditFind(FindText);
  if Form1.WB.EditFindFound then begin Form1.WB.insert(ReplacementText);
  goto 1;
  end;
end;
// Данная процедура заполняет шаблон документа.
//Pocedure TForm1.Button14Click(Sender: TObject);
begin
// Создаем новый документ по шаблону
W.documents.Add(ExtractFileDir(Application.ExeName)+'\Шаблон платежного'+
                                ' поручения.dot');
WB.TableGridlines(false);
// Подставляем текст.
FindAndInsert('###№ П.П.&','1');
FindAndInsert('###Дата&',datetostr(date));
FindAndInsert('###Вид платежа&','почтой');
FindAndInsert('###Сумма прописью&','Двести пятьдесят рублей сорок
 копеек');
FindAndInsert('###Сумма&','250,40');
FindAndInsert('###ИНН плательщика&','0000000000');
FindAndInsert('###КПП плательщика&','000000000011');
FindAndInsert('###Плательщик&','ЗАО Селена');
FindAndInsert('###Р/С плательщика&','00000000000000000000');
FindAndInsert('###БИК плательщика&','000000');
FindAndInsert('###К/С плательщика&','00000000000000000000');
FindAndInsert('###ИНН получателя&','1111111111');
FindAndInsert('###КПП получателя&','111111111100');
FindAndInsert('###БИК получателя&','111111');
FindAndInsert('###К/С получателя&','11111111111111111111');
FindAndInsert('###Р/С получателя&','11111111111111111111');
FindAndInsert('###Получатель&','ЗАО Комета');
FindAndInsert('###В.О.&',' ');
```

```
FindAndInsert('###Н.П.&',' ');
FindAndInsert('###Код&',' ');
FindAndInsert('###С.П.&',' ');
FindAndInsert('###О.П.&',' ');
FindAndInsert('###Р.П.&',' ');
FindAndInsert('#Н1&',' ');
FindAndInsert('#Н2&',' ');
FindAndInsert('#Н3&',' ');
FindAndInsert('#Н4&',' ');
FindAndInsert('#Н5&',' ');
FindAndInsert('#Н6&',' ');
FindAndInsert('#Н7&',' ');
FindAndInsert('###Назначение платежа& ','Оплата за поставку товара');
end;
```

Представленный здесь текст программы показывает, что большой разницы между реализацией в WordBasic и в Visual Basic почти нет, но некоторые отличия все же есть. Какие и чем они обусловлены? Они очевидны и вы можете их легко обнаружить сами. Результат выполнения данной процедуры показан на рис. 9.11.

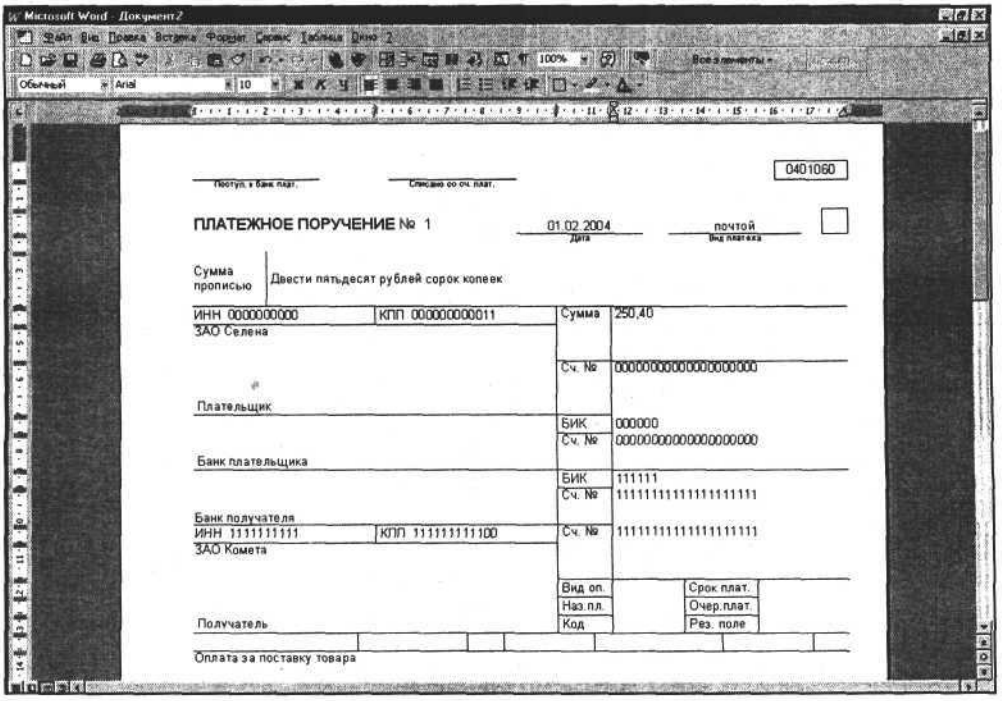
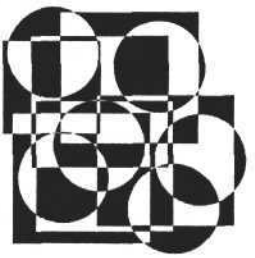


Рис. 9.11. Сформированный документ "Платежное поручение"

После создания документа можно переходить к его печати.

В этой главе мы рассмотрели особенности применения WordBasic в приложениях Delphi. Далее нет необходимости углубляться в эту тему, т. к. объект Word.Application предоставляет больше возможностей, например, доступ и программирование свойств элементов управления MS Word, что мы и рассмотрим в следующей главе.



ГЛАВА 10

Программирование свойств MS Word

В этой главе рассмотрены следующие темы:

- элементы управления приложения MS Word;
- элементы коллекции CommandBars, их отображение и расположение;
- создание пользовательской панели или меню;
- элементы управления и их свойства;
- главное меню;
- создание нового элемента управления;
- создание и использование макроса Visual Basic средствами Delphi;
- коллекция диалогов;
- пример программирования панели.

Элементы управления приложения MS Word

В *главах 5—9* были рассмотрены объекты, которые по большей части характеризуются как подчиненные по отношению к корневому объекту "документ". Они в своей совокупности представляют содержимое и характеристики документа. Используя эти объекты, мы можем создавать документ и заполнять его содержание. Таким образом, мы используем приложение MS Word как универсальный редактор нужных документов (например, отчетов), и свойств этих объектов нам вполне достаточно, но Word.Application дает программистам больше возможностей, которые нужно использовать. Ранее была рассмотрена объектная модель Application, в которой элементы управления выделены в отдельную коллекцию самим приложением Word. Коллекция CommandBars представляет собой совокупность панелей инст-

рументов (далее — панели) и панели главного меню (далее — главное меню), а также подобных элементов (панелей и меню), созданных пользователем. Каждый из этих элементов, в свою очередь, является хозяином кнопок (панели) или пунктов меню (главное меню и другие меню).

Рассмотрим свойства коллекции CommandBars. Как и любая коллекция, она содержит набор элементов и имеет свойство Count (количество элементов коллекции). Свойства и методы коллекции CommandBars представлены в табл. 10.1.

Таблица 10.1. *Основные свойства и методы коллекции CommandBars*

Свойство или метод	Тип	Описание
ActionControl	Объект	Ссылка на текущий активный элемент управления
ActiveMenuBar	Объект	Ссылка на текущий активный элемент меню
Add	Метод	Добавляет элемент коллекции
Count	Integer	Количество элементов коллекции
DisplayTooltips	Boolean	Включает подсказку для кнопок
DisplayKeysInTooltips	Boolean	Включает подсказку сочетания "горячих" клавиш для кнопок
FindControl	Метод	Осуществляет поиск элемента коллекции
Item(i:integer)	Объект	Элемент коллекции
LargeButtons	Boolean	Переключает отображение больших/маленьких кнопок
MenuAnimationStyle	Integer	Задает эффект при выводе меню
ReleaseFocus	Метод	Обновляет пользовательский интерфейс для всех элементов коллекции

Исследуем некоторые общие свойства коллекции CommandBars. Например, свойство коллекции LargeButtons позволяет выбирать размеры кнопок всех панелей — большие или маленькие. Установим значение этого свойства в True. Для этого можно использовать следующий программный код.

Изменение размера кнопок

```
procedure TForm1.LargeButtonsClick(Sender: TObject);
begin
  W.CommandBars.LargeButtons:=LargeButtons.Checked;
end;
```

Выполняя данную процедуру, мы получим результат, который не нуждается в дальнейших комментариях и представлен на рис. 10.1.

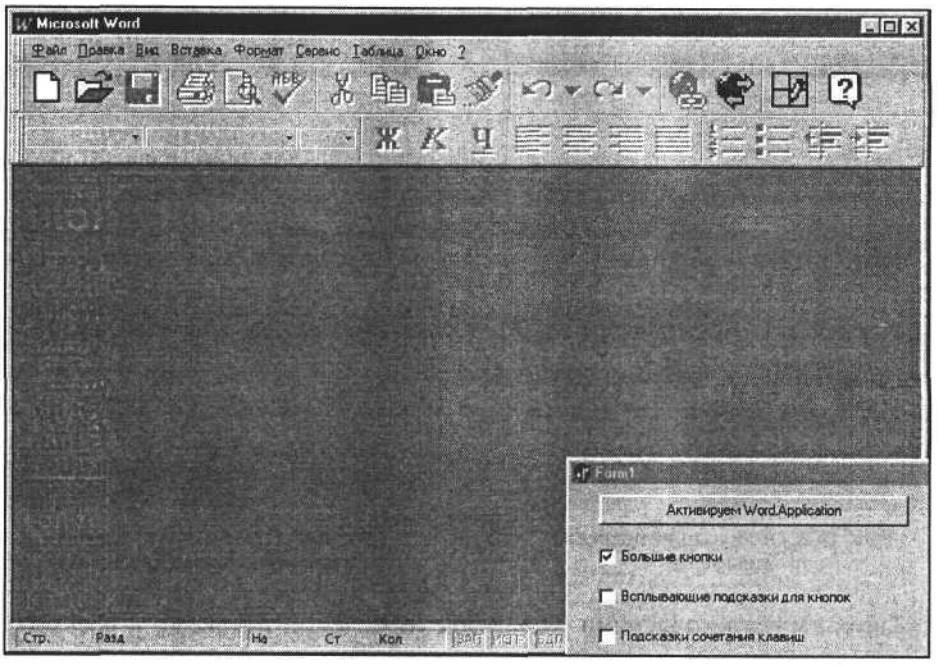


Рис. 10.1. Выбираем большие кнопки для панелей управления

Другим общим свойством коллекции панелей управления является режим показа всплывающих подсказок для кнопок, определяемый двумя свойствами логического типа: DisplayTooltips — определяет, включена или отключена подсказка; DisplayKeysInTooltips — определяет присутствие в подсказке "горячих" клавиш. Изменение режима подсказок определяется просто установкой этих свойств в определенное значение. Рассмотрим фрагмент исходного текста.

Изменение режима всплывающих подсказок для кнопок

```
procedure TForm1.DisplayTooltipsClick(Sender: TObject);
begin
  W.CommandBars.DisplayTooltips:=DisplayTooltips.Checked;
end;

procedure TForm1.DisplayKeysInTooltipsClick(Sender: TObject);
begin
  W.CommandBars.DisplayKeysInTooltips:=DisplayKeysInTooltips.Checked;
end;
```

Свойства DisplayTooltips и DisplayKeysInTooltips коллекции CommandBars, используемые в представленных процедурах, позволяют изменить режимы всплывающих подсказок (рис. 10.2 и 10.3).

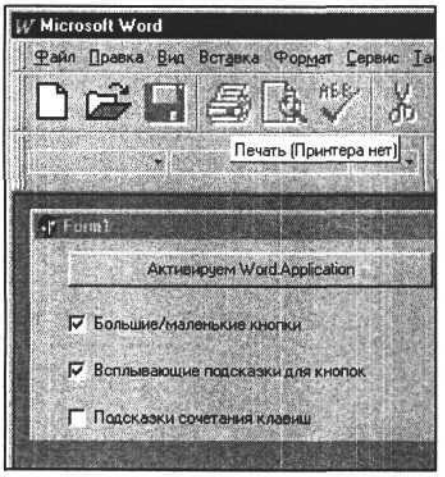


Рис. 10.2. Отображение подсказки для кнопки

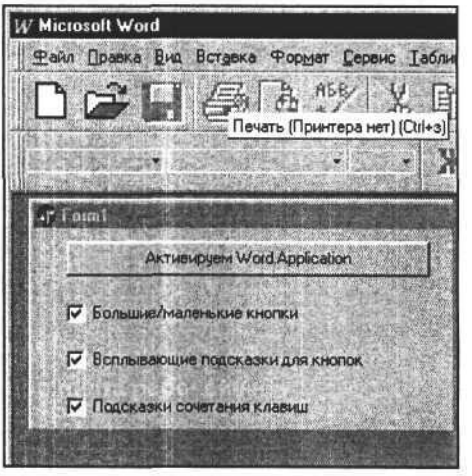


Рис. 10.3. Отображение подсказки и сочетания клавиш для кнопки

От общих свойств коллекции элементов управления перейдем к рассмотрению самих элементов коллекции — панелей управления и их свойств. Доступ к ним осуществляется через объекты Item(i:integer), где i — индекс элемента в коллекции.

Элементы коллекции CommandBars, их отображение и расположение

Для начала получим список всех элементов, входящих в коллекцию CommandBars. Для этого, используя свойство Count коллекции и свойство элемента Name, загрузим список элементов в объект типа TCheckListBox (прокручиваемый список с флажками). Свойство Name имеет строковый тип и представляет собой имя элемента. Его можно использовать для доступа к любому элементу коллекции аналогично индексу, имеющему числовой тип. Рассмотрим следующий фрагмент исходного текста в Delphi.

Список элементов коллекции панелей и меню

```
procedure TOKBottomDlg2.FormCreate(Sender: TObject);
 var a_:integer;
 eee_:string;
```

```
begin
COMMANDBARS:=Form1.W.COMMANDBARS;
for a_:=1 to COMMANDBARS.Count do begin
  eee_:=COMMANDBARS.Item[a_].name+' = '+COMMANDBARS.Item[a_].NameLocal;
  CheckListBox1.Items.Add(eee_);
  CheckListBox1.Checked[a_-1]:=CommandBars.Item[a_].Visible;
  end;
end;
```

Данная процедура, последовательно перебирая все элементы коллекции, загружает их имена в объект CheckListBox1. Дополнительно загружаются значения свойств NameLocal (имя, отображаемое в заголовке панели и соответствующее национальной версии Word, в данном случае — русскоязычной) и Visible (режим отображения элемента в окне Word). Далее, перемещаясь по списку имен загруженных в CheckListBox1 элементов, можно получить или изменить значения других свойств выбранной панели. Полезны, например, свойства: Position, Left, Top, Width, Height:integer — положение и размеры панели; RowIndex:integer — номер строки, занимаемой панелью, когда панели собраны в какой-либо части главного окна; если панель содержит кнопки или другие элементы управления, то свойство BuiltIn:boolean имеет значение True, иначе — значение False; Context — содержит строку со ссылкой на файл шаблона; Protection:integer — определяет режим защиты панели от изменений со стороны пользователя (возможные значения этого свойства, а также свойства Position см. в *Приложении 1*). С целью изучения некоторых из этих свойств рассмотрим следующий фрагмент программы, который представляет собой процедуру обработки для объекта CheckListBox1.

Выбор панели управления, изменение ее свойства Visible и расположения

```
procedure TOKBottomDlg2.CheckListBox1Click(Sender: TObject);
begin
CommandBars.Item[CheckListBox1.ItemIndex+1].Visible:=
        CheckListBox1.Checked[CheckListBox1.ItemIndex];
end;

procedure TOKBottomDlg2.PositionClick(Sender: TObject);
begin
CommandBars.Item[CheckListBox1.ItemIndex+1].Position:=Position.ItemIndex;
end;
```

Результат наших манипуляций со свойствами Visible и Position может быть таким, как показано на рис. 10.4. В данном случае для панели **Стандартная** свойство Position имеет значение msoBarFloating (4).

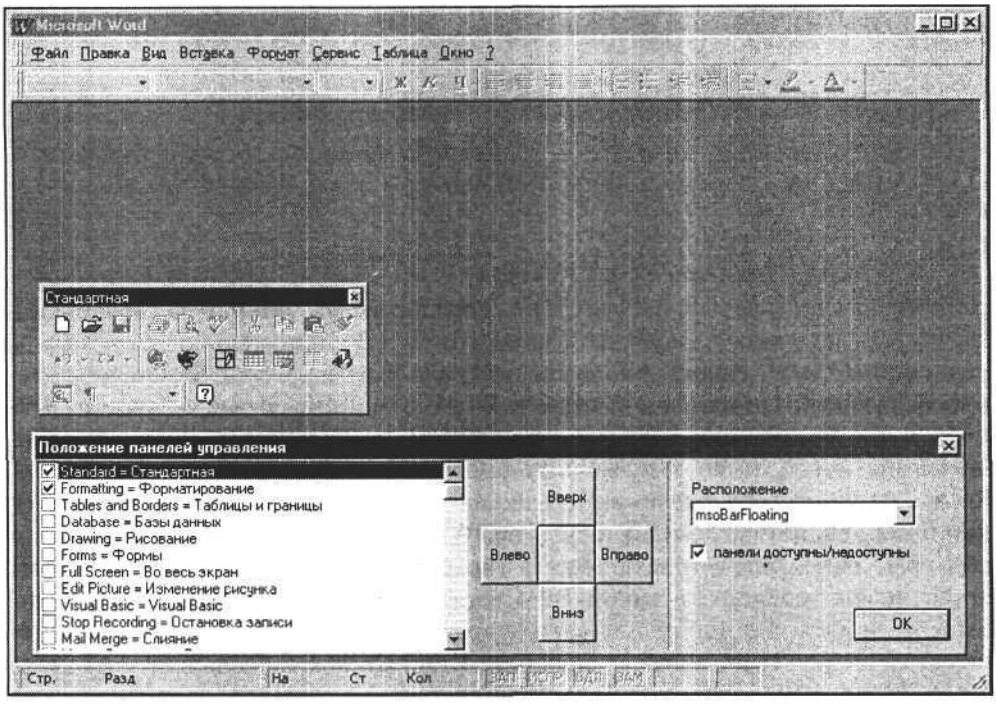


Рис. 10.4. Изменение положения панели

Далее, в режиме, когда панель находится в центре как самостоятельное окно, ее положение можно изменить — как по вертикали, так и по горизонтали. Для этого достаточно изменить значения свойств Left и Top. Следующая процедура позволяет смещать панель вправо.

Смещение панели вправо

```
procedure TOKBottomDlg2.Button3Click(Sender: TObject);
begin
  CommandBars.Item[CheckListBox1.ItemIndex+1].left:=
        CommandBars.Item[CheckListBox1.ItemIndex+1].left+10;
end;
```

Результат выполнения процедуры показан на рис. 10.5.

Нужно отметить, что не все панели доступны для изменений их свойств. Некоторые панели не могут находиться в главном окне и располагаются в других окнах, например в окне просмотра печати. Изменение их свойств вызовет ошибку выполнения. Но большинство панелей все же доступны для изменения их свойств. В качестве примера можно отобразить некоторые

панели в главном окне и придать им новое расположение. Рассмотрим процедуру, отображающую панель **Цвет заливки**.

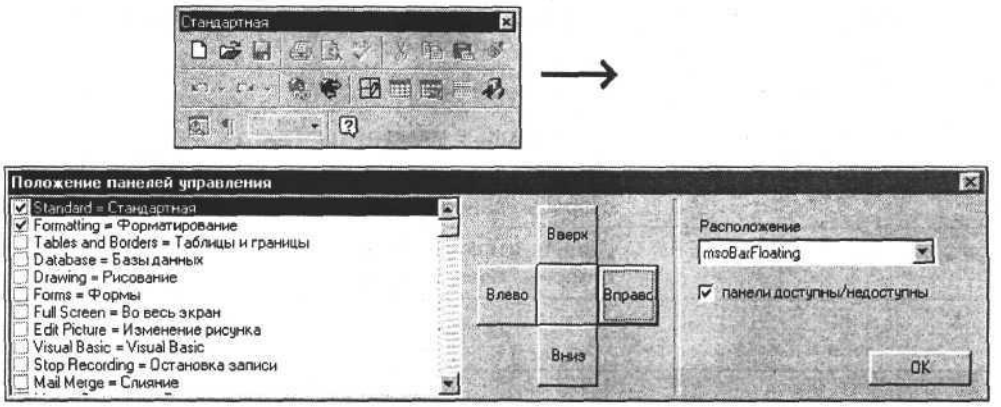


Рис. 10.5. Смещение положения панели

Отображение панели *Цвет заливки*

```
procedure TOKBottomDlg2.CheckListBox1Click(Sender: TObject);
begin
  CommandBars.Item['Fill Color'].Visible:=True;
end;
```

В результате получим панель, представленную на рис. 10.6.

Если попытаться изменить положение панели **Цвет заливки**, т. е. пристыковать к одной из сторон главного окна, то результат будет отрицательным, т. к. тип этой панели не предусматривает таких изменений. Тип определяется при создании панели, для стандартных панелей изменить его нельзя; также к этим панелям нельзя применять метод Delete; есть и другие ограничения. Если тип панели позволяет, то ее расположение можно изменить, например, пристыковав к одной из сторон главного окна. Для этого свойству Position объекта "панель" нужно присвоить одно из значений, указанных в табл. 10.2.

Таблица 10.2. *Возможные положения панелей*

Константа	Значение	Описание
msoBarFloating	4	Отдельная панель в центральной части главного окна экрана
msoBarBottom	3	Панель с кнопками, расположенная вдоль нижней стороны главного окна

Таблица 10.2 *(окончание)*

Константа	Значение	Описание
msoBarLeft	0	Панель с кнопками, расположенная вдоль левой стороны главного окна
msoBarMenuBar	6	Системное меню
msoBarPopup	5	Всплывающее меню
msoBarRight	2	Панель с кнопками, расположенная вдоль правой стороны главного окна
msoBarTop	1	Панель с кнопками, расположенная вдоль верхней стороны главного окна

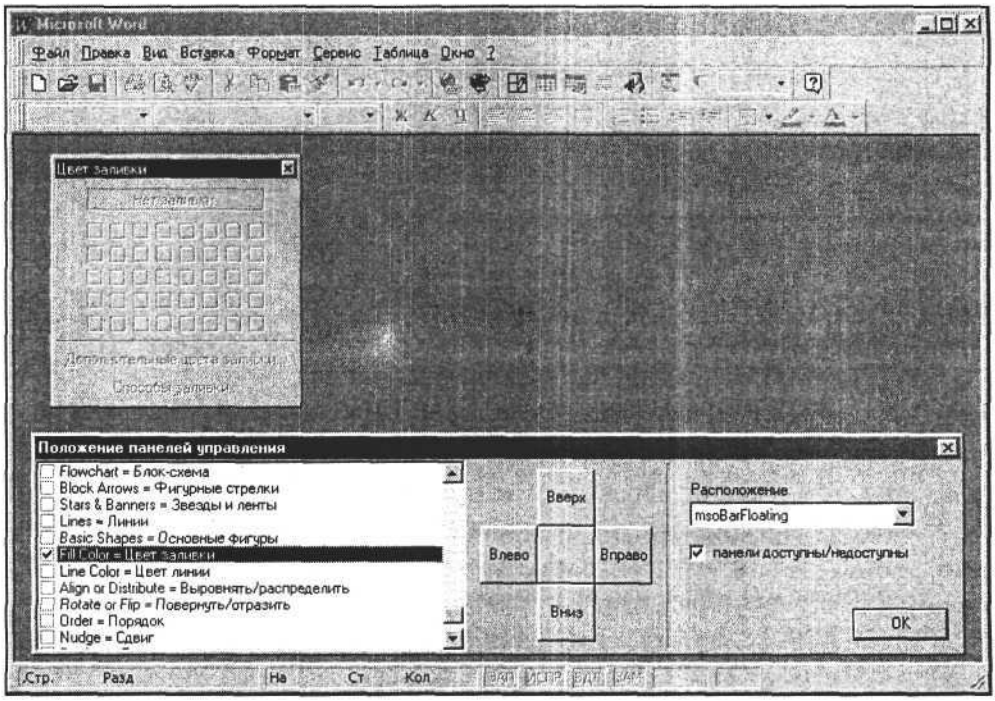


Рис. 10.6. Открываем панель **Цвет заливки**

В качестве примера изменим положение панели **Стандартная**, используя следующий оператор:

```
CommandBars.Item['Standard'].Position:=msoBarLeft;
```

где msoBarLeft=0. Результат представлен на рис. 10.7.

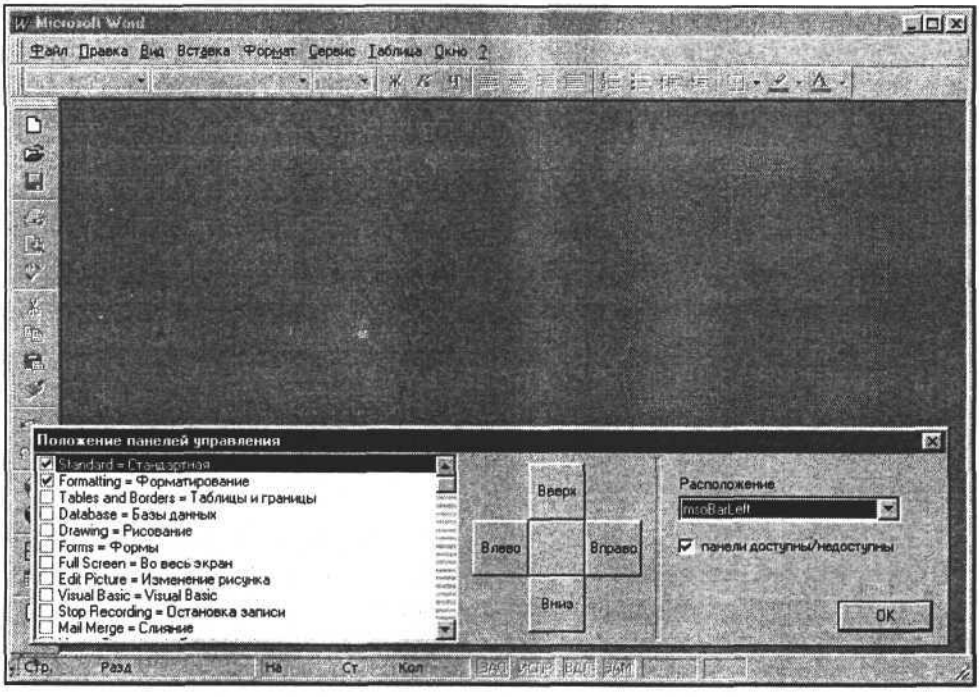


Рис. 10.7. Панель Стандартная расположена вдоль левой стороны окна

Рассмотрим еще одно свойство панели, которое позволяет активировать или деактивировать панель в главном окне приложения Word. Свойство панелей Enabled имеет тип Boolean и определяет возможность доступа пользователя к ним. Когда свойство Enabled имеет значение False, пользователь не может пользоваться данной панелью — она вообще не отображается. Если установить это свойство в значение True, то панель будет доступна и займет свое прежнее место в окне приложения Word. Возможно, этим свойством необходимо воспользоваться, когда требуется блокировать доступ пользователя к некоторым элементам управления. Рассмотрим следующую процедуру.

Отключение панелей управления

```
procedure TOKBottomDlg2.CheckBox1Click(Sender: TObject);
 var a_:integer;
begin
  for a_:=1 to CommandBars.Count do
    CommandBars.Item[a_].Enabled:=CheckBox1.Checked;
end;
```

Данная процедура, перебирая все элементы коллекции CommandBars, устанавливает их свойство Enabled либо в значение True, либо в значение False

в зависимости от состояния свойства Checked объекта CheckBox1. Установим с ее помощью (для этого надо снять соответствующий флажок в форме) свойство Enabled в значение False для всех панелей. Результат выполнения процедуры представлен на рис. 10.8.

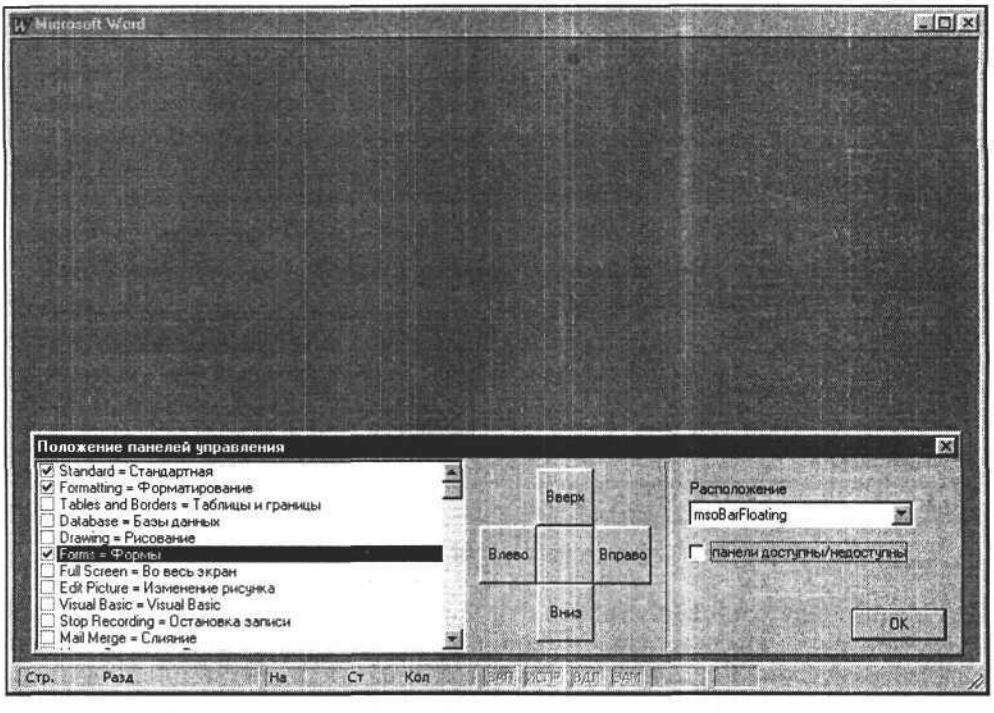


Рис. 10.8. Отключаем панели управления

Мы рассмотрели некоторые свойства встроенных панелей приложения Word. Есть возможность программно создавать и удалять пользовательские панели и меню. При работе со стандартными панелями и меню есть некоторые особенности, например их нельзя удалить. Далее рассмотрим создание новой панели (или меню).

Создание пользовательской панели или меню

Для создания пользовательской панели используем метод Add коллекции CommandBars. У этого метода есть несколько аргументов. Синтаксис вызова метода Add:

```
Add(Name, Position, MenuBar, Temporary);
```

где Name:string — название создаваемой панели, Position:integer — ее расположение (см. табл. 10.2), MenuBar:boolean — признак создания меню, Temporary:boolean — признак создания временной панели. Используя данные аргументы и их комбинации, мы можем как создать обычную панель с кнопками или меню, так и заменить существующее главное меню приложения Word. Любой из создаваемых компонентов коллекции CommandBars может быть создан как временный объект (на время одного сеанса работы приложения) или как постоянный объект. Рассмотрим создание новой панели в приложениях Delphi на примере следующей процедуры.

Создание пользовательской панели

```
procedure TForm1.Button2Click(Sender: TObject);
 const msoBarFloating=4;
 var CommandBar_:variant;
begin
  CommandBar_:=W.CommandBars.Add('Пользовательская панель',
               msoBarFloating, False, True);
  CommandBar_.Enabled:=True;
  CommandBar_.Visible:=True;
end;
```

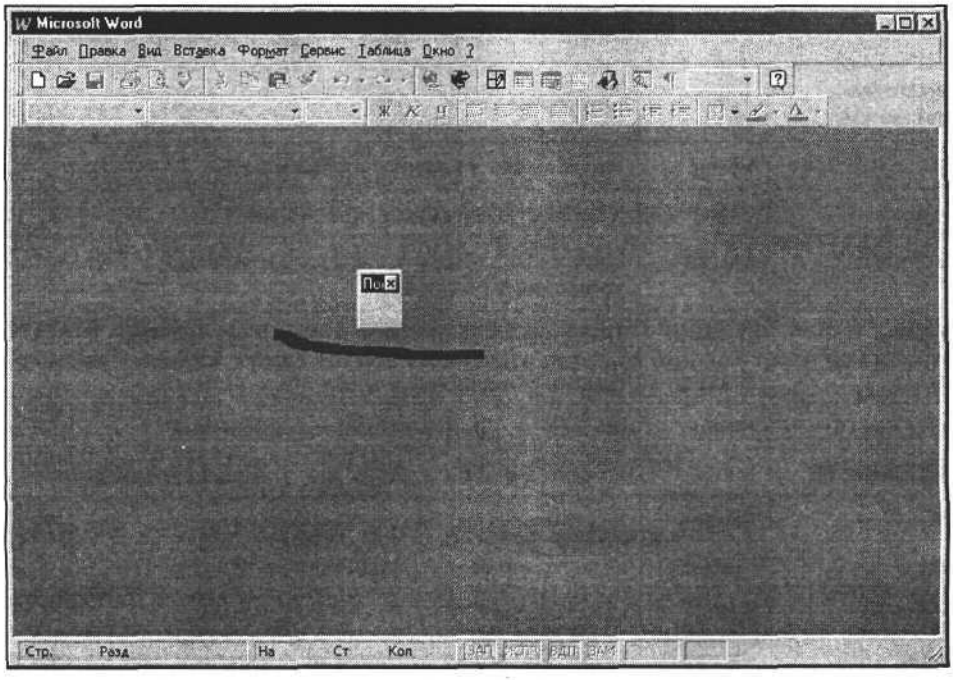


Рис. 10.9. Созданная пользовательская панель

По конкретным условиям, задаваемым аргументами в данной процедуре, новая панель создается и отображается в центральной части главного окна. Результат выполнения процедуры представлен на рис. 10.9. Поскольку на панели нет ни одной кнопки, ее размер задан по умолчанию.

Мы исследовали общие свойства панели, одинаково присущие как панелям с кнопками, так и меню. Чтобы понять отличия, необходимо рассмотреть содержание самих панелей.

Элементы управления и их свойства

Как известно, панель может содержать кнопки и другие элементы управления, например, раскрывающиеся списки или пункты меню. Эти элементы в свою очередь принадлежат самой панели и объединены в коллекцию Controls. Посредством этой коллекции осуществляется доступ ко всем элементам управления выбранной панели. Коллекция Controls имеет несколько свойств и один метод. Мы будем использовать свойства Count (количество элементов коллекции) и Item(i:integer) (элементы коллекции, где i — индекс кнопки), а также метод Add, позволяющий добавлять на панель новые элементы.

Рассмотрим свойства и методы элемента управления (табл. 10.3).

***Таблица 10.3.** Свойства и методы элемента управления*

Свойство или метод	Тип	Описание
DescriptionText	String	Назначение элемента управления
Caption	String	Надпись на элементе управления
Execute	Метод	Выполнить команду, ассоциированную с элементом управления (кнопкой)
Enabled	Boolean	Доступный/недоступный элемент управления
Visible	Boolean	Видимый/невидимый элемент управления
HelpFile	String	Файл помощи для элемента управления
HelpContextID	Integer	Индекс в файле помощи для элемента управления
ID	Integer	Идентификатор элемента управления
Index	Integer	Индекс элемента управления на панели
OnAction	String	Имя макроса для элемента управления
Reset	Метод	Сбросить пользовательские настройки для элемента управления

Таблица 10.3 *(окончание)*

Свойство или метод	Тип	Описание
SetFocus	Метод	Активировать элемент управления
TooltipText	String	Всплывающая подсказка для элемента управления

Исследуем содержание панели и некоторые свойства элементов управления. Если рассматривать меню, то его основное отличие от панели с кнопками состоит в том, что элементом управления в составе меню может быть как пункт меню, так и подменю со своими элементами управления и т. д.

Чтобы проанализировать свойства панели, используем свойства Count и Item() коллекции Controls, а также свойства Caption и TooltipText. Загрузим весь список кнопок и других элементов управления панели в объект ListBox с помощью следующей процедуры.

Загрузка списка элементов управления панели

```
procedure TOKBottomDlg5.FormCreate(Sender: TObject);
 var a_:integer;
 eee_:string;
begin
  MYCONTROLS:=OKBottomDlg4.CommandBars.Item[
OKBottomDlg4.ListBox1.ItemIndex+1].CONTROLS;
  for a_:=1 to MYCONTROLS.Count do begin
   eee_:=MYCONTROLS.Item[a_].Caption+'='+MYCONTROLS.Item[a_].TooltipText;
   ListBox1.Items.Add(eee_);
  end;
  MYCONTROL:=MYCONTROLS.Item[1];
end;
```

Результат выполнения процедуры представлен на рис. 10.10.

Как видно из рис. 10.10, переходя от одного элемента управления к другому, мы можем получать и изменять их свойства. Для этого достаточно загрузить в переменную типа variant ссылку на элемент Item(), например, как в следующем операторе:

```
MYCONTROL:=MYCONTROLS.Item[ListBox1.ItemIndex+1];
```

Далее, работая со ссылкой, мы получаем доступ к различным свойствам выбранного элемента управления. В качестве примера сделаем невидимой выбранную кнопку, изменим ее надпись (свойство Caption) и запустим на выполнение команду, связанную с данной кнопкой.

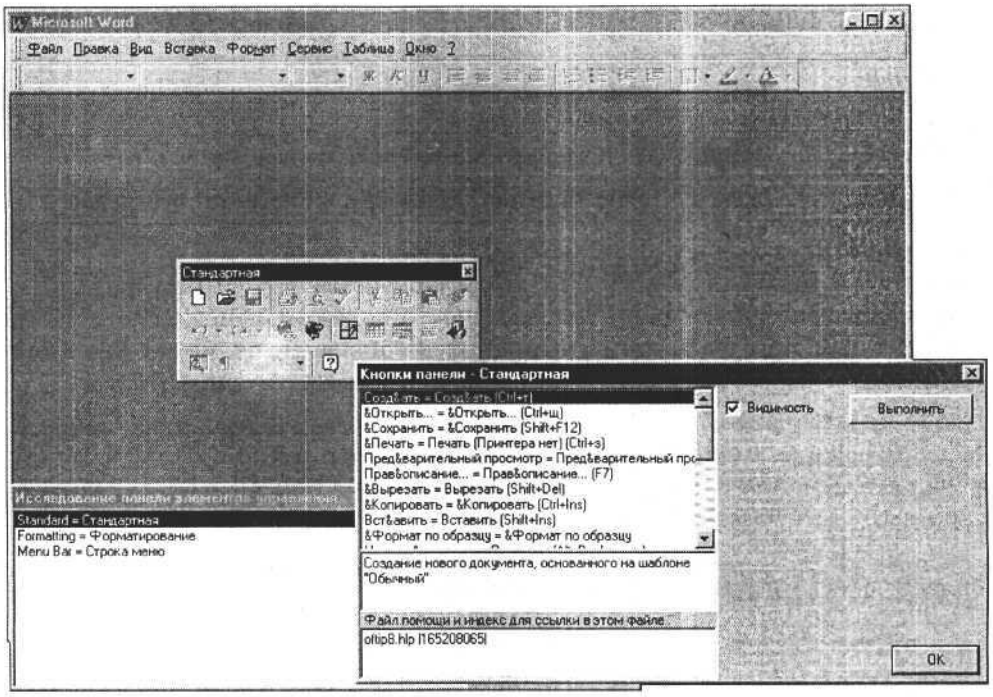


Рис. 10.10. Состав панели

Управление свойствами элемента управления

```
procedure TOKBottomDlg5.b_visibleClick(Sender: TObject);
begin
  MYCONTROL.Visible:=b_visible.Checked;
end;

procedure TOKBottomDlg5.b_CaptionClick(Sender: TObject);
begin
  MYCONTROL.Caption:='Новая кнопка';
end;

procedure TOKBottomDlg5.Button1Click(Sender: TObject);
begin
  MYCONTROL.Execute;
end;
```

Главное меню

В отличие от панели инструментов, главное меню и его элементы имеют несколько иные свойства, основным отличием является возможность для каждого элемента меню содержать свою коллекцию элементов. Проще говоря, каждый элемент меню может содержать подменю, элементы которого также могут содержать свое подменю. Таким образом, меню имеет древовидную структуру. В этом случае доступ к любым элементам обеспечивается посредством элементов коллекций Controls, принадлежащих индивидуально каждому элементу меню. На рис. 10.11 представлен пример объектной модели меню.

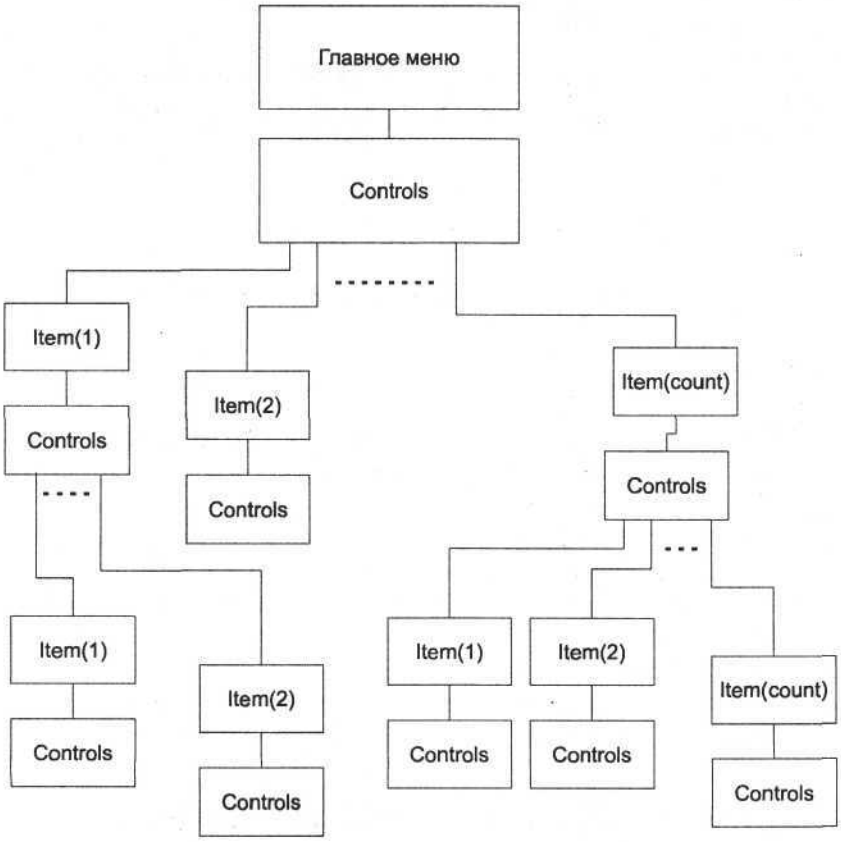


Рис. 10.11. Объектная модель главного меню

Используя коллекции Controls и методы Execute и SetFocus элементов этих коллекций, можно развернуть главное меню примерно так, как показано на рис. 10.12. Исходный текст этого модуля программы и форму для среды

разработки приложений Delphi можно посмотреть в приложении на сопроводительном диске книги.

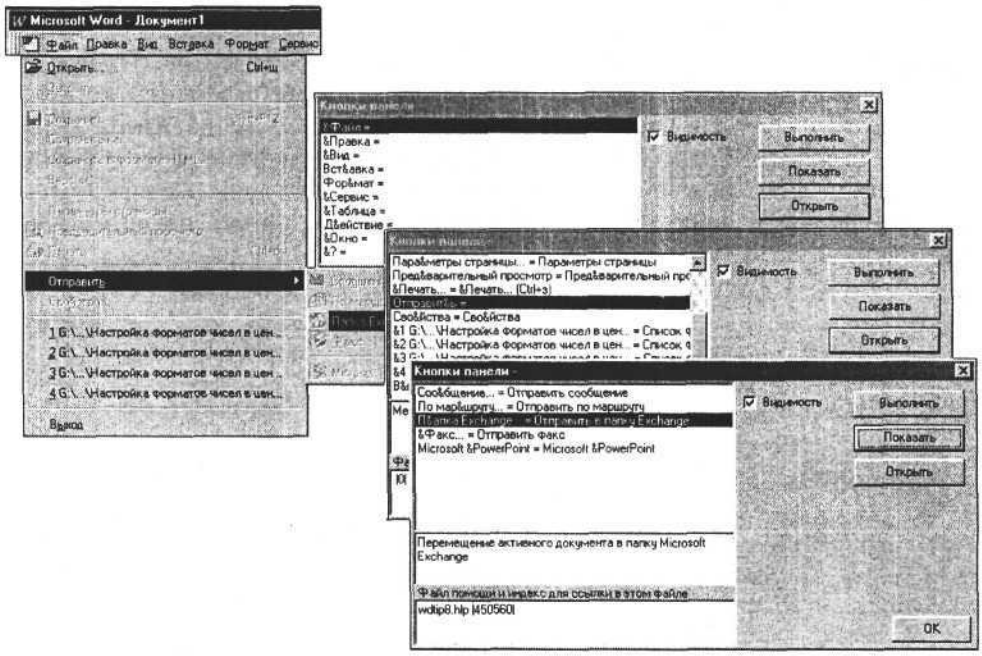


Рис. 10.12. Анализ объекта "Главное меню"

Из рис. 10.12 понятно, что мы раскрываем подменю (дочерний набор элементов меню), используя свойства и методы родительских объектов, которым принадлежат коллекции Controls (раскрываемое подменю). В качестве примера открываем меню **Файл** (элемент главного меню) и подменю **Отправить** (элемент меню **Файл**). Метод Execute запускает команду, связанную с выбранным элементом меню. Если с этим элементом связано подменю, то данная команда раскрывает его. Метод SetFocus устанавливает фокус на данный элемент меню.

От исследования существующих элементов управления меню и панелей инструментов перейдем к созданию собственных элементов.

Создание нового элемента управления

Все элементы управления любой панели собраны в коллекцию Controls. Данная коллекция не только объединяет элементы и их свойства, но и обладает методами, позволяющими создавать или удалять элементы управления. Рассмотрим два метода коллекции. Метод Add — создает новый элемент, метод Delete — удаляет выбранный элемент. Причем метод Delete

принадлежит элементу коллекции, а не самой коллекции (как метод Add) и его нельзя применить к стандартным элементам коллекции, а только к пользовательским.

Рассмотрим синтаксис вызова метода Add. В справочной системе он выглядит так:

```
Add(Type, ID, Parameter, Before, Temporary);
```

где Type:integer — тип объекта, ID:integer — идентификатор объекта[1], Parameter:string — имя связанной с данным элементом команды, Before:boolean — признак разделителя, Temporary:boolean — признак создания временного элемента. Возможные типы создаваемых объектов указаны в табл. 10.4.

Таблица 10.4. *Возможные типы создаваемых объектов*

Константа	Значение	Тип создаваемого объекта
msoControlButton	1	Кнопка
msoControlEdit	2	Поле ввода
msoControlDropdown	3	Раскрывающийся список
msoControlComboBox	4	Комбинированный список
msoControlPopup	10	Элемент меню

Создадим обычную кнопку на панели **Стандартная**. Используем метод Add, первый аргумент которого (Type) равен msoControlButton, а второй (ID) — единице. Установим значение надписи, задав в свойстве Caption текст, выберем значок (свойство FaceId) и стиль кнопки (отображение и значка, и надписи).

Создание новой кнопки

```
procedure TOKBottomDlg8.Button2Click(Sender: TObject);
begin
  msButton:=CommandBars.Item['standard'].Controls.Add(
            Type:=msoControlButton, ID:=1);
  msButton.Caption:='Простая кнопка';
  msButton.Style:= msoButtonIconAndCaption;
  msButton.FaceId:=FaceId.value;
end;
```

[1] Значения идентификатора объекта (кроме 1) определяют кнопки, связанные с определенными командами Word. Значение 1 определяет кнопку, не связанную с командой Word (т. е. пользовательскую кнопку).

Результат выполнения процедуры представлен на рис. 10.13.

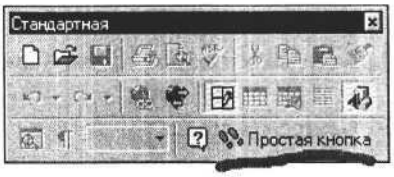


Рис. 10.13. Создаем новую кнопку на панели **Стандартная**

Изменим стиль кнопки (отображение только значка или надписи), записав в свойство msButton.Style константу msoButtonIcon или msoButtonCaption. Соответственно получаем результат, представленный на рис. 10.14 и 10.15.

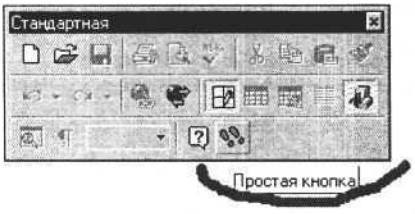


Рис. 10.14. На кнопке только значок

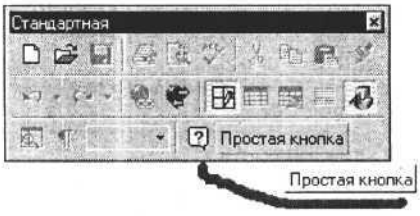


Рис. 10.15. На кнопке только надпись

При создании пользовательского элемента (кнопки) мы присваивали аргументу ID значение, равное единице. Если данному аргументу присвоить другое значение, то получим функциональную кнопку со своими значком и командой, выполняемой при обращении к элементу управления. Для примера можем использовать значения от 2 до 9 или от 1850 до 1859.

Создание элементов управления с предопределенными свойствами

```
procedure TOKBottomDlg8.Button1Click(Sender: TObject);
 var msbootom:variant;
  a_:integer;
begin
for a_:=1850 to 1859 do begin
msbootom:=CommandBars.Item[CheckListBox1.ItemIndex+1].Controls.Add(Type:=
            msoControlButton, ID:=inttostr(a_));
msbootom.Caption:='ID='+inttostr(a_);
end;
end;
```

Значок кнопки определяется величиной числовой константы, записанной в свойство FaceId элемента управления. В качестве примера создадим кнопки со значками для значений этого свойства от 0 до 500.

Создание кнопок с различными значками

```
procedure TForm1.Button13Click(Sender: TObject);
 var panel, buttom:variant;
    a_:integer;
begin
  panel:=W.CommandBars.Add('Новая панель');
  panel.Visible:=True;
  for a_:= 0 to 500 do begin
    buttom:=panel.Controls.Add(Type:=msoControlButton, ID:=1);
    buttom.Style:=msoButtonIconAndCaption;
    buttom.Caption:=inttostr(a_);
    buttom.FaceId:=a_;
  end;
end;
```

Кнопки созданной панели показаны на рис. 10.16.

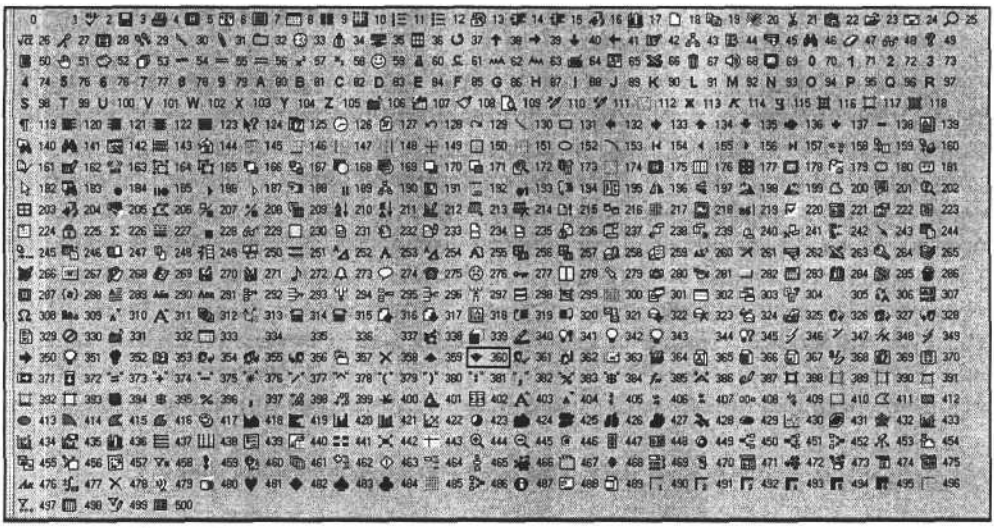


Рис. 10.16. Варианты значков кнопок

Создание и использование макроса Visual Basic средствами Delphi

Возможно, это самый интересный раздел. Как уже говорилось и было показано на многочисленных примерах, используя среду визуального программирования приложений Delphi, можно без особого труда создать и исполь-

зовать любой документ приложения Word. Но есть некоторые ограничения, которые порой достаточно трудно обойти. Одним из таких ограничений является невозможность передавать из Delphi в Visual Basic значения, типы которых не поддерживаются для такого взаимодействия. Например, в документе требуется создать объект — ломаную линию. Для создания такого объекта приходится заполнять массив данных для точек изломов этой линии, затем вызывать метод, аргументом которого является этот массив. При создании контроллера автоматизации мы не можем использовать вызов из Delphi функций Visual Basic, аргументами которых являются массивы. Как быть в таком случае или в других случаях, когда ситуация кажется безвыходной? Решением может быть создание макроса средствами Delphi с использованием объектов Visual Basic и выполнение этого макроса, а затем его удаление после выполнения задачи.

Макросы представляют собой тексты процедур или функций на языке Visual Basic, связанные с проектами. *Проекты* являются родительскими объектами по отношению к макросам. В свою очередь проект связан или с документом, или с шаблоном документа. Все проекты объединены в коллекцию проектов. Текст макроса хранится в программном модуле, в котором могут храниться и несколько макросов.

Доступ к коллекции или к любому проекту коллекции обеспечивают объекты W.VBA.VBProjects и W.VBE.VBProjects.Item(i:integer), где i — индекс элемента коллекции. Имя элемента коллекции связано с именем шаблона или документа, обычно для шаблона Normal индекс равен единице. Проекту, в свою очередь, принадлежит коллекция VBComponents элементов проекта — это и визуальные объекты, и тексты макросов. Метод Add коллекции VBComponents создает новый элемент. Чтобы создать стандартный модуль, содержащий тексты макросов, аргумент этого метода должен быть равен vbext_ct_StdModule. После создания модуля мы получаем доступ к текстовой части макроса и можем записать в него любые операторы, соответствующие спецификации языка Visual Basic. Рассмотрим пример процедуры.

Создание и использование макроса Visual Basic средствами Delphi

```
procedure TForm1.Button5Click(Sender: TObject);
 const vbext_ct_StdModule=1;
 var
 CodeModule:variant;
 CommandBar:variant;
 msbootom:variant;
 eee_:string;
begin
CodeModule:=
W.VBE.VBProjects.Item(1).VBComponents.Add(vbext_ct_StdModule).CodeModule;
```

```
eee_:='Sub Макрос1()'+Chr(10)+Chr(13)+
'MsgBox '+Chr(34)+'Выполняется процедура нажатия кнопки'+Chr(34)+
  Chr(10)+Chr(13)+
'set document=Application.Documents.Add '+Chr(10)+Chr(13)+
'Dim pts(1 To 7, 1 To 2) As Single '+Chr(10)+Chr(13)+
'pts(1, 1) = 0 '+Chr(10)+Chr(13)+
'pts(1, 2) = 0 '+Chr(10)+Chr(13)+
'pts(2, 1) = 72 '+Chr(10)+Chr(13)+
'pts(2, 2) = 72 '+Chr(10)+Chr(13)+
'pts(3, 1) = 100 '+Chr(10)+Chr(13)+
'pts(3, 2) = 40 '+Chr(10)+Chr(13)+
'pts(4, 1) = 20 '+Chr(10)+Chr(13)+
'pts(4, 2) = 50 '+Chr(10)+Chr(13)+
'pts(5, 1) = 90 '+Chr(10)+Chr(13)+
'pts(5, 2) = 120 '+Chr(10)+Chr(13)+
'pts(6, 1) = 60 '+Chr(10)+Chr(13)+
'pts(6, 2) = 30 '+Chr(10)+Chr(13)+
'pts(7, 1) = 150 '+Chr(10)+Chr(13)+
'pts(7, 2) = 90 '+Chr(10)+Chr(13)+
'ActiveDocument.Shapes.AddPolyline (pts) '+Chr(10)+Chr(13)+
  'End Sub';
CodeModule.AddFromString(eee_);
eee_:=CodeModule.name;
messagebox(handle,pchar('Макрос "Макрос1" создан в модуле "'+
                        eee_+'"'),'',0);
CommandBar:=W.CommandBars.Add(Name:='Временная панель',
                              Position:=msoBarFloating);
CommandBar.Enabled:=true;
CommandBar.Visible:=true;
messagebox(handle,pchar('Создана временная панель'),'',0);
MSBootom:=W.CommandBars.Item['Временная панель'].Controls.Add(Type:=
  msoControlButton, ID:=1);
MSBootom.Caption:='Временная кнопка';
MSBootom.OnAction:=CodeModule.name+'.Макрос1';
messagebox(handle,pchar('Создана временная кнопка'),'',0);
msbootom.Execute;
MSBootom.Delete;
messagebox(handle,pchar('Удалена временная кнопка'),'',0);
CommandBar.Delete;
messagebox(handle,pchar('Удалена временная панель'),'',0);
W.VBE.VBProjects.Item(1).VBComponents.Remove(CodeModule.parent);
messagebox(handle,pchar('Модуль "'+eee_+'" и "Макрос1" удалены'),'',0);
end;
```

В первом операторе этой процедуры мы создаем и получаем доступ к программному модулю, в который сможем записать текст создаваемого макроса. Далее в строковую переменную записываем текст создаваемого макроса, учитывая стиль написания операторов Visual Basic. Оператор

```
CodeModule.AddFromString(eee_);
```

где eee_ — строка, содержащая текст макроса, записывает текст макроса в модуль. Далее создаем меню и кнопку, в свойство OnAction которой записываем строку — имена модуля и макроса, разделенные точкой. Метод Execute кнопки запускает созданный макрос на выполнение. Далее удаляем кнопку, панель и компонент, содержащий программный модуль. Если эту процедуру выполнять по шагам, то сможем наблюдать следующее. На первом этапе мы видим в окне редактора Visual Basic, как создается программный модуль и в него записывается текст макроса (рис. 10.17).

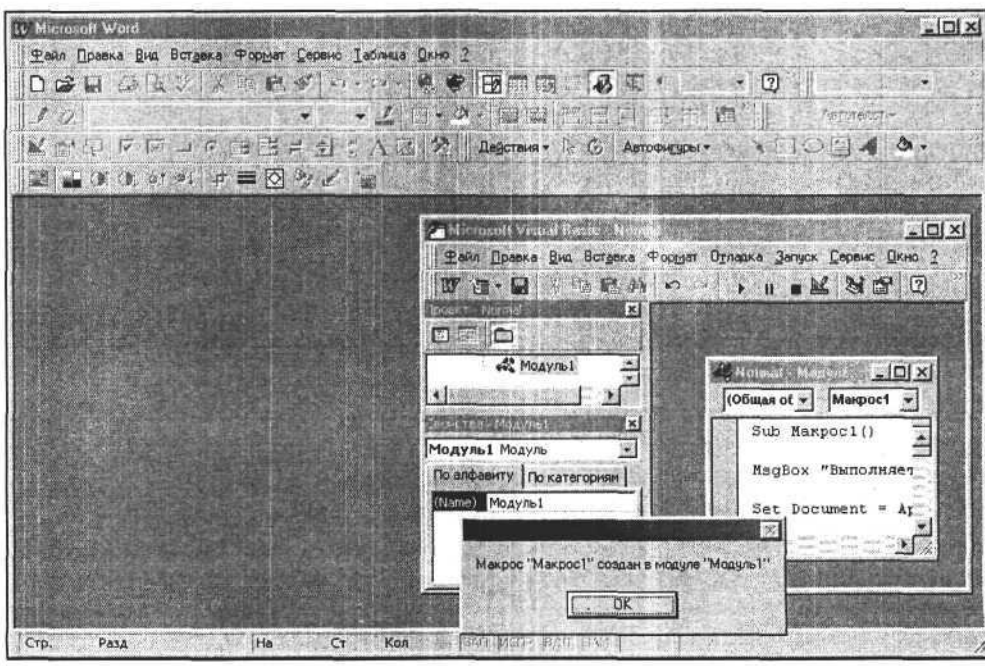


Рис. 10.17. Создан модуль и записан текст макроса

Далее, вызывается метод Execute кнопки, с которой связан макрос, и выполняется программа на Visual Basic, записанная в программном модуле (рис. 10.18).

После выполнения удаляются созданные элементы управления и программный модуль с текстом подпрограммы на Visual Basic (рис. 10.19).

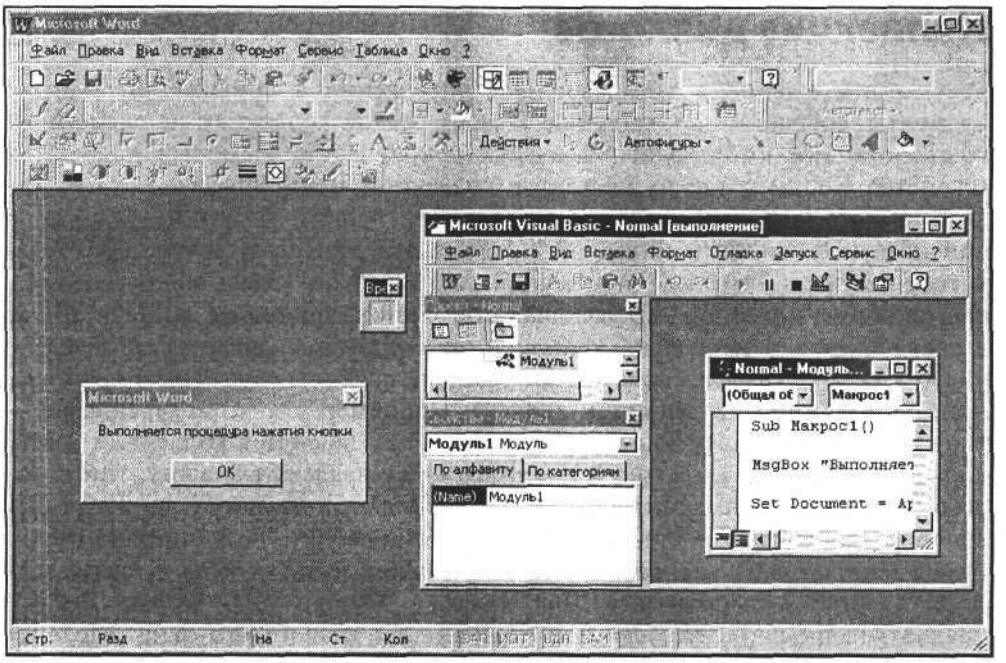


Рис. 10.18. Макрос запущен на выполнение

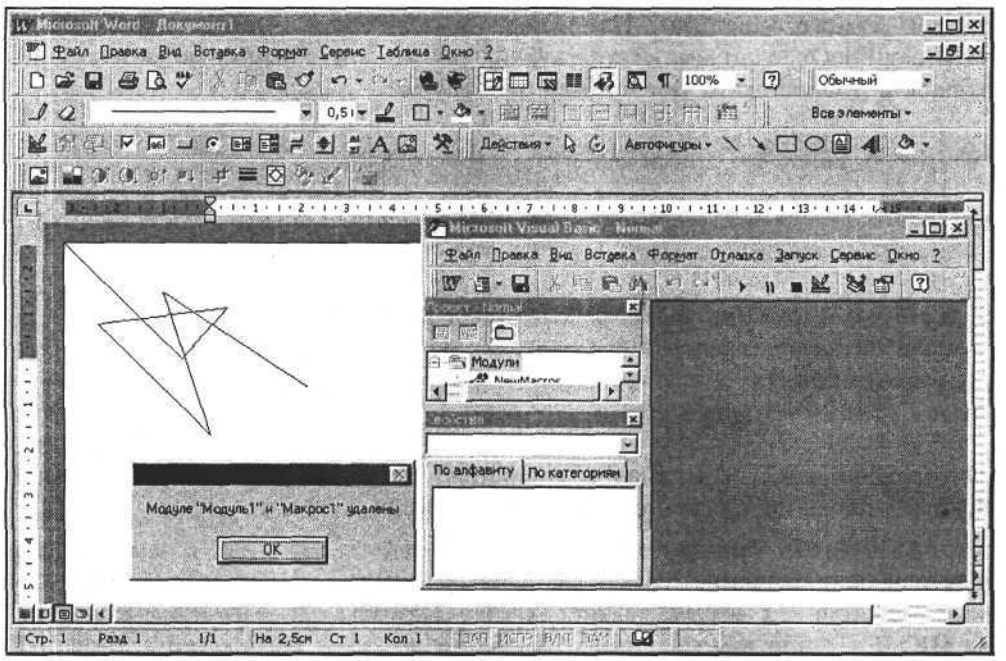


Рис. 10.19. Макрос выполнен, элементы управления и программный модуль удалены

Коллекция диалогов

Рассмотрим еще один тип элементов управления в приложении Word — элементы, собранные в коллекцию Dialogs. У этой коллекции не очень много свойств. Нас интересуют два из них — количество элементов коллекции (свойство Count) и набор элементов коллекции Item(i:integer), где i — индекс диалога (диалогового окна) в коллекции. Чтобы ознакомиться со всем списком и значениями индексов и аргументов диалогов, достаточно обратиться к нужной странице справочной системы Word. Здесь мы рассмотрим только общие принципы вызова диалогов Word из программ, разработанных в среде Delphi, и несколько примеров использования диалогов. У каждого диалога есть методы и свойства. Есть общие свойства диалогов и свойства, присущие каждому отдельному диалогу. Например: свойство type:integer — тип диалога (это свойство совпадает с индексом диалога в коллекции и есть у каждого диалога); свойство Name:string — принадлежит диалогу, тип которого определяется константой wdDialogFileSaveAs, и не может принадлежать диалогу типа wdDialogConnect. Это нетрудно проверить — достаточно попытаться использовать свойство Name для диалога второго типа — сразу возникнет ошибка выполнения. Для каждого диалога есть четыре метода, предназначенных для запуска или отображения диалогового окна. Рассмотрим их.

Метод Execute выполняет действия, связанные с диалогом, но без отображения самого диалога. Например, вызвав этот метод для диалога wdDialogFileOpen, можно открыть файл так же, как с помощью метода Open коллекции Documents. Вот пример процедуры.

Открытие файла без отображения диалогового окна

```
procedure TForm1.Button11Click(Sender: TObject);
 var msdialog:variant;
begin
  msdialog:=W.Dialogs.Item(wdDialogFileOpen);
  msdialog.Name:=inputbox('Введите имя файла','','*.doc');
  msdialog.Execute;
end;
```

Метод Show отображает диалог, выполняет заданные в нем действия и возвращает значение, связанное с нажатием кнопок диалога.

Метод Display отображает диалог, но не выполняет заданные в нем действия и возвращает значение, связанное с нажатием кнопок диалога. Возвращаемые значения: −2 — была нажата кнопка **Закрыть**; −1 — была нажата кнопка **OK**; 0 — была нажата кнопка **Пропустить**; > 0 — была нажата кнопка 1, 2... диалога.

Метод Update вызывается для гарантированной установки значений элементов диалога. Перед вызовом диалога или по окончании вызова мы можем установить значения некоторых визуальных элементов диалога или получить их значения, введенные или измененные пользователем. В качестве примера рассмотрим следующую процедуру и результат ее выполнения. Данный пример можно использовать, когда требуется задать в качестве имени файла определенное значение, отличное от значения по умолчанию.

Пример изменения значения элемента диалога до его выполнения

```
procedure TForm1.Button15Click(Sender: TObject);
 var myDialog_:variant;
begin
  myDialog_:=W.Dialogs.Item(wdDialogFileSaveAs);
  myDialog_.Name:='Подставляем свое имя файла';
  myDialog_.Display;
end;
```

Результат выполнения данной процедуры представлен на рис. 10.20.

Из уже пройденных материалов нетрудно сделать вывод, что при умелом использовании можно с легкостью превратить приложение Word в инструмент для подготовки различных отчетных форм и использовать это в ваших приложениях, разрабатываемых в среде Delphi. Положительной особенностью здесь является тот факт, что при переходе от одной версии MS Word к другой не требуется вносить в разрабатываемые приложения какие-либо изменения, потому что новые версии MS Office поддерживают работу функций, используемых в предыдущих версиях. Для практики программирования панели в приложении Word немного изменим рассмотренный ранее пример.

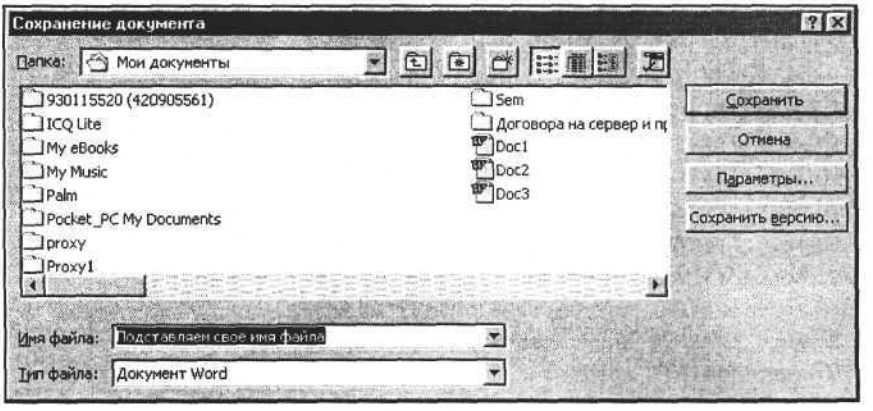


Рис. 10.20. Управление диалоговым окном

Пример программирования панели

Изменим процедуру формирования платежного поручения, рассмотренную ранее. В данном примере мы дополнительно отключим все стандартные панели Word, блокируем доступ пользователя к изменению содержания документа и создадим пользовательскую панель, на которой разместим кнопки сохранения документа, выполнения печати и изменения масштаба отображения документа в окне Word.

Процедура формирования документа "Платежное поручение"

```
procedure TForm1.Button14Click(Sender: TObject);
   var a_:integer;
 CommandBar_:variant;
  msbootom_:variant;
begin
// Отключаем панели
for a_:=1 to W.CommandBars.Count do
      W.CommandBars.Item[a_].Enabled:=false;
// Создаем новый документ на основе шаблона
W.documents.Add(ExtractFileDir(Application.ExeName)+
              '\Шаблон платежного поручения.dot');
// Подставляем текст
FindAndInsert('###№ П.П.&','1');
FindAndInsert('###Дата&',datetostr(date));
FindAndInsert('###Вид платежа&','почтой');
FindAndInsert('###Сумма прописью&','Двести пятьдесят рублей сорок '+
       'копеек');
FindAndInsert('###Сумма&','250,40');
FindAndInsert('###ИНН плательщика&','0000000000');
FindAndInsert('###КПП плательщика&','000000000011');
FindAndInsert('###Плательщик&','ЗАО Селена');
FindAndInsert('###Р/С плательщика&','00000000000000000000');
FindAndInsert('###БИК плательщика&','000000');
FindAndInsert('###К/С плательщика&','00000000000000000000');
FindAndInsert('###ИНН получателя&','1111111111');
FindAndInsert('###КПП получателя&','111111111100');
FindAndInsert('###БИК получателя&','111111');
FindAndInsert('###К/С получателя&','11111111111111111111');
FindAndInsert('###Р/С получателя&','11111111111111111111');
FindAndInsert('###Получатель&','ЗАО Комета');
FindAndInsert('###В.О.&','');
```

```
FindAndInsert('###Н.П.&','');
FindAndInsert('###Код&','');
FindAndInsert('###С.П.&','');
FindAndInsert('###О.П.&','');
FindAndInsert('###Р.П.&','');
FindAndInsert('#H1&','');
FindAndInsert('#H2&','');
FindAndInsert('#H3&','');
FindAndInsert('#H4&','');
FindAndInsert('#H5&','');
FindAndInsert('#H6&','');
FindAndInsert('#H7&','');
FindAndInsert('###Назначение платежа& ','Оплата за поставку товара');
// Блокируем возможность изменения документа пользователем
W.ActiveDocument.Protect(Password:='',NoReset:=False,
            Type:=wdAllowOnlyFormFields);
// Создаем временную неперемещаемую панель
CommandBar_:=W.CommandBars.Add('Печать документа',msoBarTop,
                False, True);
CommandBar_.Enabled:=true;
CommandBar_.Visible:=true;
CommandBar_.Protection:=msoBarNoMove;
// Создаем кнопки на панели
for a_:=3 to 7 do begin
  try
  msbootom_:=CommandBar_.Controls.Add(Type:=msoControlButton,
                    ID:=inttostr(a_));
  msbootom_.Caption:='ID='+inttostr(a_);
  except
end;
end;
end;
```

Результат выполнения процедуры представлен на рис. 10.21. После печати и закрытия документа необходимо восстановить прежний вид панелей. Это нетрудно сделать, используя методы программирования панелей и макросов.

Мы рассмотрели возможности объекта Word.Application, обеспечивающие формирование документов Word. Конечно, это далеко не все его возможности, но даже они позволяют создавать документ Word посредством приложений, разработанных в среде Delphi. Далее рассмотрены аналогичные возможности использования объекта Excel.Application в приложениях Delphi.

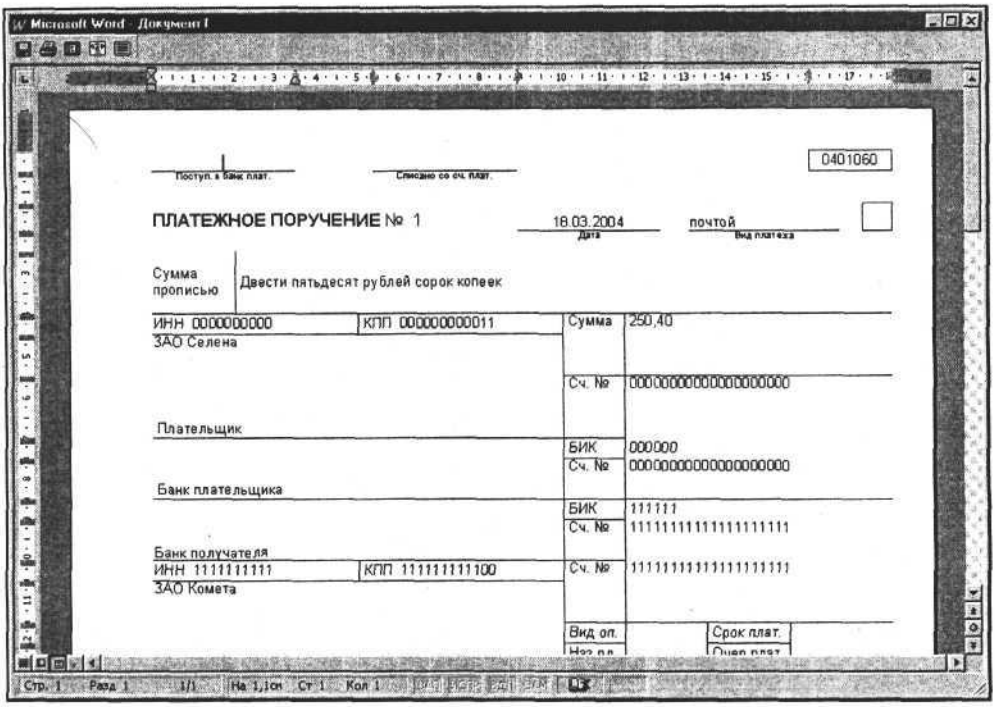
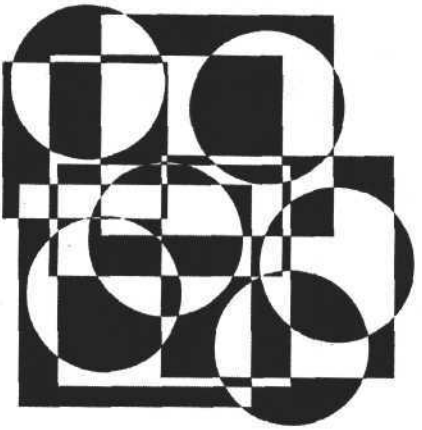


Рис. 10.21. Пользовательская настройка приложения Word для печати документа



ЧАСТЬ III

Разработка документов и приложений MS Excel в Delphi

ГЛАВА 11

Работа с объектом Excel.Application

В первой части книги мы рассмотрели в общих чертах объектную модель приложения MS Excel. В первую очередь эта модель интересует нас как средство работы с рабочими книгами Excel, которые также представляют собой объекты и являются составной частью объектной модели приложения. Как было показано в первой части книги, доступ к объектам MS Excel, к открытым книгам и всем компонентам любой книги производится через корневой объект Application. Доступ к этому объекту в приложениях Delphi можно получить с помощью функции CreateOleObject, которая возвращает ссылку на объект Application, а единственным ее аргументом является строковый идентификатор 'Excel.Application'.

В этой главе рассмотрены следующие темы:

- создание объекта Excel.Application, запуск и визуализация окна приложения;
- создание рабочей книги;
- создание рабочей книги на основе шаблона;
- открытие существующей рабочей книги;
- доступ к рабочей книге;
- сохранение рабочей книги;
- настройка окон рабочей книги;
- работа с листами рабочей книги;
- чтение и запись информации в ячейки листа рабочей книги.

Создание объекта Excel.Application, запуск и визуализация окна приложения

Запуск приложения Excel производится непосредственно при вызове функции CreateOleObject, входящей в состав стандартной библиотеки ComObj.pas.

Функция возвращает ссылку на объект, представляющий собой переменную типа variant. Рассмотрим следующий программный код.

Создание объекта Excel.Application

```
uses ComObj;
E:variant;
procedure TForm1.Button1Click(Sender: TObject);
begin
  E:=CreateOleObject('Excel.Application');
end;
```

Результатом выполнения данной процедуры будет запуск приложения Excel на выполнение. Вы можете обнаружить это, просмотрев список задач, запущенных на выполнение в операционной системе. Но для того чтобы убедиться, что приложение запущено, можно и не обращаться к списку выполняющихся задач. Можно просто сделать объект Excel видимым. Для этого используем свойство Visible объекта Application. Если это свойство установить в значение True, то окно приложения Excel тут же отобразится на экране. Манипулируя со свойством Visible, мы можем скрывать окно приложения или отображать его на экране.

Примечание

Для ускорения формирования готового документа лучше, чтобы приложение было запущено в фоновом режиме, т. е. его окно не должно отображаться. Напротив, если требуется отследить ход формирования рабочей книги по шагам, то необходимо включить режим отображения (при Visible=True).

Рассмотрим следующую процедуру.

Отображение окна приложения Excel

```
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  E.Visible:=CheckBox1.Checked;
end;
```

На рис. 11.1 представлено окно Excel, отображенное на экране монитора, как результат выполнения приведенной процедуры.

Как видно из рис. 11.1, вновь созданный и запущенный экземпляр приложения Excel не содержит ни одной рабочей книги. Все рабочие книги, которые в данный момент могут быть активны или принадлежать объекту Application (переменная E), являются принадлежностью коллекции WorkBooks, которая в свою очередь принадлежит корневому объекту. Свойство Count:integer коллекции WorkBooks содержит количество открытых рабочих книг (в нашем случае Count=0). Если мы создаем новую рабочую или

открываем ранее сохраненную книгу, то значение Count увеличивается, если закрываем, то уменьшается каждый раз на единицу.

Далее мы рассмотрим некоторые свойства и методы этой коллекции.

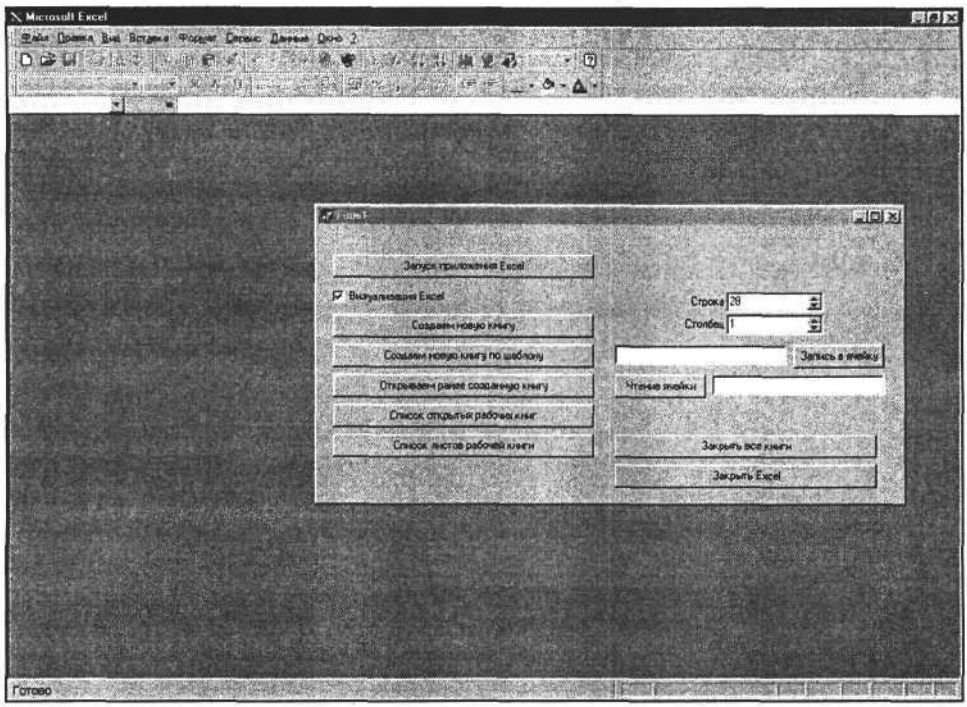


Рис. 11.1. Отображение окна приложения Excel на экране задается флажком **Визуализация Excel**

Создание рабочей книги

Метод Add коллекции WorkBooks позволяет создать новую рабочую книгу. При этом если аргументом метода будет строка, указывающая на файл шаблона, то новая книга будет создана на основе этого шаблона. Если аргументов нет, то будет создана обычная книга в режиме "по умолчанию". Создадим новую книгу в режиме по умолчанию с помощью следующей процедуры.

Создание рабочей книги по умолчанию

```
procedure TForm1.Button4Click(Sender: TObject);
begin
  E.WorkBooks.Add;
end;
```

Результат выполнения данной процедуры представлен на рис. 11.2. Создана обычная рабочая книга.

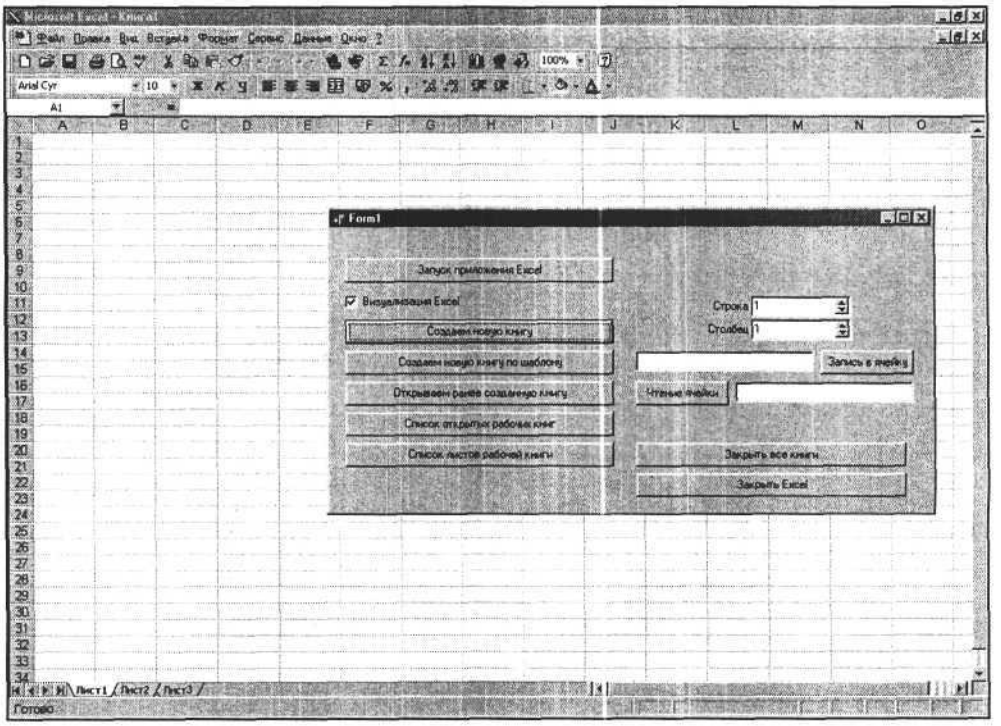


Рис. 11.2. Создаем рабочую книгу "по умолчанию"

После того как мы создали новую рабочую книгу, значение свойства Count коллекции Workbooks увеличилось на единицу, а объект Item(1) содержит ссылку на созданную рабочую книгу.

Создание рабочей книги на основе шаблона

Создадим рабочую книгу по шаблону с помощью метода Add, в качестве аргумента которого используем или целочисленную константу, или строку, указывающую на файл шаблона. В качестве примера рассмотрим следующие процедуры, созданные в среде Delphi. В первом случае аргументом метода Add служит константа, в результате чего создается лист с диаграммой, во втором случае в качестве шаблона используем один из стандартных шаблонов MS Office.

Создание рабочей книги на основе шаблона

```
procedure TForm1.Button5Click(Sender: TObject);
 const xlWBATChart=-4109;
begin
  E.WorkBooks.add(xlWBATChart);
end;

procedure TForm1.Button5Click(Sender: TObject);
begin
  if not OpenDialog1.Execute then exit;
  E.WorkBooks.add(OpenDialog1.FileName);
end;
```

При выполнении второй процедуры нужно выбрать в специальном окне (рис. 11.3) файл с расширением XLT из стандартной поставки MS Office.

Новая рабочая книга будет создана на основе выбранного шаблона (рис. 11.4).

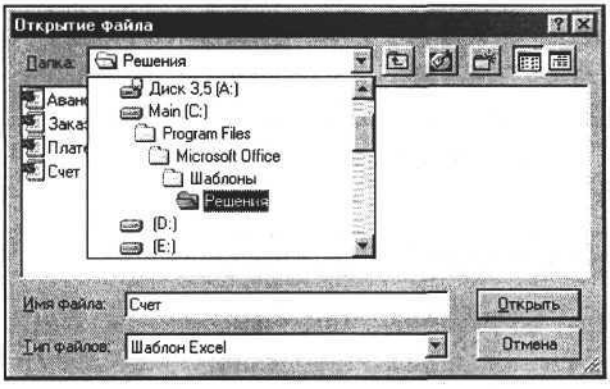


Рис. 11.3. Выбираем шаблон для создания книги

Открытие существующей рабочей книги Excel

Использование методов коллекции WorkBooks позволяет не только создавать, но и открывать имеющиеся рабочие книги, которые хранятся в файлах. Для этих целей предназначены методы Open и OpenText. Первый метод открывает файл формата XLS, а второй — обычные текстовые файлы. Для простоты используем метод Open с одним аргументом — строкой-указателем на файл. Вот текст процедуры Delphi, использующей метод Open.

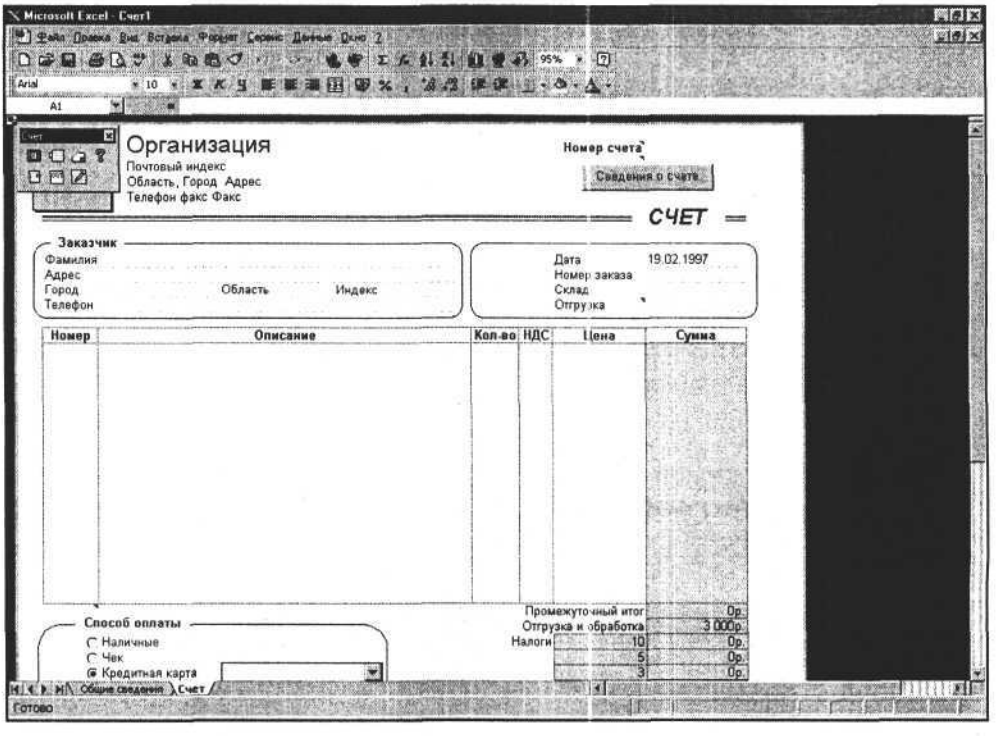


Рис. 11.4. Рабочая книга, созданная по шаблону

Открытие существующей рабочей книги Excel

```
procedure TForm1.Button7Click(Sender: TObject);
begin
  if not OpenDialog2.Execute then exit;
  E.WorkBooks.Open(OpenDialog2.FileName);
end;
```

Полная спецификация вызова метода Open представлена в справочной системе MS Excel и имеет следующий вид:

```
Open(FileName, UpdateLinks, ReadOnly, Format, Password, WriteResPassword,
  IgnoreReadOnlyRecommended, Origin, Delimiter, Editable, Notify,
  Converter, AddToMRU);
```

Обязательным является первый аргумент. Для задания специфических режимов открытия файла можно использовать и другие аргументы. Их список и краткое назначение приведены в табл. 11.1.

Таблица 11.1. Аргументы метода Open

Имя	Тип	Назначение
FileName	String	Имя файла
UpdateLinks	Integer	Режим обновления ссылок в рабочей книге
ReadOnly	Boolean	True — открыть в режиме "только для чтения"
Format	Integer	Формат открытия текстовых файлов
Password	String	Пароль, если книга была сохранена с установкой пароля
WriteResPassword	String	Пароль, необходимый для сохранения изменений, если он был задан ранее
IgnoreReadOnlyRecommended	Boolean	True — отключение сообщения о том, что файл открывается в режиме "только для чтения", если он был сохранен в режиме "только для чтения"
Origin	Integer	Кодировка для открываемого текстового файла
Delimiter	Integer	Код символа-разделителя ячеек таблицы (при открытии текстового файла)
Editable	Boolean	Дополнительный режим при открытии Excel файлов более ранних версий, чем версия 5.0
Notify	Boolean	Если была попытка открыть файл в режиме чтение/запись, но в этот момент времени это было невозможно, то при значении True этого аргумента приложение получит уведомление, когда файл станет доступен. Если значение аргумента False или опущено, и файл занят, то попытки открыть его для чтения/записи обречены на неудачу
Converter	Integer	Индекс конвертора, используемого при открытии файла
AddToMRU	Boolean	True — имя открываемого файла добавляется в список недавно открытых файлов меню **Файл**

При открытии рабочей книги можно использовать любую комбинацию аргументов метода Open. Например, чтобы открыть файл в режиме "только для чтения", изменим представленный выше пример процедуры и представим его в следующем виде.

Открытие рабочей книги в режиме "только для чтения"

```
procedure TForm1.Button7Click(Sender: TObject);
begin
  if not OpenDialog2.Execute then exit;
  E.WorkBooks.Open(FileName:= OpenDialog2.FileName, ReadOnly:=True);
end;
```

Доступ к рабочей книге

После того как мы открыли и создали несколько рабочих книг в приложении Excel, можно перейти к анализу содержимого коллекции WorkBooks. Объекты Item(i:integer) содержат ссылки на все рабочие книги коллекции WorkBooks (i:integer — индекс книги в коллекции). В качестве аргумента при обращении к Item может выступать и строковая переменная, содержащая имя книги. Свойство Count коллекции содержит количество открытых документов коллекции. Используя эти свойства коллекции, мы можем вывести список всех рабочих книг и перейти к работе с любой из них. Для этого рассмотрим следующие процедуры.

Получение списка рабочих книг и ссылки на выбранную рабочую книгу

```
procedure TOKBottomDlg2.FormCreate(Sender: TObject);
 var a_:integer;
begin
  WorkBooks:=Form1.E.WorkBooks;
  for a_:=1 to WorkBooks.count do begin
     ListBox1.Items.Add(WorkBooks.Item[a_].name+
       '; '+Workbooks.Item[a_].FullName);
  end;
end;

procedure TOKBottomDlg2.ListBox1Click(Sender: TObject);
begin
  WorkBooks.item[ListBox1.ItemIndex+1].Activate;
  WorkBook:=WorkBooks.item[ListBox1.ItemIndex+1];
end;
```

С помощью этих процедур мы можем вывести список всех рабочих книг и отобразить их имена в объекте ListBox (список). Затем, перебирая содержимое списка, активируем любую из открытых рабочих книг и получаем ссылку на нее для последующего доступа к этой рабочей книги (рис. 11.5).

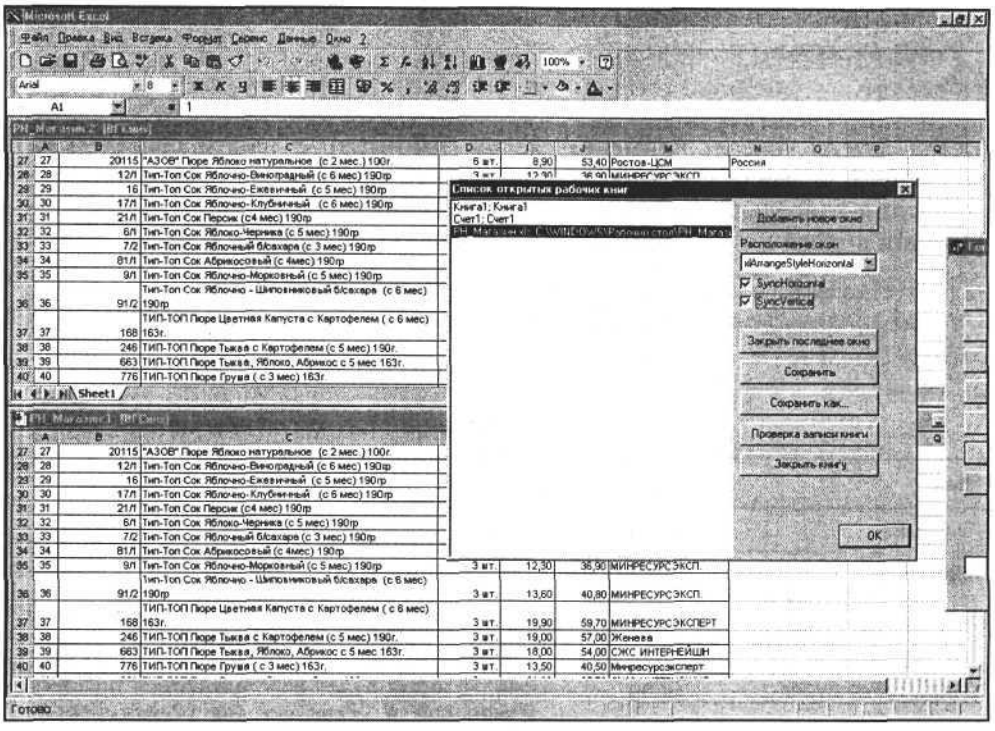


Рис. 11.5. Выбираем рабочую книгу в списке

Для активизации рабочей книги из списка открытых используем метод Activate объекта Item(i:integer), где i — индекс открытой рабочей книги, а объект Item(i:integer) представляет собой ссылку на рабочую книгу.

Сохранение рабочей книги

После того как рабочая книга выбрана, мы можем ее редактировать, сохранить под прежним или другим именем и закрыть. Для этого используем методы Save, SaveAs и Close.

Сохранение рабочей книги

```
procedure TOKBottomDlg2.Button2Click(Sender: TObject);
begin
  WorkBook.Save;
end;

procedure TOKBottomDlg2.Button3Click(Sender: TObject);
begin
```

```
  if not SaveDialog1.Execute then exit;
  WorkBook.SaveAs(SaveDialog1.FileName);
end;
```

Метод SaveAs должен применяться, как минимум, с одним обязательным аргументом — именем файла, но может быть применен с несколькими аргументами, определяющими режим сохранения файла (табл. 11.2). Спецификация вызова метода приведена в справке по Visual Basic и имеет вид:

```
SaveAs(Filename, FileFormat, Password, WriteResPassword,
  ReadOnlyRecommended, CreateBackup, AddToMru, TextCodePage,
  TextVisualLayout);
```

Таблица 11.2. *Аргументы метода SaveAs*

Имя	Тип	Назначение
FileName	String	Имя файла
FileFormat	Integer	Формат сохраняемого файла
Password	String	Пароль, который будет использоваться для открытия сохраненной рабочей книги
WriteResPassword	String	Пароль, который будет использоваться для записи изменений в открытую для редактирования рабочую книгу
ReadOnlyRecommended	Boolean	True — при открытии книги отображается сообщение о том, что файл может быть открыт только в режиме "для чтения"
CreateBackup	Boolean	True — создать резервный файл
AccessMode	Integer	Режим доступа к файлу
ConflictResolution	Integer	Реакция приложения на конфликт при записи книги
AddToMRU	Boolean	True — добавить имя записываемого файла в список недавно открытых файлов меню **Файл**
TextCodePage		Не используется
TextVisualLayout		Не используется

После того как рабочая книга сохранена, можно закрыть ее, для этого используем метод Close объекта WorkBook. Для того чтобы закрыть сразу все открытые книги, используем метод Close коллекции WorkBooks. Перед тем как закрыть рабочую книгу, нелишне проверить — сохранена она или нет. Для этого используем свойство Saved объекта WorkBook. Значение Saved=True означает, что со времени последнего сохранения рабочая книга

не была изменена, т. е. сохранять ее не нужно. В следующем примере мы проверяем, есть или нет изменения в книге, а затем закрываем ее.

Проверка сохранения рабочей книги и ее закрытие

```
procedure TOKBottomDlg2.Button5Click(Sender: TObject);
begin
  if Workbook.Saved
    then messagebox(handle,'Документ '+ 'сохранен!','Внимание!',0)
    else messagebox(handle,'Документ не '+ 'сохранен!','Внимание!',0);
end;

procedure TOKBottomDlg2.Button6Click(Sender: TObject);
begin
  Workbook.Close;
end;
```

Результат выполнения первой процедуры представлен на рис. 11.6.

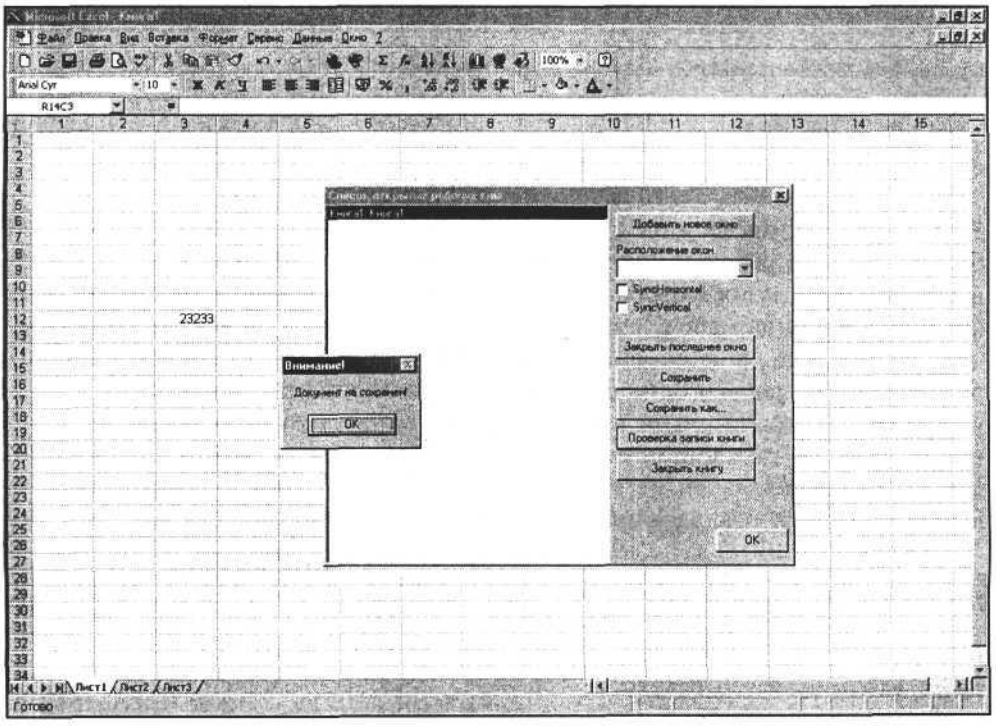


Рис. 11.6. Проверяем факт записи рабочей книги на диск

Мы научились открывать рабочую книгу, сохранять ее и закрывать. Но перед тем, как сохранять, бывает нужно внести некоторые изменения. Это мы и рассмотрим далее.

Настройка окон рабочей книги

Процессор электронных таблиц Excel — замечательный инструмент для табличного представления данных и их обработки. Главной особенностью такой обработки является гибкое применение формул или целых подпрограмм. Причем такой обработке можно подвергнуть как отдельные ячейки, так и целые массивы данных. При этом немаловажным остается удобство ввода и отображения информации для пользователя. Рассмотрим одну особенность рабочих книг MS Excel, которая и служит для этого.

Любую рабочую книгу можно отобразить в виде окна, даже не одного окна, а двух и более окон. Для того чтобы добавить новое окно рабочей книги, используем метод NewWindow объекта WorkBook. Вот процедура, позволяющая добавить новое окно для уже открытой рабочей книги.

Добавление нового окна для открытой рабочей книги

```
procedure TOKBottomDlg2.Button1Click(Sender: TObject);
begin
  WorkBook.NewWindow;
end;
```

После выполнения данной процедуры будет отображено сразу два окна для одной и той же рабочей книги. При этом нетрудно заметить, что изменения в первом окне тут же отображаются во втором, и наоборот. Если открыто более одного окна рабочей книги, то эти окна можно разместить в определенном порядке, используя заданные стили — в виде мозаики, каскадом, расположив в главном окне приложения равномерно по горизонтали или по вертикали. Дополнительно к этому можно синхронизировать перемещения в окнах как по горизонтали или по вертикали, так и в обоих направлениях. Для этого предназначен метод Arrange коллекции Windows. Рассмотрим следующий пример, реализованный в виде процедуры.

Размещение окон по вертикали и их полная синхронизация

```
procedure SetArrangeWindows;
const xlArrangeStyleVertical=-4166;
begin
  with OKBottomDlg2 do begin
```

```
    Workbook.Windows.Arrange(ArrangeStyle:= xlArrangeStyleVertical,
        ActiveWorkbook:=True,SyncHorizontal:=True,SyncVertical:=True);
  end;
end;
```

Результат выполнения процедуры представлен на рис. 11.7.

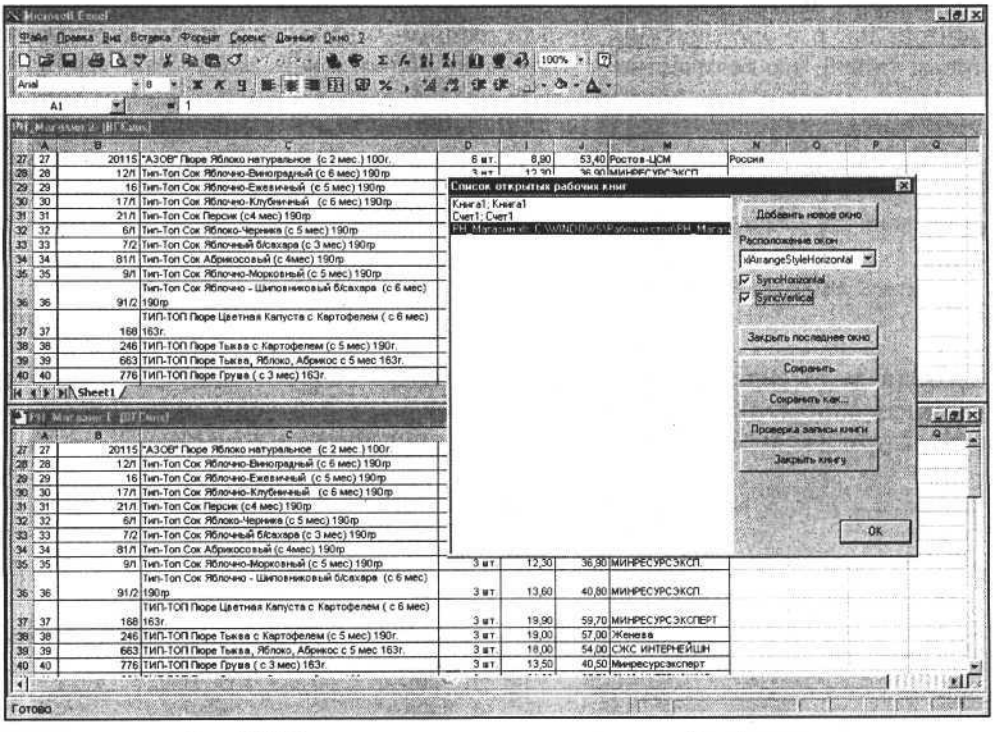


Рис. 11.7. Вертикальное размещение двух окон рабочей книги

В данном примере рабочая книга состоит из одного листа, но в большинстве случаев рабочая книга может содержать гораздо больше листов. Обычно по умолчанию при создании книги она уже содержит три листа. Поэтому для записи или чтения информации с листа необходимо сначала выбрать его. В следующем разделе мы рассмотрим, как это делается.

Работа с листами рабочей книги

Лист, ячейки которого непосредственно хранят информацию, является принадлежностью книги. В рабочей книги может быть больше одного листа. Как получить доступ к списку листов или к любому листу рабочей книги? Ответ: с помощью коллекции Sheets. Как и любая коллекция, она содержит

свойство Count:integer (количество элементов коллекции) и набор объектов Item(i:integer) — собственно листы, где i — индекс выбранного листа (от 1 до Count). Некоторые методы коллекции Sheets: Select — выделение всех листов рабочей книги, Copy — копирование всех листов в новую рабочую книгу, PrintPreview — предварительный просмотр печати, PrintOut — вывод на печать, Add — добавление нового листа в рабочую книгу.

Рассмотрим метод Add подробнее. Его можно использовать как без аргументов, так и с аргументами, определяющими место, куда будут добавлены листы (лист), их количество и тип. Если использовать метод Add так, как показано в следующем примере, то будет добавлен лист перед листом Sheet.

Добавление нового листа перед указанным листом рабочей книги

```
procedure TOKBottomDlg3.Button1Click(Sender: TObject);
begin
  Sheets.Add(Before:=Sheet);
end;
```

Добавив листы или просто открыв или создав рабочую книгу, мы можем получить список листов и доступ к любому листу рабочей книги. Для этого используем уже известные свойства коллекции Sheets.

Получение списка листов рабочей книги

```
procedure TOKBottomDlg3.FormCreate(Sender: TObject);
 var a_:integer;
begin
for a_:=1 to Sheets.count do ListBox1.Items.Add(Sheets.Item[a_].name);
end;
```

Результат выполнения процедуры представлен на рис. 11.8.

После загрузки списка листов мы можем получить доступ к любому листу для работы с ним. Для начала попробуем изменить непосредственно свойства самого листа, например его имя. Имя листа представляет собой строку и содержится в свойстве Name объекта Sheet. Доступ к листу рабочей книги получим, например, с помощью следующей процедуры.

Получение доступа к листу рабочей книги

```
var Sheet:variant;
procedure TOKBottomDlg3.ListBox1Click(Sender: TObject);
begin
  Sheet:=Sheets.item[ListBox1.ItemIndex+1];
end;
```

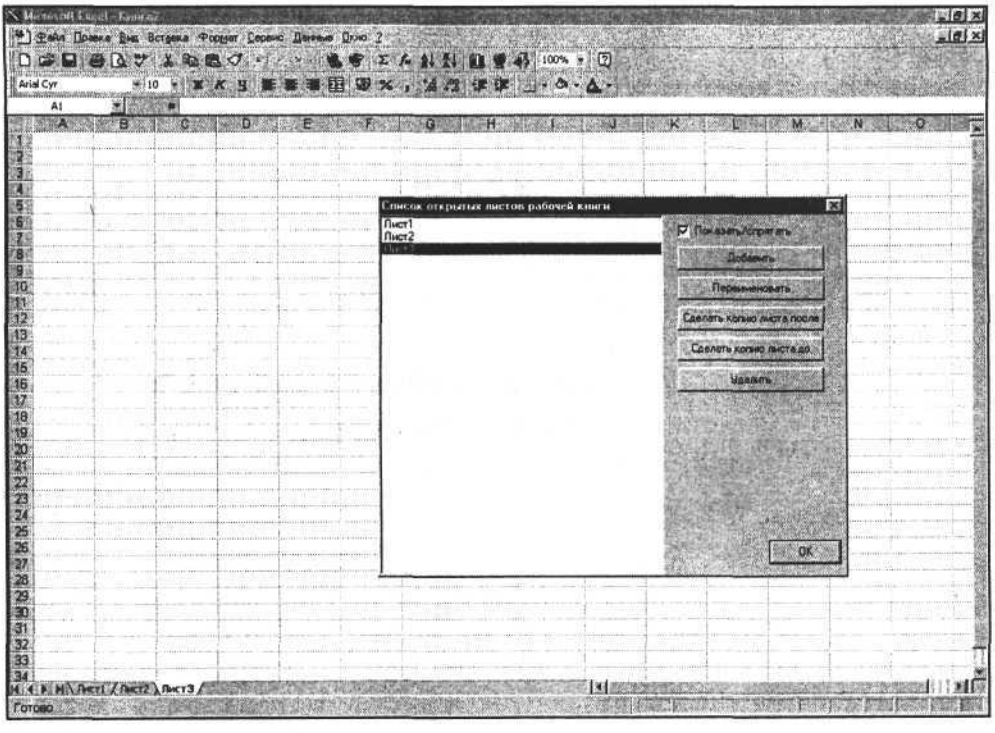


Рис. 11.8. Список листов рабочей книги

Для того чтобы переименовать выбранный рабочий лист, запишем в свойство Name новое значение имени листа, например:

```
Sheet.Name:='Лист3____';
```

Результат такого преобразования представлен на рис. 11.9.

Мы можем скопировать лист и его содержание с помощью метода Copy объекта Sheet. Этот метод позволяет копировать лист и вставлять копию или до или после оригинала — это зависит от значения аргумента метода Copy. Вот пример использования этого метода в процедурах, разработанных в среде Delphi.

Копирование листа рабочей книги и вставка его копии до и после оригинала

```
procedure TOKBottomDlg3.Button5Click(Sender: TObject);
begin
  Sheet.Copy(before:=Sheet);
end;
```

```
procedure TOKBottomDlg3.Button4Click(Sender: TObject);
begin
  Sheet.Copy(after:=Sheet);
end;
```

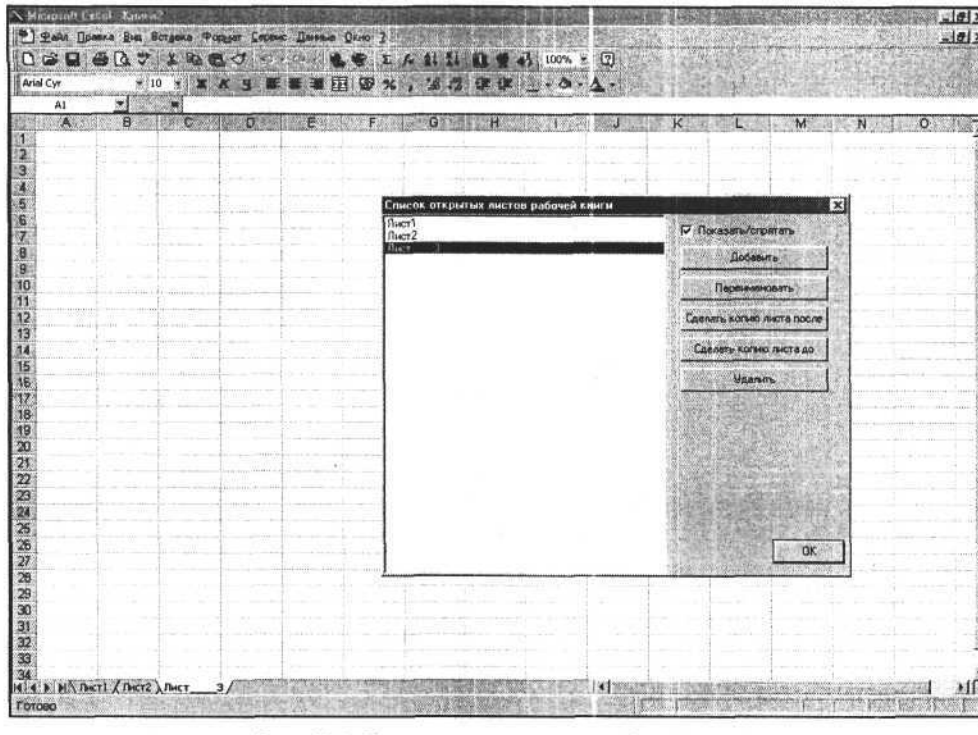


Рис. 11.9. Переименование листа рабочей книги

Результат выполнения этих процедур может быть таким, как показано на рис. 11.10, при этом копируется не только содержание листа, но и все его настройки.

После работы с книгой может понадобиться удалить некоторые листы. Для этого предназначен метод Delete объекта Sheet. Если попытаться с его помощью удалить выбранный лист, то приложение выведет на экран сообщение (рис. 11.11).

Блокировать данное сообщение и другие сообщения приложения позволяет следующий оператор:

```
E.DisplayAlerts:=False;
```

Следующая процедура удаляет лист, подавив предупреждающие сообщения приложения.

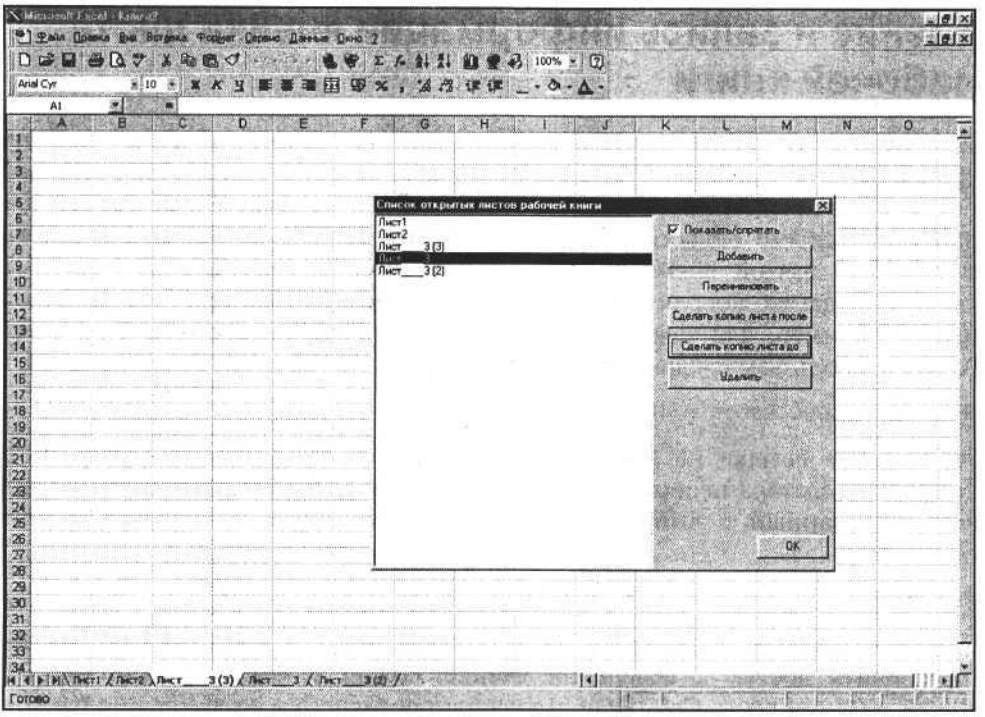


Рис. 11.10. Создание копии листа рабочей книги

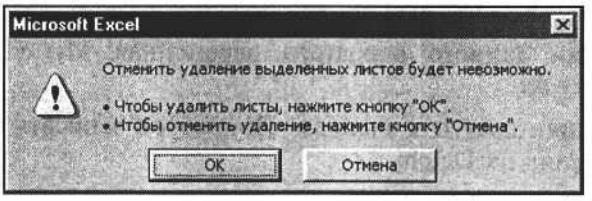


Рис. 11.11. Окно подтверждения удаления листа

Удаление листа рабочей книги

```
procedure TOKBottomDlg3.Button2Click(Sender: TObject);
begin
  E.DisplayAlerts:=False;
  Sheet.Delete;
  E.DisplayAlerts:=True;
end;
```

После того как мы открыли рабочую книгу и выбрали необходимый лист, можно перейти к чтению или записи информации.

Чтение и запись информации ячейки листа рабочей книги

Для доступа к ячейкам можно использовать два разных объекта — объект типа Range, который ассоциируется с областью ячеек, или непосредственно объект Cell (ячейка листа рабочей книги). Если первый объект удобен для работы с целыми областями ячеек, то второй больше подходит для работы с отдельно взятой ячейкой. Эти объекты принадлежат объекту "лист" и требуют задания координат ячейки или области ячеек при обращении к ним. Например, для задания объекта, ассоциированного с областью ячеек A1:D5, используем следующий оператор:

```
MyRange:=Sheet.Range[A1:D5];
```

где Sheet — ссылка на лист рабочей книги. После удачного выполнения данного оператора переменная MyRange:variant содержит ссылку на объект, ассоциированный с выбранной областью ячеек. Другой вариант данного оператора:

```
MyRange:= E.Sheets.Item(1).Range('A1:D5');
```

Здесь мы получаем ссылку на область ячеек первого листа рабочей книги. Обеспечить доступ к отдельной ячейке можно с помощью как объекта Range, так и объекта Cell. Использование последнего в операторе получения доступа к ячейке может выглядеть так:

```
MyCell:=Sheet.Cells[1,1];
```

При выполнении данного оператора переменная MyCell будет хранить ссылку на ячейку A1.

Итак, мы получили доступ к ячейкам. Попробуем записать в них информацию из приложения на Delphi.

Запись информации в ячейки листа рабочей книги

```
procedure TOKBottomDlg4.Button1Click(Sender: TObject);
 var a_:integer;
begin
  randomize;
  for a_:=1 to 100 do Sheet.Cells(a_,1):=random(10000);
end;
```

В данном примере переменная Sheet содержит ссылку на лист рабочей книги (эта переменная была инициализирована ранее). Полный исходный текст приложения находится на сопроводительном диске книги. Результат выполнения процедуры представлен на рис. 11.12.

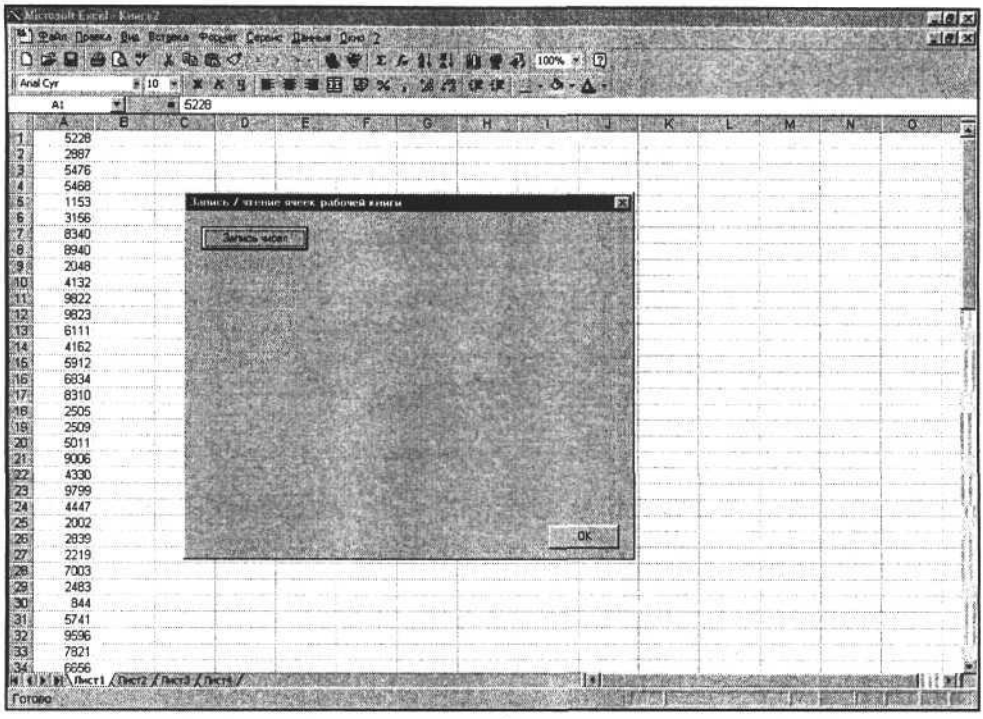


Рис. 11.12. Запись случайных чисел в ячейки листа рабочей книги Excel

Если для записи в ячейки мы присваивали объекту Cells(row, column) значение, то для чтения данных используем оператор, в котором строковой переменной присваивается значение объекта Cells(row, column). В качестве примера импортируем значения ячеек рабочей книги, запишем их значения в объект ListBox1 и отобразим в форме.

Чтение информации листа рабочей книги Excel

```
procedure TOKBottomDlg4.Button2Click(Sender: TObject);
 var eee_:string;
    a_:integer;
begin
ListBox1.Items.Clear;
for a_:=1 to 100 do begin
  eee_:=Sheet.Cells[a_,1];
  ListBox1.Items.Add(eee_);
  end;
end;
```

Результат чтения данных представлен на рис. 11.13.

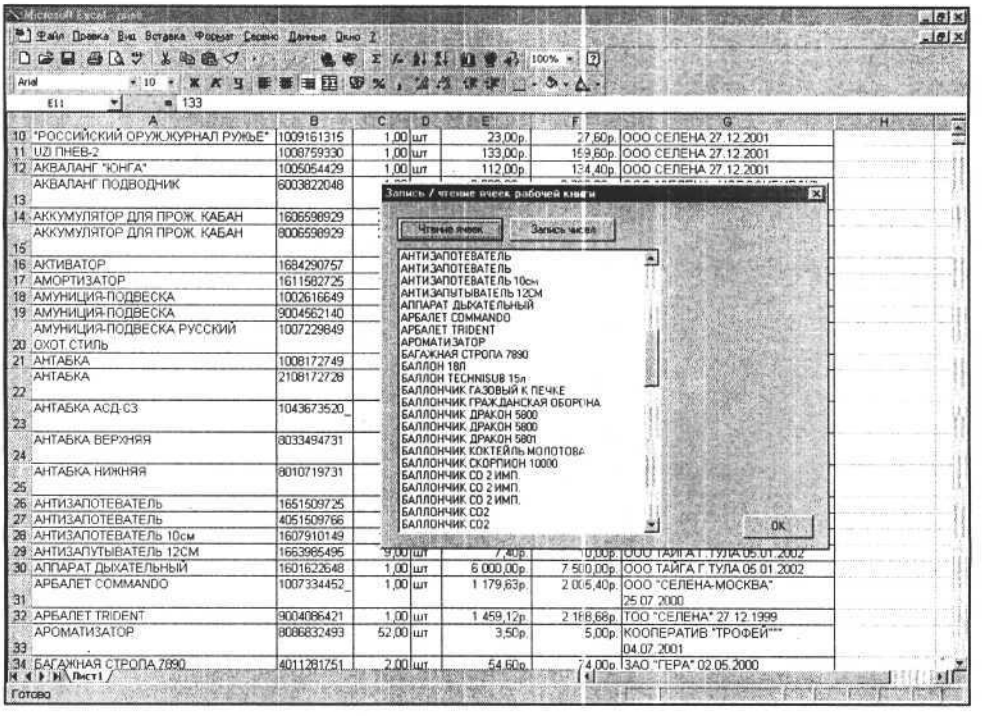


Рис. 11.13. Чтение информации листа рабочей книги Excel

В приведенных примерах для записи данных в ячейку мы присваивали значения непосредственно объекту Cells() или Range(). Эти данные распознавались как значения ячеек и отображались на листе. Но ячейки могут иметь разные типы. Например, они могут содержать как целые значения, так и формулы. Для чтения или записи значений используются, например, свойства Value и Text ячейки, для записи формулы — свойство Formula.

Изменим наши процедуры записи и чтения значений ячеек.

Использование свойств Value и Text ячейки для записи и чтения ее значения

```
procedure TOKBottomDlg4.Button1Click(Sender: TObject);
 var a_:integer;
begin
randomize;
for a_:=1 to 100 do Sheet.Cells(a_,1).Value:=random(10000);
end;

procedure TOKBottomDlg4.Button2Click(Sender: TObject);
 var eee_:string;
    a_:integer;
```

```
begin
ListBox1.Items.Clear;
for a_:=1 to 100 do begin
  eee_:=Sheet.Cells[a_,1].Text;
  ListBox1.Items.Add(eee_);
end;
end;
```

Ячейка или область представляет собой объект. Этот объект описывает не только значение, хранящееся и отображаемое в ячейке, но и другие ее параметры. Остальные свойства ячейки, которые будут рассмотрены в следующей главе, определяют способ отображения информации.

ГЛАВА 12

Работа с ячейками

Основа ввода и отображения информации в рабочих книгах Excel — работа с ячейками листа. Каждая ячейка представляет собой объект со множеством свойств, необходимых для отображения информации. Ячейка — прямоугольная область, имеющая определенные координаты в границах листа. Эти координаты могут выражаться как в буквенно-цифровом формате, так и в цифровом формате в виде номеров столбца и строки, пересечение которых и образует ячейку. Основным свойством каждой ячейки является ее содержание, или иначе *отображаемое значение*. Но часто бывает недостаточно просто отобразить какое-либо значение, будь то строка или число, поэтому для лучшего восприятия информации используется такое понятие, как *способ отображения*. На способ отображения влияют такие факторы, как *формат представления*, различные способы *размещения значения* относительно границ ячейки, *шрифт*, толщина и цвет *линий границ*, а также цвет и стиль *заливки* ячейки. Для доступа ко всем свойствам и содержимому ячеек в Excel используются два объекта — Range и Cells. Между ними есть только одно отличие — первый объект обеспечивает доступ к области[1] ячеек, а второй — только к одной ячейке. В конце предыдущей главы мы рассмотрели пример использования этих объектов для обмена информацией между приложением Delphi и рабочей книгой Excel, в настоящей главе рассмотрим работу с ячейками как с объектами, позволяющими представлять информацию в удобном для пользователя виде.

Объекты Range и Cells

Объекты Range и Cells обеспечивают доступ к выбранным ячейкам листа рабочей книги. По сути, Range и Cells представляют собой методы, возвращающие ссылку на объект-ячейку (или на область), но для понимания

---

[1] Область — одна ячейка или интервал смежных ячеек.

можно ассоциировать их с объектами. Для задания адреса ячеек, на которые мы хотим получить ссылку, используем аргументы этих методов. Для Range аргументом является строка адреса, а для Cells — номера строки и столбца. Так, например, для получения ссылки на объект-область можно использовать следующий оператор:

```
MyRange:=E.ActiveSheet.Range['B2'];
```

Оператор с использованием объекта Cells:

```
MyRange:=E.ActiveSheet.Cells[2,2];
```

После того как определен объект (ячейка или область ячеек), с ним можно работать.

Чтение и запись значений ячеек; очистка ячеек

Переменная, которую отображает ячейка, хранится в свойствах Text и Value объекта Range (Cells), поэтому для того чтобы ее получить, достаточно прочитать значение одного из этих свойств. Если тип данных (формат) значения ячейки неизвестен, используем свойство Text, чтобы получить его в виде строки. Когда тип данных известен, можно попытаться использовать свойство Value.

Допустим, нам неизвестен тип данных ячейки. Используем следующую процедуру, которая позволяет прочитать ее значение.

Чтение значения ячейки

```
procedure TOKBottomDlg2.Button1Click(Sender: TObject);
begin
  value1.Text:=Range.Text;
end;
```

При выполнении этой процедуры значение ячейки записывается в свойство Text объекта Value типа TEdit. Для записи в ячейку достаточно поменять местами объекты в операторе представленной выше процедуры и использовать свойство Value объекта Range, т. к. свойство Text в данном случае подходит только для чтения. Так мы и поступим, т. е. запишем содержание объекта TEdit приложения Delphi в ячейку рабочей книги Excel как текст.

Запись значения в ячейку

```
procedure TOKBottomDlg2.Button2Click(Sender: TObject);
begin
  Range.Value:=value1.Text;
end;
```

Используя данную процедуру, запишем строку символов '11111111' в ячейку B2. Результат представлен на рис. 12.1. Такой же результат можно получить, используя объект Cells:

```
Cell.Value:=value2.Text;
```

И в этом случае записанные в ячейку символы отображаются как строка.

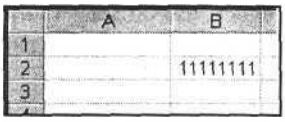


Рис. 12.1. Запись строки в ячейку

Для того чтобы записать значение в числовом формате, достаточно немного изменить процедуру и записывать в свойство Value не строковую переменную, а переменную типа Integer или Currency, как это показано на следующем примере.

Запись числового значения в ячейку

```
procedure TOKBottomDlg2.Button3Click(Sender: TObject);
begin
  Range.Value:=strtocurr(value1.Text);
end;
```

Результат выполнения процедуры представлен на рис. 12.2. В данном случае объект Range автоматически определил тип записываемого значения и отобразил его соответствующим образом, изменив формат ячейки.

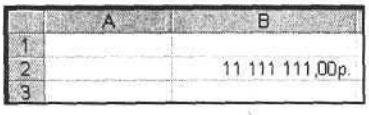


Рис. 12.2. Запись числа в ячейку

Если требуется отобразить записываемое значение в виде даты, то в свойство Value мы должны записать переменную, которая имеет в разрабатываемом приложении Delphi тип TDate.

Запись даты в ячейку

```
procedure TOKBottomDlg2.Button4Click(Sender: TObject);
begin
  Range.Value:=strtodate(value1.Text);
end;
```

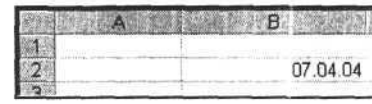


Рис. 12.3. Запись даты в ячейку

Соответственно результат будет выглядеть, как показано на рис. 12.3.

Мы выяснили следующий факт: свойство Value объекта Range, имеющее тип variant, допускает запись данных разного типа, что позволяет изменить формат ячейки в дальнейшем.

Это утверждение относится не только к объекту Range, но и к объекту Cells, т. к. они оба представляют собой ссылку на ячейку (или область). Но независимо от того, какой тип имело записанное в ячейку значение, мы можем изменить формат ячейки в любой момент.

Формат отображения данных ячейки

Формат ячейки определяется свойством NumberFormat объекта Range. Это свойство имеет строковый тип, а форматы данных представляют собой определенным образом заданные символьные константы. Используя свойство NumberFormat, можно изменить формат любой ячейки или области, а также прочитать значение формата с целью его анализа для внесения изменений.

Определим формат ячейки, которая содержит дату. Для этого воспользуемся представленной ниже процедурой, записанной для приложения Delphi (см. приложение на сопроводительном диске книги).

Определение формата ячейки

```
procedure TOKBottomDlg10.addresKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
  if Key<>13 then exit;
  Range:=form1.E.ActiveSheet.Range[addres.Text];
  NumberFormat.Text:=Range.NumberFormat;
end;
```

Для заданного адреса ячейки данная процедура возвращает ее формат, что мы и видим на рис. 12.4.

Итак, мы определили формат ячейки, отображающей дату. Для данного случая формат представляет собой строку типа 'ДД.ММ.ГГ'. В строке, где обычно записывают формулы, отображается числовое значение даты без формата. Если мы изменим формат, то само значение не изменится. Изменится только формат отображения его непосредственно в ячейке. Изменим

формат этой ячейки, чтобы отобразить год в четырехзначном формате с помощью следующей процедуры.

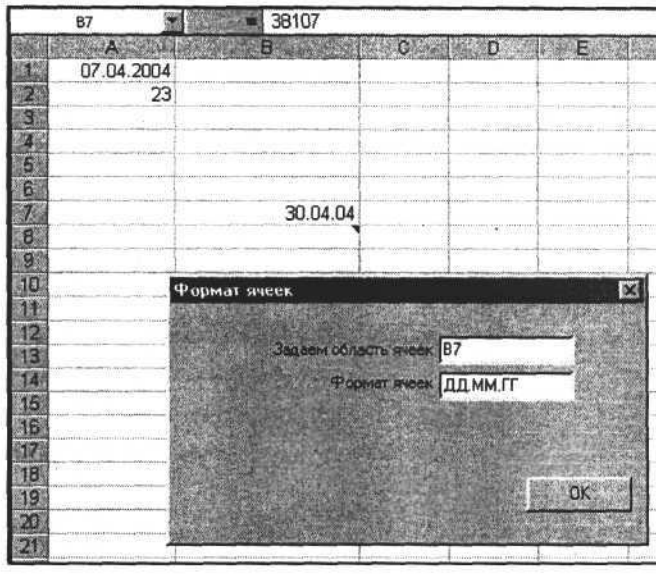


Рис. 12.4. Определяем формат выбранной ячейки

Изменение формата ячейки

```
procedure TOKBottomDlg10.NumberFormatChange(Sender: TObject);
begin
  Range.NumberFormat:=NumberFormat.Text;
end;
```

Изменив строку для задания формата к виду 'ДД.ММ.ГГГГ', мы сразу обнаружим изменение на рабочем листе книги Excel (рис. 12.5).

Как видно из рис. 12.4 и 12.5, значение ячейки в строке формул не изменилось — изменилось только отображение этого значения в ячейке рабочего листа. Понятно, что для данной ячейки необязательно задавать формат как дату. Если задать другой формат, то отображение записанного в нее числового значения изменится. Значение любого формата, обычно используемого в Excel, можно получить опытным путем. Например, установив его посредством изменения свойства ячеек, а затем прочитав с помощью описанные ранее процедуры.

Попробуем поэкспериментировать с форматом ячейки, записывая в свойство NumberFormat либо известные значения формата, либо то, что нам под-

сказывают опыт и интуиция. Например, зададим формат в виде строки символов '000 000 000 000'. В результате значение ячейки будет отображено как целое число, разбитое на разряды (рис. 12.6).

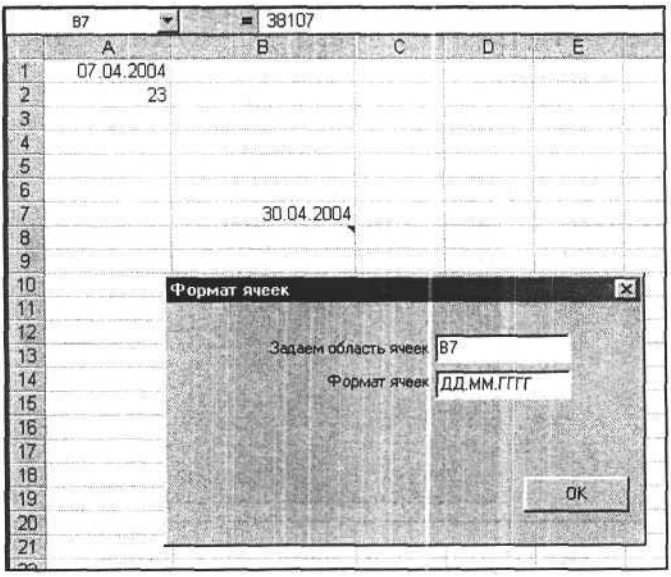


Рис. 12.5. Изменяем формат представления даты для выбранной ячейки

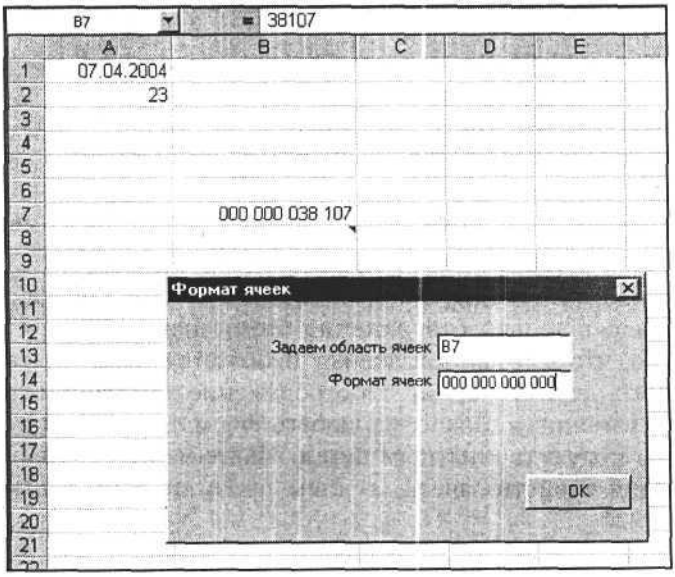


Рис. 12.6. Задание пользовательского формата для выбранной ячейки

Формулы

Мы рассмотрели способы записи значения ячейки из приложений Delphi, но ячейка и сама может принимать любое значение, когда она содержит формулу. *Формула* представляет собой некое математическое выражение, состоящее из констант, адресов ячеек, стандартных функций Excel, пользовательских функций и математических символов, записанных в виде строки. Для придания ячейке таких свойств используют запись строки, представляющей формулу, в свойство Formula объекта Range. Для чтения формулы также используется свойство Formula.

Зададим для выбранной ячейки ее формулу с помощью следующей процедуры.

Запись формулы

```
procedure TOKBottomDlg2.Button5Click(Sender: TObject);
begin
  Range.Formula:=value1.Text;
end;
```

В ячейку B7 запишем формулу, позволяющую вычислить конечную дату, просуммировав начальную дату и количество дней. Формула представляет собой выражение =A1+A2. Результат выполнения этой процедуры представлен на рис. 12.7.

Есть и другой способ, который требует определенных знаний от пользователя, но в некоторых условиях быть более эффективным. Этот способ основан на использовании мастера функций, представляющего собой последовательность диалоговых окон, которые позволяют конструировать формулу для данной ячейки по шагам. Для активизации данного способа необходимо вызвать метод FunctionWizard объекта Range. Рассмотрим следующую процедуру, которую можно использовать для конструирования формулы.

Вызов мастера функций

```
procedure TOKBottomDlg2.Button10Click(Sender: TObject);
begin
  Range.FunctionWizard;
end;
```

Вызов этой процедуры, содержащей только оператор активизации метода FunctionWizard, повлечет запуск диалогового окна (рис. 12.8). После выполнения всех необходимых шагов в ячейке Range будет записана формула.

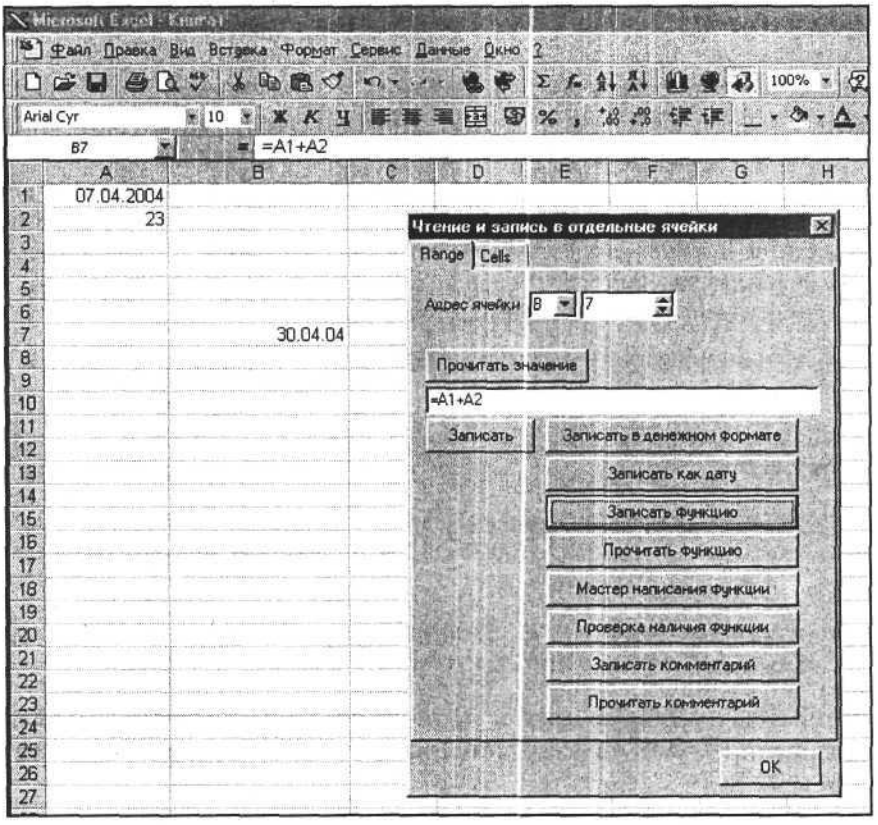


Рис. 12.7. Запись формулы в выбранную ячейку

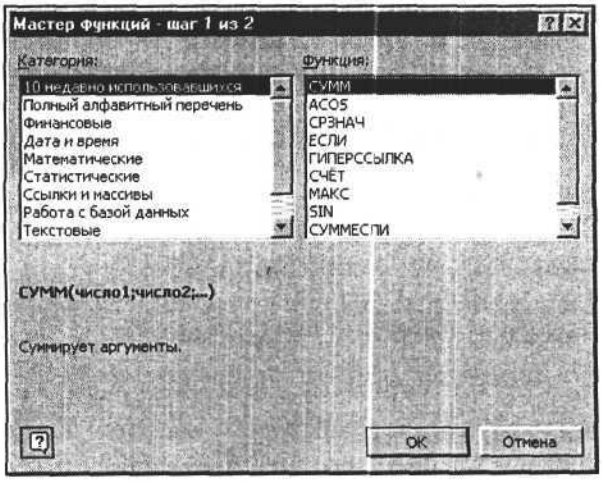


Рис. 12.8. Окно мастера функций

Иногда требуется проверить, что находится в ячейке — значение, записанное пользователем, или сформированное в результате выполнения формулы. Для этого можно анализировать содержимое свойства Formula объекта Range, но лучше использовать свойство HasFormula. Если оно имеет значение True, то ячейка содержит формулу, если False — то нет.

Проверка наличия формулы в ячейке

```
procedure TOKBottomDlg2.Button9Click(Sender: TObject);
begin
  if Range.HasFormula
  then messagebox(handle,'Данная ячейка содержит формулу!','Внимание!',0)
  else messagebox(handle,'Данная ячейка не содержит формулу!',
                  'Внимание!',0)
end;
```

В нашем примере ячейка B1 содержит формулу — это можно определить с помощью представленной процедуры, результат выполнения которой показан на рис. 12.9.

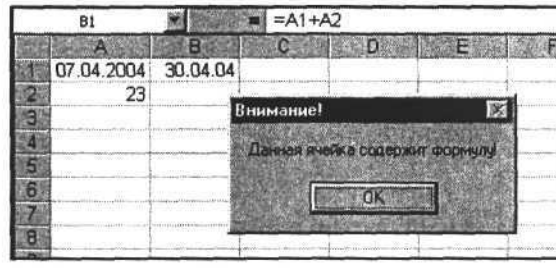


Рис. 12.9. Проверка наличия формулы в ячейке

Определив, что нужная ячейка содержит формулу, мы можем прочитать эту формулу с целью корректировки или анализа ее содержимого. Процедура чтения формулы может быть такой.

Чтение формулы

```
procedure TOKBottomDlg2.Button8Click(Sender: TObject);
begin
  value1.Text:=Range.Formula;
end;
```

Запись и чтение комментариев

Еще одним важным свойством, которым обладает ячейка листа рабочей книги Excel, является *комментарий*. Смысл использования комментария очевиден. Его можно записать индивидуально для каждой ячейки, он не отображается в печатной форме и служит только для описания содержимого ячеек. В одну ячейку можно записать один комментарий. Программно комментарий представляет собой объект Comment, принадлежащий ячейке. Для того чтобы изменить комментарий, необходимо очистить ячейку от старого комментария, а затем записать новый. Прочитать комментарий, записанный в данную ячейку, позволяет свойство Text объекта Comment.

Запишем, а затем прочтем комментарий для любой ячейки листа рабочей книги. Для этого используем следующие процедуры.

Запись и чтение комментария

```
procedure TOKBottomDlg2.Button6Click(Sender: TObject);
begin
  Range.ClearComments;
  Range.AddComment(value1.Text);
end;

procedure TOKBottomDlg2.Button7Click(Sender: TObject);
begin
  value1.Text:=Range.Comment.Text;
end;
```

Результат выполнения представленных процедур показан на рис. 12.10.

Мы рассмотрели некоторые действия с одной выбранной ячейкой с применением объектов Range и Cells. Результаты при использовании любого из этих объектов полностью идентичны. Но если требуется воздействие на область, нам придется использовать объект Range. Аргументом данного объекта может быть адрес одной ячейки или прямоугольной области из нескольких ячеек.

Область (интервал ячеек)

Попробуем задать координаты области и поработать с ними. В первую очередь нас могут заинтересовать такие параметры этой области, как количество ячеек и ее реальный адрес на листе рабочей книги Excel. Во-вторых, нам будет полезно узнать, как изменить, скопировать или очистить содержание ячеек этой области.

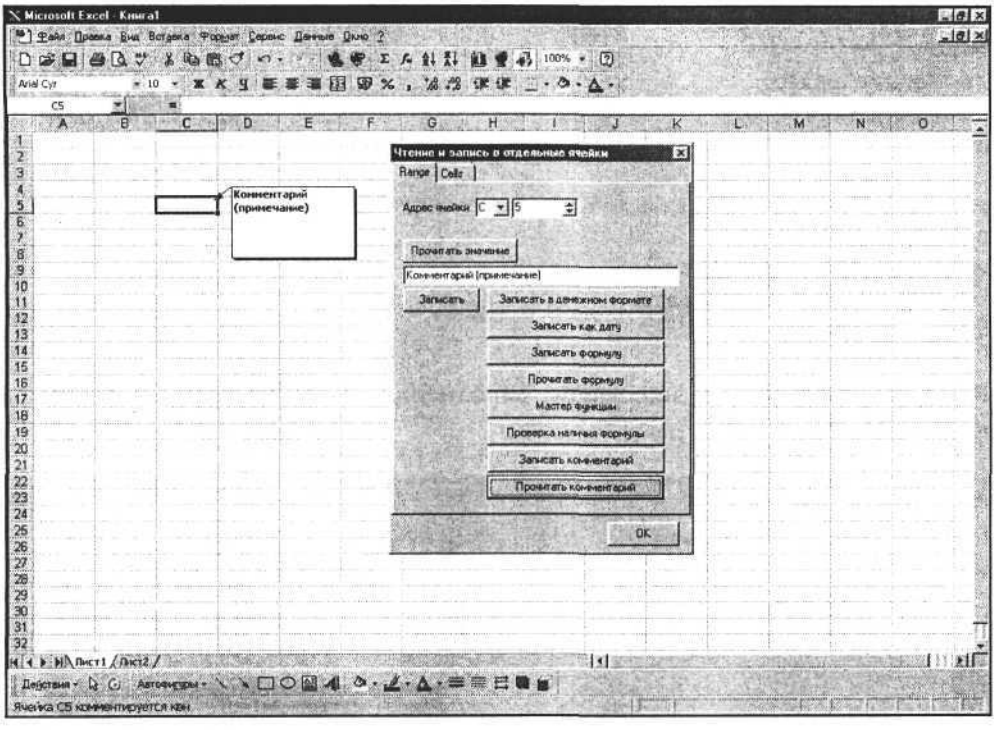


Рис. 12.10. Запись и чтение комментария для ячейки

Рассмотрим в качестве примера следующие процедуры (см. приложение на сопроводительном диске книги).

Параметры области, заполнение ячеек области

```
procedure TOKBottomDlg3.addresExit(Sender: TObject);
begin
// Определяем ссылку на область
  Range:=form1.E.ActiveSheet.Range[addres.Text];
  count.Text:=inttostr(Range.Count); // Количество ячеек в области
  area_addres.Text:=Range.Address;       // Реальный адрес области
end;

procedure TOKBottomDlg3.Button1Click(Sender: TObject);
begin
// Выделяем область
  Range.Select;
end;
```

```
procedure TOKBottomDlg3.Button3Click(Sender: TObject);
begin
// Очищаем область
  Range.Clear;
end;

procedure TOKBottomDlg3.Button2Click(Sender: TObject);
begin
// Заполняем ячейки выбранной области
  Range.Value:=text1.Text;
end;
```

Результат выполнения процедуры заполнения ячеек области представлен на рис. 12.11.

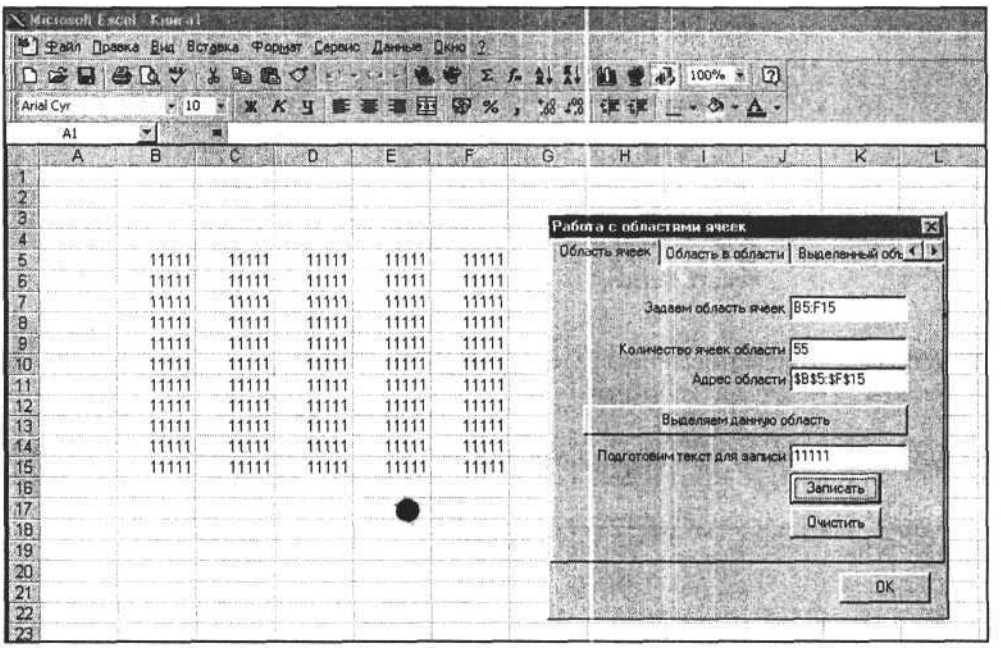


Рис. 12.11. Заполнение ячеек области

К области можно применять почти все методы, которые мы применяли к одной ячейке. Есть и дополнительные возможности, присущие области. Важной особенностью является использование виртуального адреса внутри заданной области. Это означает, что внутри области можно задавать адреса, начиная с адреса A1 для верхней левой ячейки области. Эта замечательная особенность обусловлена тем, что объект Range включает в себя такой же по характеристикам дочерний объект Range. Первая ячейка дочернего объ-

екта имеет виртуальный адрес A1, а ее реальный адрес совпадает с адресом начала области. Это может избавить программиста от многих проблем, связанных с адресацией ячеек, и существенно упростить исходный текст программы. Например, зададим адрес и размер области 'B5:F15'. Получим ссылку на нее:

```
Range:=E.ActiveSheet.Range['B5:F15'];
```

После этого определим дочернюю область, принадлежащую заданной области. Для этого используем оператор:

```
MyRange:=Range['A1:B2'];
```

Область MyRange принадлежит области Range и находится в ее начале. Далее заполним ячейки области MyRange строкой символов, например:

```
MyRange.Value:='222';
```

Все эти манипуляции можно выполнить иначе (см. следующую процедуру).

Использование виртуальной области

```
procedure TOKBottomDlg3.Button4Click(Sender: TObject);
begin
  Range.Range[addres1.Text].Value:=text2.Text;
end;
```

Результат будет таким же (рис. 12.12).

Мы рассмотрели случаи, когда область задается программным путем посредством адресов. Но часто пользователь просто выделяет область ячеек, а затем выполняет с ней последовательность каких-либо действий. Программист Delphi, разрабатывая свое приложение, должен учесть этот случай и обработать выделенную часть листа рабочей книги Excel. Одним из свойств объекта Application является объект Selection, который обладает множеством свойств и методов, присущих визуальным компонентам Excel, в том числе некоторыми свойствами объекта Range, позволяющими обеспечить доступ к выделенной области. В общем случае для обеспечения доступа к выделенным ячейкам можно воспользоваться оператором

```
Range:=E.Selection;
```

который присваивает переменной Range значение ссылки на область выделенных ячеек. После этого программист может воздействовать на эту область.

Есть и другой способ обращения к выделенной области — без использования промежуточных ссылочных переменных. Можно напрямую применять

к объекту Selection методы и использовать свойства объекта Range, но только когда выделена хотя бы одна ячейка. Следующие процедуры демонстрируют это.

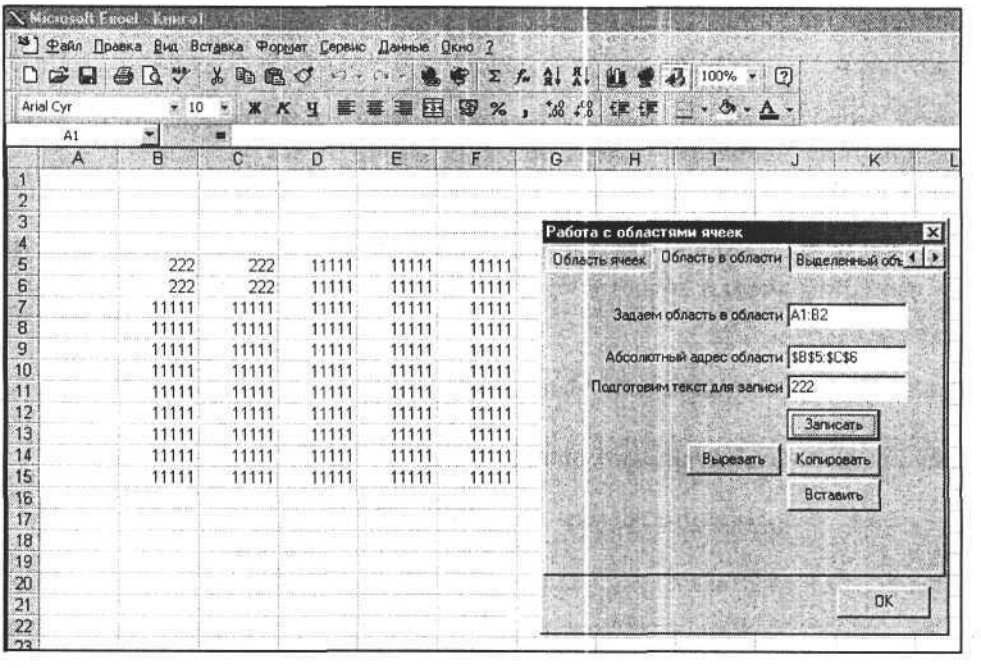


Рис. 12.12. Работа с виртуальной областью

Определение адреса и заполнение выделенной области

```
procedure TOKBottomDlg3.Button11Click(Sender: TObject);
begin
  address_s.Text:=E.Selection.Address;
end;

procedure TOKBottomDlg3.Button12Click(Sender: TObject);
begin
  E.Selection.Value:=text_s.Text;
end;
```

На рис. 12.13 представлена форма приложения Delphi, в которой используются представленные процедуры, определяющие адрес выделенной области и заполняющие все ее ячейки значением, выбранным пользователем этого приложения.

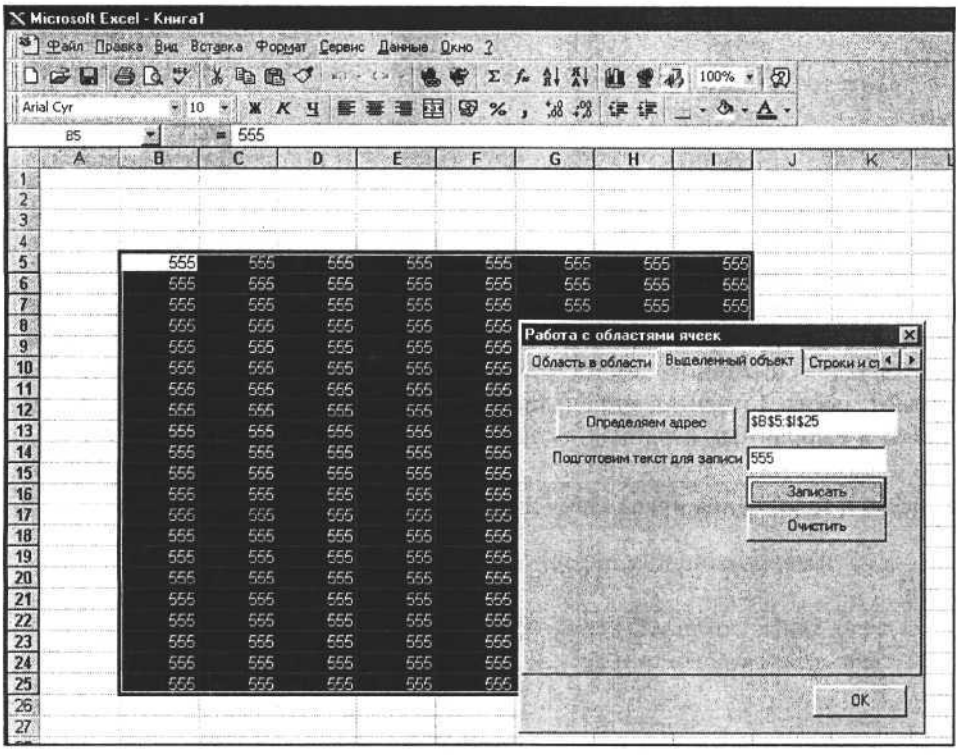


Рис. 12.13. Работа с областью выделенных ячеек

Когда требуется программно выделить интервал ячеек листа Excel, используют метод Select объекта Range. После этого можно обращаться к ячейкам уже как к выделенной области, используя объект Selection.

Обычно к выделенной области применяют методы, позволяющие копировать ее содержимое в буфер обмена (метод Copy), или, реже, методы, позволяющие переместить (метод Cut) содержимое в другую область или приложение или документ. Выделенную область ячеек можно также заменить информацией из буфера обмена (метод PasteSpecial). Все эти методы используются, как правило, применительно к объекту Selection, но иногда бывает необходимо применить их непосредственно к ячейкам или областям ячеек, используя объект Range. Все зависит от конкретной постановки задачи и от стиля программирования.

Программное выделение и очистка области

```
procedure TOKBottomDlg3.Button1Click(Sender: TObject);
begin
  Range.Select;
end;
```

```
procedure TOKBottomDlg3.Button13Click(Sender: TObject);
begin
  E.Selection.Clear;
end;
```

От объектов, описывающих ячейки и области, перейдем к другим объектам — столбцам и строкам. Если задана область ячеек и получена ссылка Range на нее, то количество столбцов будет определяться свойством Count коллекции Columns объекта Range, а количество строк — свойством Count коллекции Rows объекта Range. Доступ ко всем строкам и столбцам листа рабочей книги предоставляют коллекции Rows и Columns объекта ActiveSheet — активного листа рабочей книги. Следующие процедуры позволяют, соответственно, определить количество строк и столбцов, очистить ячейки в выбранных столбцах и строках и заполнить их.

Очистка и заполнение ячеек строк и столбцов листа рабочей книги

```
procedure TOKBottomDlg3.FormCreate(Sender: TObject);
begin
  row_ount.text:=inttostr(form1.E.Application.ActiveSheet.Rows.Count);
  col_ount.text:=inttostr(form1.E.Application.ActiveSheet.Columns.Count);
end;

procedure TOKBottomDlg3.Button7Click(Sender: TObject);
begin
  E.ActiveSheet.Rows[Row.Value].Clear;
  E.ActiveSheet.Rows[Row.Value].Value:= text_rc.Text;
end;

procedure TOKBottomDlg3.Button8Click(Sender: TObject);
begin
  E.ActiveSheet.Columns[Col.Value].Clear;
  E.ActiveSheet.Columns[Col.Value].Value:= text_rc.Text;
end;
```

На рис. 12.14 показан результат выполнения представленных процедур.

Вырезание, вставка и удаление ячейки

Мы рассмотрели только манипуляции с содержимым ячеек, при этом сами ячейки оставались неизменными, т. е. их другие характеристики не менялись. Если бы ячейка имела заливку и линии границы, то изменение или даже полная очистка ее содержимого не повлияла бы на ее другие визуаль-

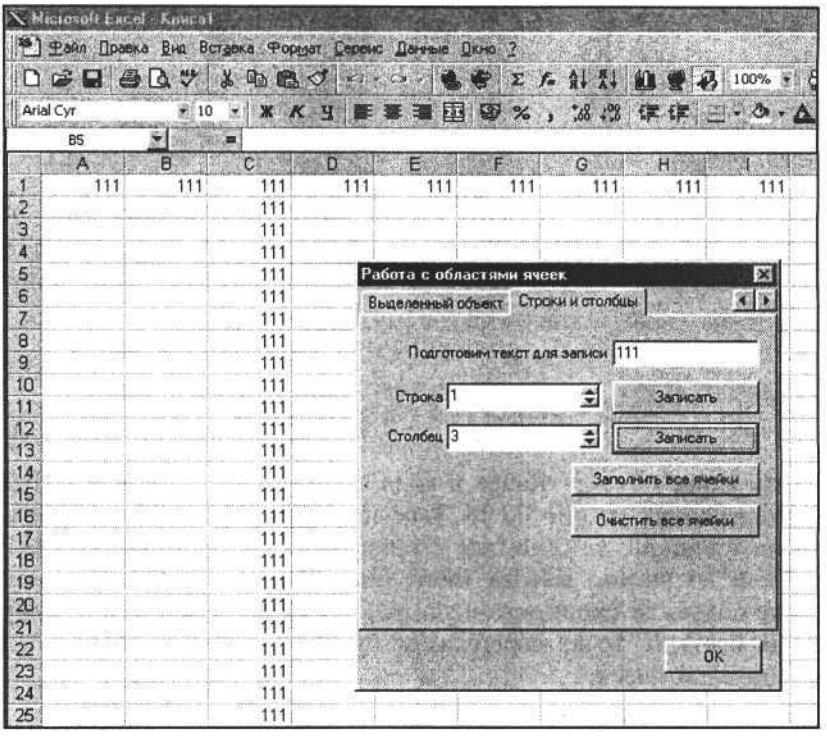


Рис. 12.14. Работа со строками и столбцами ячеек

ные характеристики — они остались бы неизменными. Радикальный способ, позволяющий полностью изменить ячейку по определенному адресу, — это удаление ячейки или вставка на ее место другой. При этой манипуляции происходит смещение остальных ячеек к началу или к концу в зависимости от выбранной операции над ячейками. Для удаления ячейки предназначен метод Delete объекта Range, а для вставки — метод Insert объекта Range. Оба метода могут быть вызваны с аргументом. При удалении аргумент метода Delete определяет, какая ячейка из столбца или строки будет замещать удаленную ячейку. При вставке новой ячейки аргумент метода Insert определяет, какие ячейки будут сдвинуты — ячейки прилегающего столбца или строки. Методы Delete и Insert можно применять не только к отдельным ячейкам, но и к группе ячеек, а также к строкам и столбцам ячеек.

Рассмотрим, как используются эти методы в приложениях Delphi. На рис. 12.15 представлен фрагмент листа рабочей книги Ecxel, в ячейках которого записаны некоторые числовые константы. Требуется вставить новые четыре ячейки в область A1:B2. Используем метод Insert без аргументов. По умолчанию, при вставке ячеек сдвигаются ячейки, расположенные ниже вставляемой области. Используем следующую процедуру.

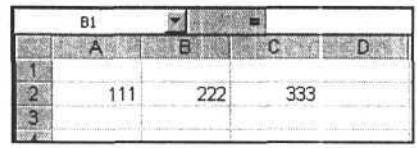


Рис. 12.15. Фрагмент листа рабочей книги

Вставка новых ячеек

```
procedure TOKBottomDlg4.Button1Click(Sender: TObject);
begin
  Form1.E.Application.Range[address.Text].Insert;
end;
```

Если задать адрес объекта Range в виде строки 'A1:B2', то мы получим результат, показанный на рис. 12.16. Как видно из рис. 12.16, ячейки области A1:B2 были смещены вниз на две строки, а на их место вставлены новые, пустые. Если до вызова метода Insert был вызван метод Copy, или буфер обмена уже содержал скопированную ранее информацию (т. е. в буфере уже находились ячейки), то их содержимое будет вставлено в область A1:B2 на место смещенных ячеек.

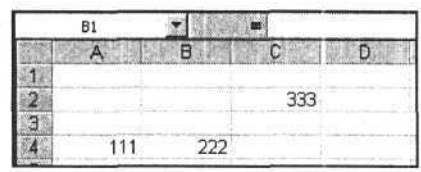


Рис. 12.16. Вставка новых ячеек

Поиск и замена текста

Методы поиска и замены текста очень важны при использовании шаблонов документов[1]. В прошлых главах мы рассмотрели, как с помощью поиска и замены в шаблонах документов для Word получить метод быстрой и гибкой подготовки документов. В рабочих книгах Excel также нет смысла отказываться от этой возможности, поэтому сейчас мы ее и рассмотрим.

Поиск текста выполняется путем вызова метода Find. В самом простом случае метод Find должен содержать один аргумент — искомый текст. Если

[1] Напомню, что здесь *шаблоном документа* называется образец документа, заполняемый нужным текстом. Не путать с файлом шаблона .DOT.

требуется уточнить режим поиска, то используются дополнительные необязательные аргументы этого метода, которые позволяют определить направление поиска, область поиска и другие параметры. Метод Find возвращает ссылку на ячейку, если поиск прошел удачно, иначе возвращается ссылка на пустой объект. Рассмотрим, как можно реализовать поиск в приложениях Delphi.

Поиск текста на листе рабочей книги

```
procedure TForm1.Button6Click(Sender: TObject);
 var eee_:string; FindRange:variant;
begin
  eee_:=InputBox('Поиск текста!','','');
  if eee_<>'' then begin
    try
    FindRange:=E.Cells.Find(What:=eee_);
    eee_:= FindRange.text;
    except
    messagebox(handle,'Искомый текст не найден!','Внимание!',0);
    end;
  end;
end;
```

Используя представленную выше процедуру, зададим условия поиска (рис. 12.17).

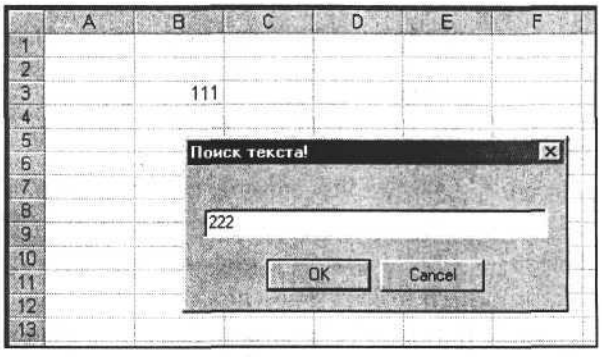


Рис. 12.17. Задание условия поиска

Метод Find вернул ссылку на искомый объект. Если этот объект существует реально (в случае успешного поиска), то мы сможем прочитать содержимое ячейки, на которую указывает ссылка, и это не вызовет ошибку. Иначе возникнет исключительная ситуация, обработав которую мы выведем сообщение об отрицательном результате поиска (рис. 12.18).

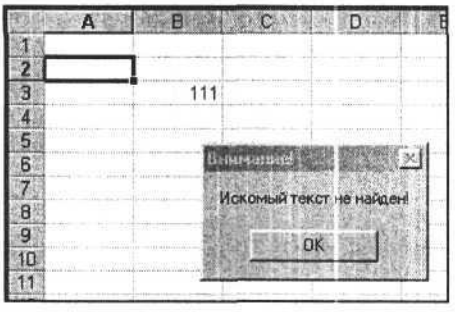


Рис. 12.18. Вывод результата поиска

Мы рассмотрели случай поиска на всем листе рабочей книги Excel. Чтобы произвести поиск текста в ограниченной области ячеек, необходимо вызвать метод Find объекта Range, где Range — область (интервал ячеек). Например:

```
FindRange:=Range('A1:H500').Find('222');
```

В этом случае поиск будет выполнен только в области ячеек, ограниченной интервалом A1:H500.

Полная спецификация вызова метода Find:

```
Find(What, After, LookIn, LookAt, SearchOrder, SearchDirection,
     MatchCase, MatchByte);
```

Аргументы метода Find приведены в табл. 12.1.

Таблица 12.1. *Аргументы метода Find объекта Range*

Аргумент	Тип	Значение
What	String	Строка поиска
After	Range	Ячейка или область, после которой производится поиск; позволяет задать адрес начала поиска
LookIn	Integer	Где ищем (в значениях ячейки, в формулах или в примечаниях)
LookAt	Integer	Поиск совпадения строки текста целиком или ее части
SearchOrder	Integer	Порядок поиска (перебираем строки или столбцы)
SearchDirection	Integer	Направление поиска (к началу, к концу)
MatchCase	Boolean	Учет регистра букв при поиске
MatchByte	Boolean	Предназначен для двухбайтовых символов (в версиях для Windows не используется)

После успешного поиска текста метод Find возвращает ссылку на объект-ячейку, используя который мы можем изменить содержание ячейки. Повторяя поиск и замену многократно, можно сформировать необходимый документ, но для этого есть более эффективный способ — использование функции поиска и замены. Эта функция в Excel реализуется методом Replace, который имеет два обязательных аргумента — искомый текст и текст для замены. В качестве примера использования этого метода рассмотрим следующую процедуру.

Поиск и замена текста

```
procedure TForm1.Button7Click(Sender: TObject);
 var eee_,uuu_:string;
begin
  eee_:=InputBox('Поиск текста!','','');
  uuu_:=InputBox('Замена на текст!','','');
  if eee_<>'' then begin
    try
    range:=E.Cells.Replace(What:=eee_,Replacement:=uuu_);
    except
    messagebox(handle,'Искомый текст не найден!','Внимание!',0);
    end;
  end;
end;
```

В результате выполнения этой процедуры будет заменен текст, если условия поиска выполнены. Иначе будет сгенерирована исключительная ситуация, обработав которую мы выведем сообщение об ошибке и невозможности замены текста. Полная спецификация метода Replace:

```
Replace(What, Replacement, LookAt, SearchOrder, MatchCase, MatchByte);
```

Аргументы метода Replace приведены в табл. 12.2.

Таблица 12.2. *Аргументы метода Replace объекта Range*

Аргумент	Тип	Значение
What	String	Строка поиска
Replacement	String	Строка замены
LookAt	Integer	Поиск совпадения строки текста целиком или ее части
SearchOrder	Integer	Направление поиска (строки, столбцы)
MatchCase	Boolean	Учет регистра букв при поиске
MatchByte	Boolean	Предназначен для двухбайтовых символов (в версиях для Windows не используется)

Высота и ширина ячейки

От методов, позволяющих изменять значение ячеек листа рабочей книги Excel, переходим к изучению других свойств и методов основного объекта приложения Excel — ячейки. Начнем с простого — с определения и изменения размеров самой ячейки, связанных с размерами строк и столбцов. Мы определились, что ячейка всегда определяется объектами Range или Cells. Используя свойства ColumnWidth и RowHeight этих объектов, мы имеем возможность изменить ширину и высоту ячейки. Очевидно, что эти изменения повлекут изменения ширины столбца и высоты строки. В качестве примера использования этих свойств рассмотрим процедуры, позволяющие определить и изменить размеры заданной ячейки.

Определение и изменение размера ячейки

```
procedure TOKBottomDlg5.addresExit(Sender: TObject);
begin
  Range:=E.ActiveSheet.Range[addres.Text];
  ColumnWidth.Text:=FloatToStr(Range.ColumnWidth);
  RowHeight.Text:=FloatToStr(Range.RowHeight);
end;

procedure TOKBottomDlg5.ColumnWidthChange(Sender: TObject);
begin
  Range.ColumnWidth:=StrToFloat(ColumnWidth.Text);
end;

procedure TOKBottomDlg5.RowHeightChange(Sender: TObject);
begin
  Range.RowHeight:=StrToFloat(RowHeight.Text);
end;
```

Результат выполнения этих процедур приведен на рис. 12.19.

Можно добавить, что если объект Range ассоциируется с областью ячеек, то действие данных процедур повлечет за собой изменение размеров всех ячеек этой области. Для изменения размеров строки (строк), столбца (столбцов) нужно записать новые значения в свойства RowHeight и ColumnWidth объектов Rows.Item(i:integer) и Columns.Item(i:integer) соответственно. Чтобы изменить высоту и ширину всех строк и столбцов ячеек, используйте свойства RowHeight и ColumnWidth коллекций строк и столбцов (Rows и Columns). С учетом этого несколько изменим единственный оператор в процедурах настройки высоты и ширины ячейки и получим следующий программный текст.

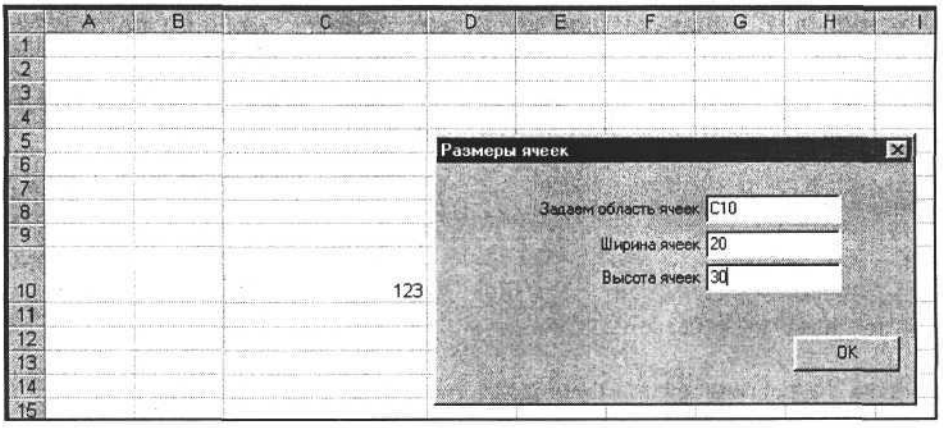


Рис. 12.19. Изменение размеров ячейки

Изменение ширины и высоты ячейки

```
procedure TOKBottomDlg5.ColumnWidthChange(Sender: TObject);
begin
  Columns.Item(col).ColumnWidth:=StrToFloat(ColumnWidth.Text);
end;

procedure TOKBottomDlg5.RowHeightChange(Sender: TObject);
begin
  Rows.Item(row).RowHeight:=StrToFloat(RowHeight.Text);
end;
```

Результаты выполнения этих и рассмотренных ранее процедур будут полностью идентичными (см. рис. 12.19). Различаются только исходные данные. В первом случае они определяются адресом ячейки, во втором — номером строки и столбца.

Выравнивание текста в ячейке

Отображаемое значение можно разместить в ячейке Excel по-разному. Обычно способ размещения соответствует типу и величине значения ячейки. Для числовых данных текст смещен к правой границе, а для строковых — к левой границе ячейки, но бывают исключения. Исключения из правил определяются требованиями, предполагающими эффективное отображение информации для более полного ее восприятия. В таких случаях и используют свойства ячейки, позволяющие задать режимы горизонтального

и вертикального выравнивания, установить режимы переноса по словам и автоподбора ширины, а также выбрать угол поворота текста.

Рассмотрим эти свойства. Следующий программный код задает режимы горизонтального и вертикального выравнивания текста по центру.

Выравнивание текста в ячейке

```
const
xlHAlignCenter = -4108;
xlVAlignCenter = -4108;

procedure TOKBottomDlg8.HorizontalAlignmentChange(Sender: TObject);
begin
  range.HorizontalAlignment:=xlHAlignCenter;
end;

procedure TOKBottomDlg8.VerticalAlignmentChange(Sender: TObject);
begin
  range.VerticalAlignment:=xlVAlignCenter;
end;
```

На рис. 12.20 представлен результат выравнивания текста по центру ячейки.

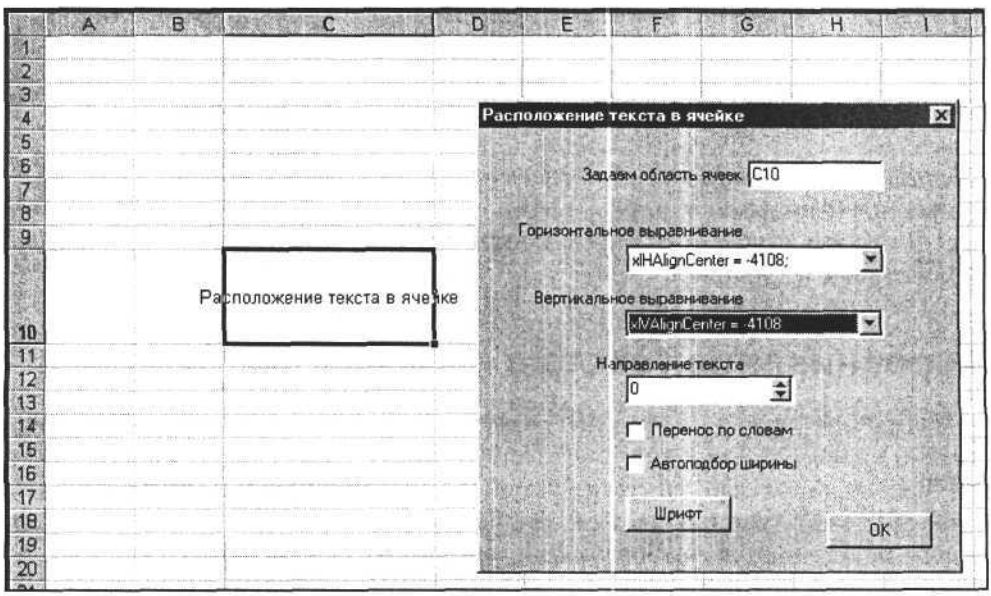


Рис. 12.20. Выравнивание текста в ячейке

Если длина текста превосходит ширину ячейки, то это может повлечь за собой искажения отображения значения ячейки (см. рис. 12.20). Для решения этой проблемы можно воспользоваться режимом переноса по словам или режимом автоматического подбора ширины текста по ширине ячейки.

Рассмотрим режим переноса по словам. Он включается, когда свойство ячейки WrapText установлено в значение True, и отключается, когда свойство WrapText установлено в значение False.

Перенос по словам

```
procedure TOKBottomDlg8.WrapTextClick(Sender: TObject);
begin
  Range.WrapText:=WrapText.Checked;
end;
```

Режим переноса по словам позволяет разместить текст, как показано на рис. 12.21.

В этом режиме высота ячейки автоматически изменяется в зависимости от размера текста. Если текст в ячейке выравнивается по ее левой границе, то можно задать величину отступа, определяемую свойством IndentLevel объекта Range. Оператор Range.IndentLevel=2; задает отступ от левой границы ячейки в два пункта.

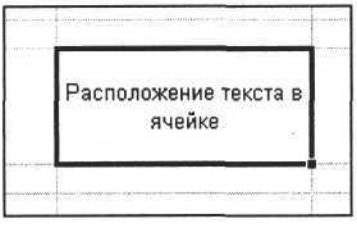


Рис. 12.21. Перенос текста по словам

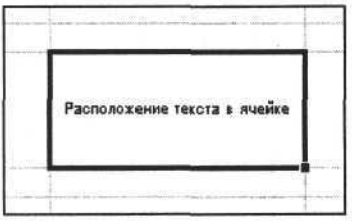


Рис. 12.22. Использование автоподбора ширины текста

Когда требуется, не изменяя размер ячейки, записать в нее текст, ширина которого может превысить ширину ячейки, необходимо воспользоваться свойством ShrinkToFit. Если установить это свойство в значение True, то размер шрифта текста будет автоматически устанавливаться так, чтобы текст по ширине всегда мог вписаться в ячейку (рис. 12.22).

Еще один способ изменения расположения текста в ячейке — его поворот. Поворот текста, отображающего значение ячейки, определяется свойством Orientation и может быть задан величиной от −90 до +90 градусов.

Поворот текста в ячейке

```
procedure TOKBottomDlg8.OrientationChange(Sender: TObject);
begin
  Range.Orientation:=Orientation.Value;
end;
```

Результат выполнения представленной выше процедуры Delphi может выглядеть, как показано на рис. 12.23. При программировании поворота текста нужно учитывать, что шаг угла поворота составляет 1 градус в пределах от −90 до +90 градусов.

Рис. 12.23. Поворот текста

Если длина текста, размещаемого в ячейке, настолько велика, что он не может быть размещен там без существенных изменений размеров ячейки, а по конкретным условиям задачи размеры строк и столбцов изменять нельзя,

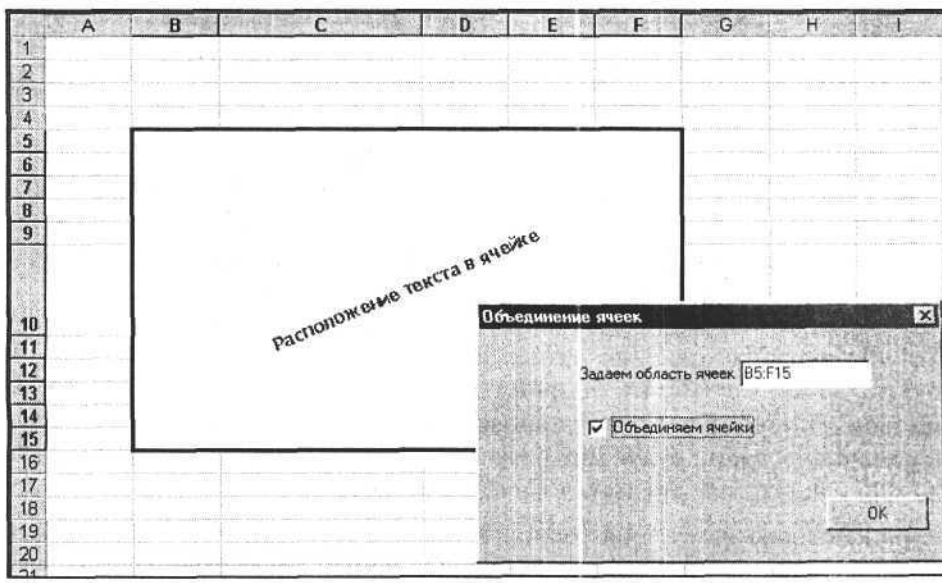


Рис. 12.24. Объединение ячеек

то следует использовать режим *объединения ячеек*. Объединение ячеек осуществляется установкой в значение True свойства MergeCells объекта Range, ассоциированного с областью ячеек.

Объединение ячеек

```
procedure TOKBottomDlg9.MergeCellsClick(Sender: TObject);
begin
  Range.MergeCells:=MergeCells.Checked;
end;
```

На рис. 12.24 приведен пример объединения группы ячеек с использованием представленной процедуры. Адрес объединенной ячейки соответствует адресу левой верхней ячейки области, в которой выполнялось объединение.

Шрифт

Важным свойством текста в Excel является шрифт отображаемых значений ячеек и текста других объектов. Для ячеек шрифт является одним из свойств объекта Range, при этом в одной ячейке не может быть, например, два фрагмента текста с разным шрифтом. Шрифт определяется свойством Font ячейки или области ячеек. Шрифт по существу тоже является объектом со своими свойствами, определяющими те или иные характеристики отображения текста. Свойства объекта Font и их характеристика представлены в табл. 12.3.

Таблица 12.3. *Свойства объекта Font*

Свойство	Тип	Значение
Name	String	Имя
Background	Integer	Стиль прорисовки фона (используется только для диаграмм)
FontStyle	String	Строковое описание стиля
Bold	Boolean	Полужирный
Italic	Boolean	Курсив
Size	Integer	Размер шрифта
Strikethrough	Boolean	Зачеркнутый
Superscript	Boolean	Верхний индекс
Subscript	Boolean	Нижний индекс
OutlineFont	Boolean	Не используется

Таблица 12.3 *(окончание)*

Свойство	Тип	Значение
Shadow	Boolean	Не используется
Underline	Integer	Подчеркивание
ColorIndex	Integer	Индекс цвета из палитры цветов
Color	Integer	Цвет

Программистов Delphi в первую очередь будет интересовать, каким образом шрифт стандартных объектов разрабатываемых приложений может соответствовать объекту Font, принадлежащему объекту Excel.Application и подчиненным объектам. Иными словами — как управлять шрифтом создаваемых документов в Excel из приложений. В Delphi свойства шрифта несколько отличаются от свойств шрифта в Excel, поэтому чтобы решить проблему согласования, достаточно написать небольшую процедуру, преобразующую свойства шрифта Delphi в свойства шрифта Excel. Вот пример такой процедуры.

Выбор шрифта для текста ячейки

```
function SetFontRange(range_:variant; font:Tfont):boolean;
begin
  SetFontRange:=True;
  try
  Range_.Font.Name:=font.Name;
  if fsBold in font.Style
    then Range_.Font.Bold:=True
    else Range_.Font.Bold:=False;
  if fsItalic in font.Style
    then Range_.Font.Italic:=True
    else Range_.Font.Italic:=False;
  Range_.Font.Size:=font.Size;
  if fsStrikeOut in font.Style
    then Range_.Font.Strikethrough:=True
    else Range_.Font.Strikethrough:=False;
  if fsUnderline in font.Style
    then Range_.Font.Underline:=xlUnderlineStyleSingle
    else Range_.Font.Underline:=xlUnderlineStyleNone;
  Range_.Font.Color:=font.Color;
  except
  SetFontRange:=false;
  end;
end;
```

```
procedure TOKBottomDlg8.Button1Click(Sender: TObject);
begin
  if not FontDialog1.Execute then exit;
  SetFontRange(range,FontDialog1.Font);
end;
```

На рис. 12.25 видно изменение шрифта ячейки, полученное с помощью представленного программного кода.

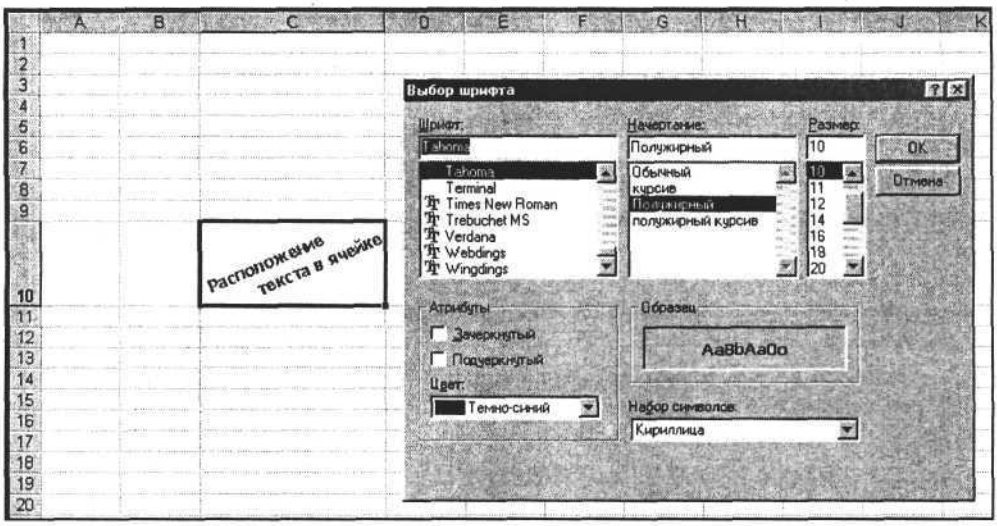


Рис. 12.25. Настройка шрифта

Границы ячейки

Перейдем к анализу графического оформления ячейки. Ячейка представляет собой прямоугольную область. Эта область, кроме отображаемого в ней значения, имеет такие свойства, как заливка и граница. Как заливка ячейки, так и граница имеют соответствующие свойства (цвет, толщину, тип, узор, цвет узора). Рассмотрим эти свойства подробней. Граница ячейки представляет собой линии, ограничивающие ее с четырех сторон. Линии объединены в коллекцию Borders, доступ к любой из них осуществляется через элементы этой коллекции. Каждый элемент коллекции предоставляет доступ к отрезку прямой, прилегающему к той или другой стороне ячейки. Диагонали ячейки тоже являются элементами этой коллекции. Каждый элемент коллекции Borders является объектом и имеет свои индивидуальные свойства, что позволяет задать тип линии и цвет отдельно для каждой линии границы ячейки. Рассмотрим процедуры, позволяющие обеспечить дос-

туп к любой линии границы (или ко всей границе сразу) выбранной ячейки, установить ее толщину, тип линии и цвет.

Задаем свойства линии границы ячейки

```
procedure TOKBottomDlg6.BordersChange(Sender: TObject);
begin
// Получаем доступ к выбранной линии границы или ко всей границе ячейки
if CheckBox1.Checked then Border:=Range.Borders.item[Borders.ItemIndex+5]
else Border:=Range.Borders;
end;

procedure TOKBottomDlg6.WeightChange(Sender: TObject);
begin
// Устанавливаем толщину линии границы ячейки
  Border.Weight:=Weight.ItemIndex;
end;

procedure TOKBottomDlg6.LineStyleChange(Sender: TObject);
begin
// Устанавливаем тип линии границы ячейки
  Border.LineStyle:=xlDouble;
end;

// Устанавливаем цвет линии границы ячейки
procedure TOKBottomDlg6.Button1Click(Sender: TObject);
begin
  if not ColorDialog1.Execute then exit;
  Border.Color:=ColorDialog1.Color;
end;
```

Результат настройки линий границ ячеек с использованием приведенных процедур представлен на рис. 12.26. Полный текст этих процедур имеется на сопроводительном компакт-диске книги.

Заливка ячейки

Заливка ячейки определяется комбинацией следующих составляющих — цвета, узора и цвета узора, заполняющего пространство ячейки. Программный доступ к параметрам заливки обеспечивает свойство Interior объекта Range, где Range — ссылка на ячейку или область. Свойство Interior представляет собой объект, свойства которого связаны с визуальными свойствами внутреннего пространства ячейки.

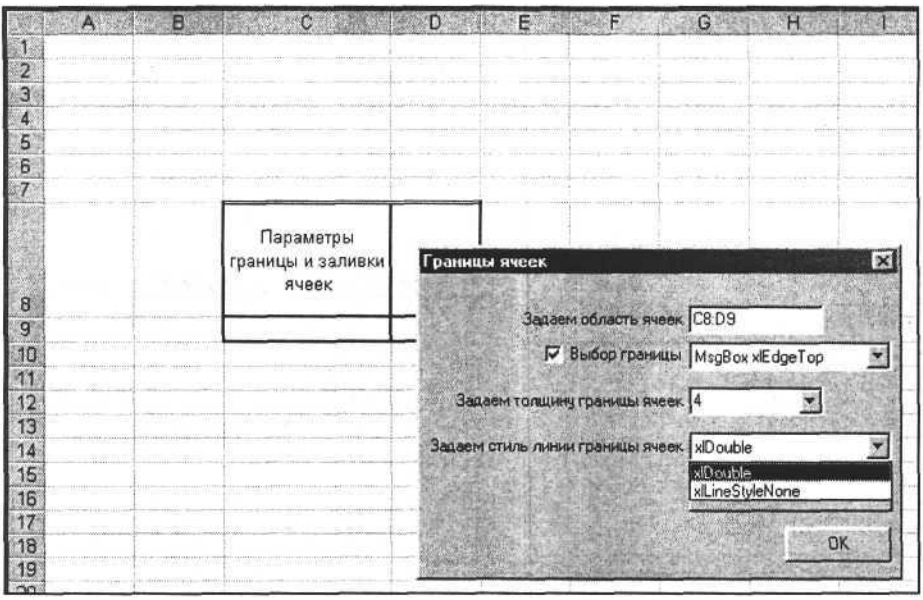


Рис. 12.26. Настройка параметров границы ячейки

Цвет заливки определяется свойством Color объекта Interior и задается как комбинация трех цветов:

```
Color:=RGB(R, G, B);
```

где R, G, B — числовые значения, соответствующие красному, зеленому и синему цветам. Цвет заливки можно также задать как выбранный на цветовой палитре — для этого используется свойство ColorIndex объекта Interior.

Выберем цвет заливки, записав в свойство Color значение цветовой комбинации. Для этого используем следующую процедуру.

Задание цвета заливки ячейки

```
procedure TOKBottomDlg7.Button1Click(Sender: TObject);
begin
  if not ColorDialog1.Execute then exit;
  Interior.Color:=ColorDialog1.Color;
end;
```

Результат выполнения этой процедуры представлен на рис. 12.27.

Узор заливки области ячейки определяется свойством Pattern, а его цвет — свойством PatternColor объекта Interior. Цвет также можно выбрать на цветовой палитре Excel путем записи в свойство PatternColorIndex индекса выбранного цвета.

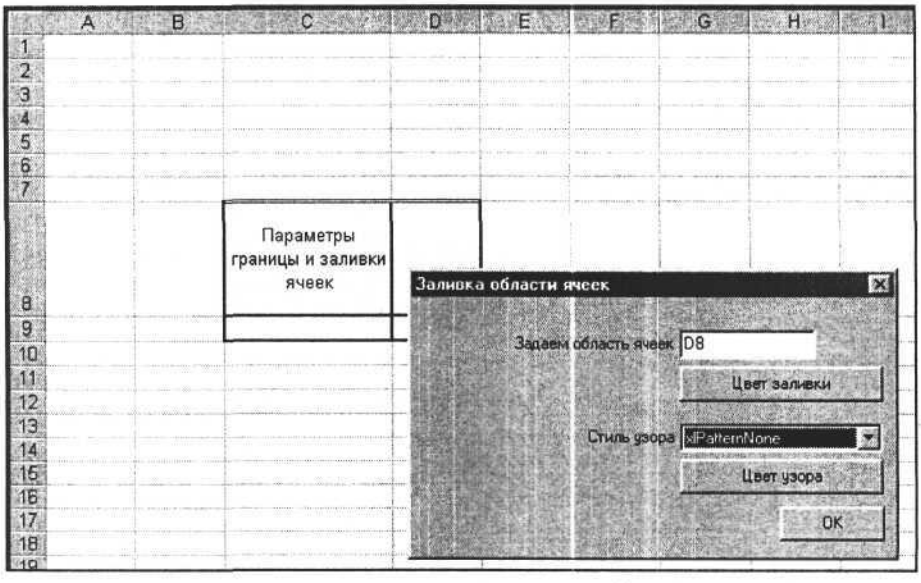


Рис. 12.27. Выбор цвета заливки ячейки

Выбор узора заливки ячейки

```
procedure TOKBottomDlg7.PatternChange(Sender: TObject);
begin
  case Pattern.ItemIndex of
  0:Interior.Pattern:=xlPatternGrid;
  1:Interior.Pattern:=xlPatternNone;
  2:Interior.Pattern:=xlPatternCrissCross;
  end;
end;

procedure TOKBottomDlg7.Button2Click(Sender: TObject);
begin
  if not ColorDialog1.Execute then exit;
  Interior.PatternColor:=ColorDialog1.Color;
end;
```

Узор заливки определяется выбором одного из всех возможных вариантов и задается числовой константой, которая записывается в свойство Pattern. На рис. 12.28 представлен пример выбора узора ячейки и его цвета, осуществленного с помощью приведенных процедур.

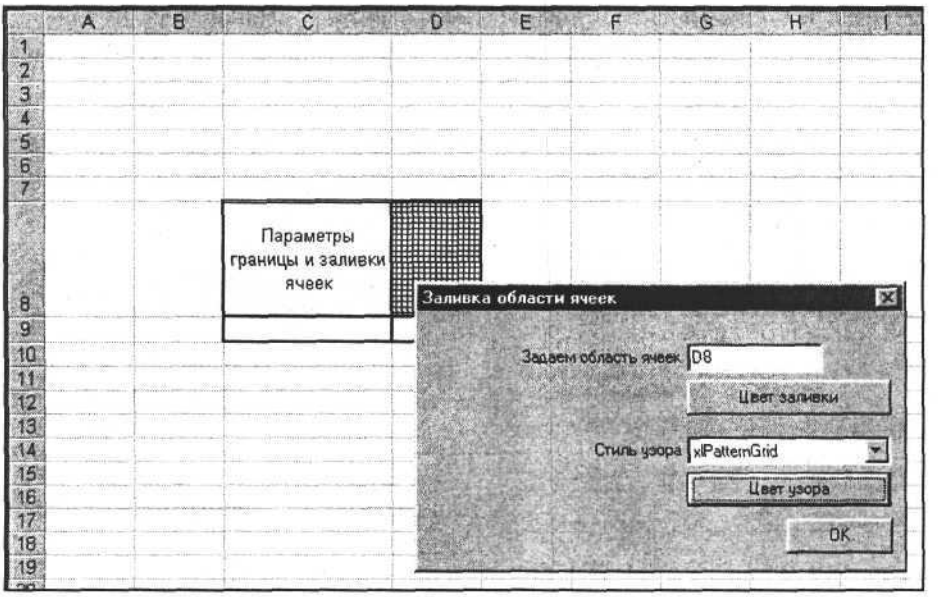


Рис. 12.28. Выбор узора заливки ячейки

Пример программы — подготовка формы налоговой декларации НДС

Применим наши знания на практике. Пользователям часто приходится заполнять различные формы в Excel. Обычно эта работа монотонна и однообразна, но необходима и очень важна. Одним из видов такой работы является заполнение форм налоговых деклараций, которых на большом предприятии очень много, а заполнять их приходится в огромном количестве за очень небольшой промежуток времени. Конечно, эту проблему можно решить, используя в формах Excel функции и небольшие макросы. Но если вы программист Delphi и программа для расчета налога разработана в этой среде, то нет никакой необходимости полностью переквалифицироваться в программиста Visual Basic — достаточно получать и воплощать новые знания, используя для решения новых задач проверенные методы и стиль программирования.

Рассмотрим пример создания формы налоговой декларации НДС. Формы, которые требуется заполнять, можно взять в любой справочно-правовой системе, например "Консультант Плюс". Для заполнения формы достаточно использовать методы поиска и замены. Но сначала подготовим форму.

Суть подготовки заключается в расстановке некоторых строковых констант в определенные ячейки. В процессе работы программы эти константы будут

заменены реальными значениями из базы данных. На рис. 12.29 представлен шаблон документа для формы налоговой декларации, подготовленный для заполнения из приложения Delphi.

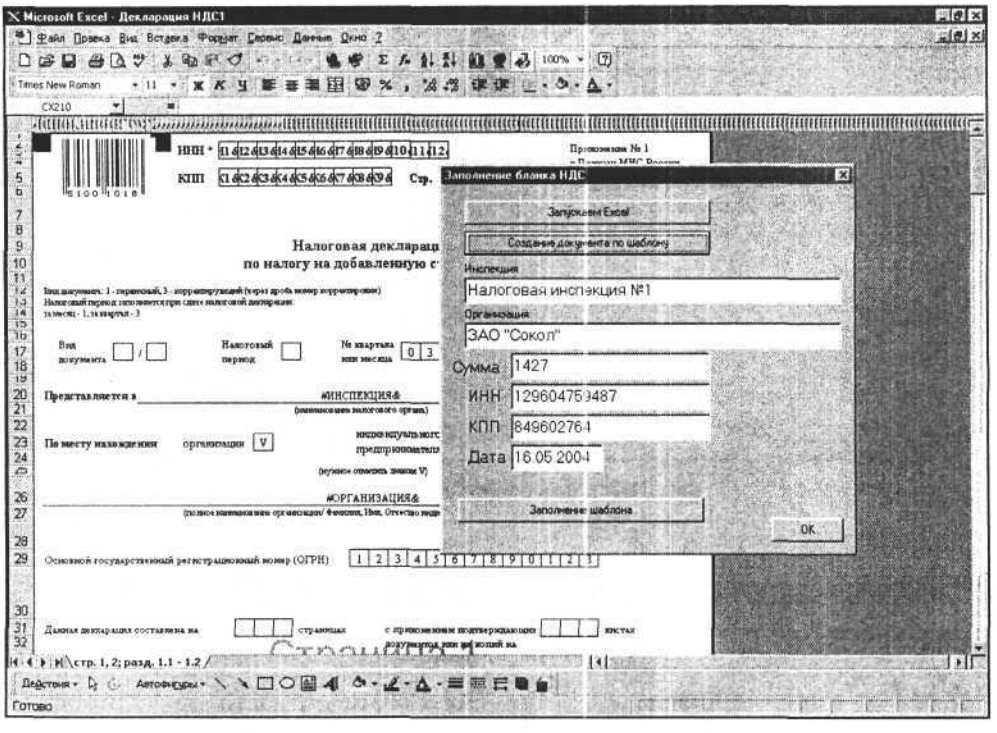


Рис. 12.29. Подготовленный шаблон документа для формы налоговой декларации

Как вы обратили внимание, некоторые ячейки шаблона документа содержат строки типа #ИНСПЕКЦИЯ& и #ОРГАНИЗАЦИЯ& или строки типа И1&, И2&... И12& и К1&, К2& ... (рис. 12.30).

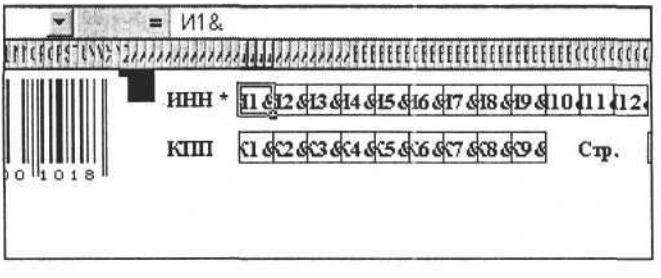


Рис. 12.30. Фрагмент шаблона документа для формы налоговой декларации

Эти строки будут использоваться для поиска и подстановки на их место реальных данных при заполнении формы налоговой декларации. Рассмотрим представленный ниже программный код, предназначенный для формирования формы. Он основан на использовании метода Replace объекта Range, который производит замену текста в рабочей книге Excel. Метод Replace вызывает ошибку выполнения, если искомый фрагмент не найден. Мы использовали это обстоятельство в функции FindAndReplace, которая возвращает значение True или False в зависимости от результата поиска и замены. Обратите внимание на то, что объект Range указывает на область A1:EL230. Это позволяет осуществлять поиск не на всем листе, а на его части, что ускоряет работу. Использование MS Office XP также существенно ускоряет поиск и замену фрагментов текста в рабочей книге Excel и подготовку формы налоговой декларации.

Заполнение формы налоговой декларации

```
function FindAndReplace(find_,rep_:string):boolean;
 var range:variant;
begin
  FindAndReplace:=False;
  if find_<>'' then begin
    try
    range:=E.Range['A1:EL230'].Replace(What:=find_,Replacement:=rep_);
    FindAndReplace:=True;
    except
    FindAndReplace:=False;
    end;
  end;
end;

procedure TOKBottomDlg11.Button3Click(Sender: TObject);
 var a_:integer;
begin
  FindAndReplace('#ИНСПЕКЦИЯ&',inspekcia.Text);
  FindAndReplace('#ОРГАНИЗАЦИЯ&',organizacia.Text);
  for a_:=1 to 12 do FindAndReplace('И'+inttostr(a_)+'&',INN.Text[a_]);
  for a_:=1 to 9 do FindAndReplace('К'+inttostr(a_)+'&',KPP.Text[a_]);
  for a_:=1 to 15 do FindAndReplace('С'+inttostr(a_)+'&',
                          (SUMMA.Text+'        ')[a_]);
  for a_:=1 to 2 do FindAndReplace('Д'+inttostr(a_)+'&',DATA.Text[a_]);
  for a_:=1 to 2 do FindAndReplace('М'+inttostr(a_)+'&',DATA.Text[a_+3]);
  for a_:=1 to 4 do FindAndReplace('Г'+inttostr(a_)+'&',DATA.Text[a_+6]);
end;
```

Результат выполнения представленных процедур, заполняющих шаблон документа для формы налоговой декларации, показан на рис. 12.31.

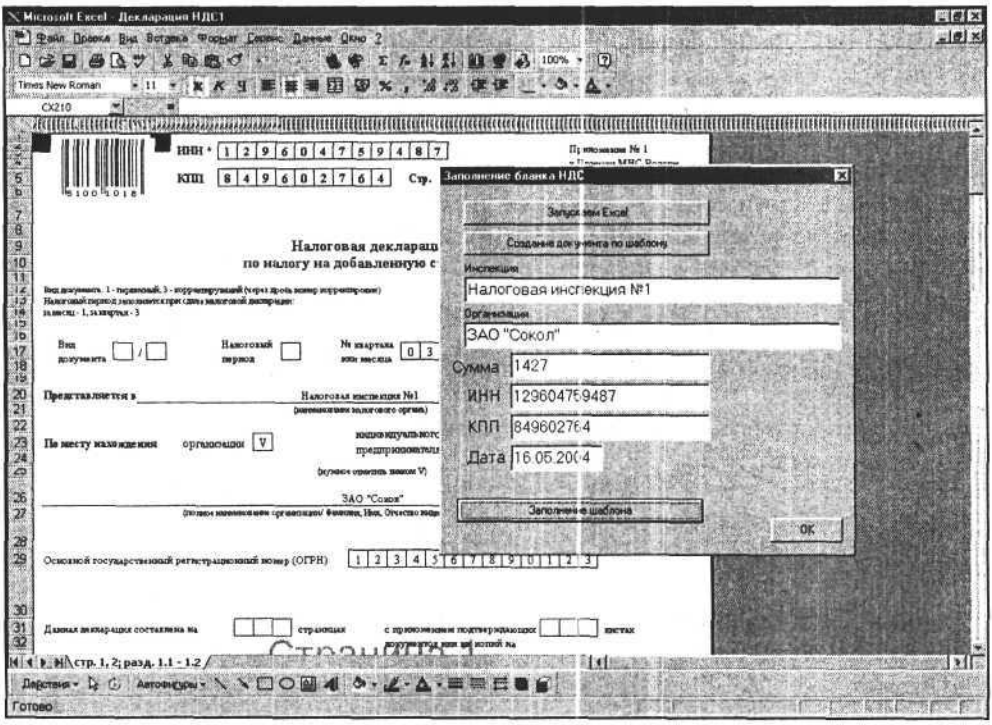


Рис. 12.31. Заполненная форма налоговой декларации

ГЛАВА 13

Работа с объектами в книге Excel

В этой главе рассмотрены следующие темы:

- коллекция объектов Shapes;
- надпись;
- выноски;
- линии;
- произвольные фигуры;
- объекты WordArt.

Коллекция объектов Shapes

Как и в документах Word, в рабочих книгах Excel информацию можно размещать не только в ячейках, но и в объектах, которые могут располагаться произвольно в любом месте листа. Речь идет об объектах коллекции Shapes, объединяющей в себе различные виды таких объектов, как надписи, линии и другие геометрические фигуры, выноски, OLE-объекты, рисунки, объекты WordArt и др. Коллекция Shapes имеет несколько свойств и методов, которые описывают ее и позволяют манипулировать ее содержимым. Свойство Count содержит информацию о количестве элементов коллекции. Метод Item(i:variant) возвращает ссылку на объект коллекции, где i — порядковый номер объекта в коллекции (начиная с 1) или имя объекта. Метод SelectAll выделяет все элементы коллекции. Методы AddCallout, AddConnector, AddCurve, AddFormControl, AddLabel, AddLine, AddOLEObject, AddPicture, AddPolyline, AddShape, AddTextbox, AddTextEffect создают новые элементы коллекции различного типа. Наиболее распространенным и типичным из элементов коллекции Shapes является объект типа TextBox (надпись). Рассмотрим его подробнее.

Надпись

Надпись (объект TextBox) создается с помощью метода AddTextbox коллекции Shapes. При вызове этого метода необходимо указать параметры создаваемого объекта — направление текста объекта, его координаты и размеры. Если метод AddTextbox вызывается из приложений Delphi, то переменные, задающие координаты и размеры, должны иметь тип Extended. Следующий пример процедуры демонстрирует использование данного метода в приложениях.

Создание надписи

```
procedure TForm1.Button3Click(Sender: TObject);
 const msoTextOrientationHorizontal=1;
 var left_,top_:Extended;
begin
  left_:=100;
  top_:=100;
  TextBox:=E.ActiveWorkBook.ActiveSheet.Shapes.AddTextBox(
       msoTextOrientationHorizontal, left_,top_, 100, 100);
end;
```

Результат выполнения представленной процедуры, если в нее внести незначительные изменения, может выглядеть примерно так, как показано на рис. 13.1. В данном примере создан объект с определенными координатами и размерами, заданными в управляющей программе. Задано горизонтальное направление текста, но при создании объекта можно выбрать любой из возможных типов ориентации (направления) текста (см. *Приложение 1*).

Даже если объект TextBox создан с определенными свойствами, это не значит, что свойства этого объекта нельзя изменить впоследствии. Для этого необходимо получить доступ к созданному объекту, а затем к свойствам самого объекта. Доступ к объекту коллекции Shapes можно получить, используя метод Item(i) данной коллекции. Например, оператор

```
Shape:=Shapes.Item(i);
```

помещает в переменную Shape ссылку на элемент коллекции, после чего мы можем изменять свойства непосредственно самого объекта.

Рассмотрим основные свойства объекта TextBox. Поскольку этот объект предназначен в основном для отображения текста, параметры шрифта, а также направление и способ выравнивания текста являются одними из основных его свойств. Внутренние поля тоже определяют положение текста внутри надписи. С использованием свойств объекта Line, принадлежащего объекту TextBox, задаются тип, толщина и цвет линий, ограничивающих

объект. Заливка надписи (цветом, узором или рисунком) задается с помощью свойств и методов объекта Fill, который тоже принадлежит объекту TextBox. Размер и положение надписи определяются ее свойствами Width, Height, Top, Left, которые в приложениях Delphi должны иметь тип Extended.

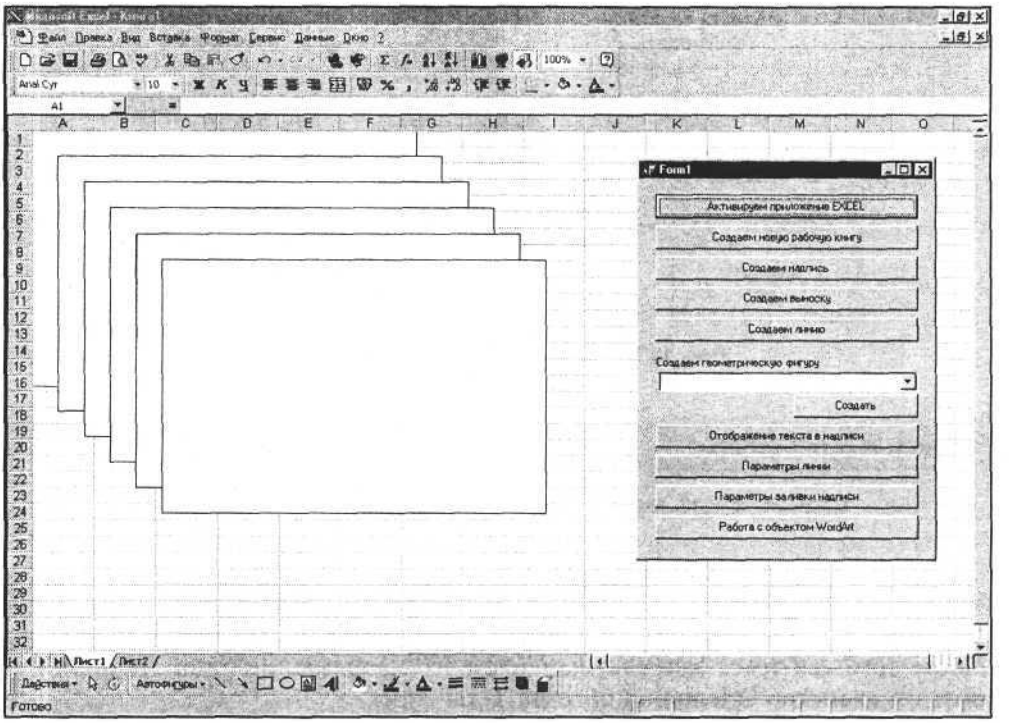


Рис. 13.1. Создание надписи

Надпись можно защитить и скрыть текст, но защита действует только тогда, когда включена защита листа.

Уже понятно, что объект TextBox представляет собой систему взаимосвязанных объектов, имеющую древовидную структуру. Для упрощения разберем только ее основные составляющие (рис. 13.2).

Из рис. 13.2 ясно, что доступ к основным визуальным свойствам надписи осуществляется через промежуточные звенья. Например, все свойства текста и сам текст содержатся в свойствах и объектах, принадлежащих объекту TextFrame. Попробуем изменить некоторые его характеристики. Очевидно, чтобы отследить изменения текста, необходимо записать сам текст. Для этого используем свойство Text, в которое можно записать или из которого можно прочитать текст надписи. Обращаем внимание на то, что свойство Text принадлежит объекту Characters, а не объекту TextFrame.

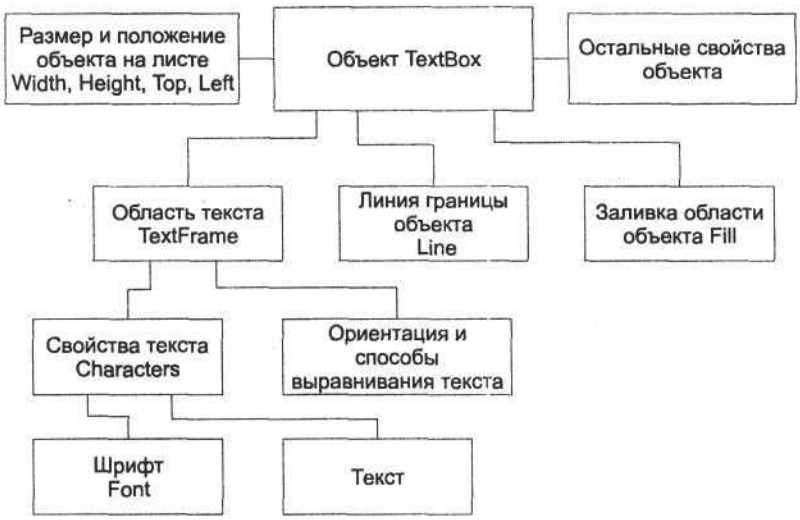


Рис. 13.2. Объектная модель объекта TextBox

Запись текста надписи

```
procedure TOKBottomDlg2.Memo1Change(Sender: TObject);
begin
  TextBox.TextFrame.Characters.Text:=Memo1.Text;
end;
```

После того как текст записан, изменяем его направление (ориентацию), используя свойство Orientation объекта TextFrame.

Изменение направления текста

```
procedure TOKBottomDlg2.OrientationChange(Sender: TObject);
begin
  TextBox.TextFrame.Orientation:=Orientation.ItemIndex+1;
end;
```

Дополнительно, для изменения положения текста можно воспользоваться свойствами, задающими выравнивание текста и его поля (исходный текст приложения представлен на сопроводительном компакт-диске книги). Результат применения различных вариантов ориентации и выравнивания текста представлен на рис. 13.3.

Установка необходимого шрифта осуществляется настройкой определенных свойств объекта Font. В предыдущей главе мы рассматривали, как он выби-

рается применительно к тексту ячейки. Для других объектов процедура аналогична.

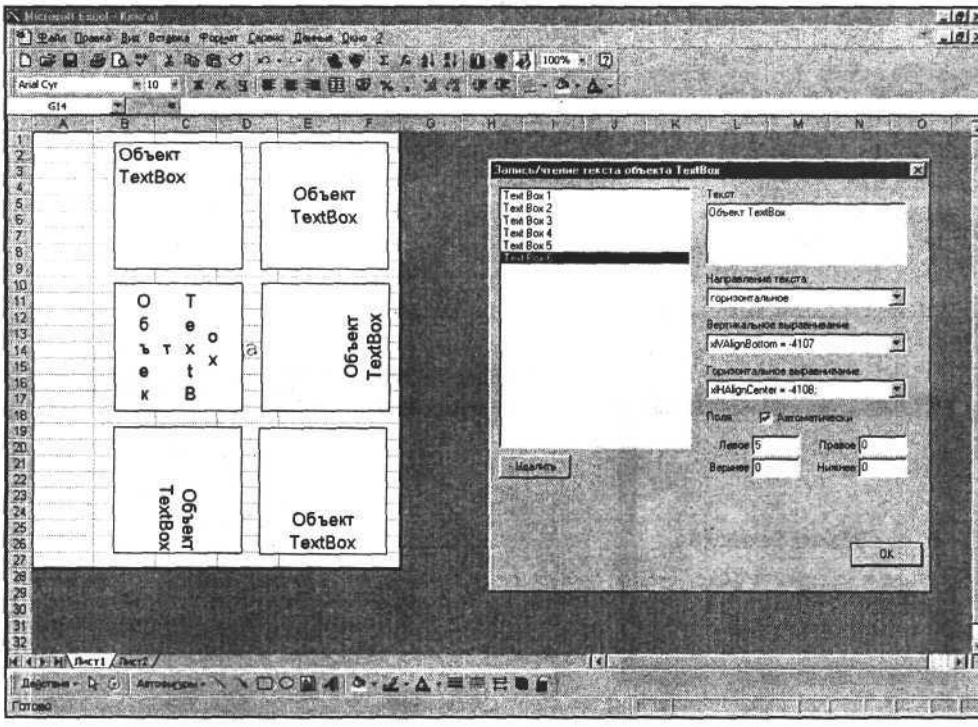


Рис. 13.3. Варианты ориентации и выравнивания текста в TextBox

Линии границы

Параметры линии границы объекта определяются свойствами объекта Line, который сам является свойством и принадлежит объекту TextBox. При создании надписи она имеет видимые линии границы с определенными параметрами, но у нас есть возможность сделать эти линии вообще невидимыми. Для этого достаточно изменить состояние свойства Visible объекта Line — установить его в значение False.

Задание видимости линии границы

```
procedure TOKBottomDlg3.LineVisibleClick(Sender: TObject);
begin
  Line.Visible:=LineVisible.Checked;
end;
```

Задать толщину линии можно с помощью свойства Weight объекта Line. В приложениях Delphi значение толщины линии должно задаваться дробным числом типа Extended. Следующий программный код отражает изменение толщины линии (рис. 13.4).

Задание толщины линии границы

```
procedure TOKBottomDlg3.WeightChange(Sender: TObject);
begin
  Line.Weight:=Weight.Value*0.25;
end;
```

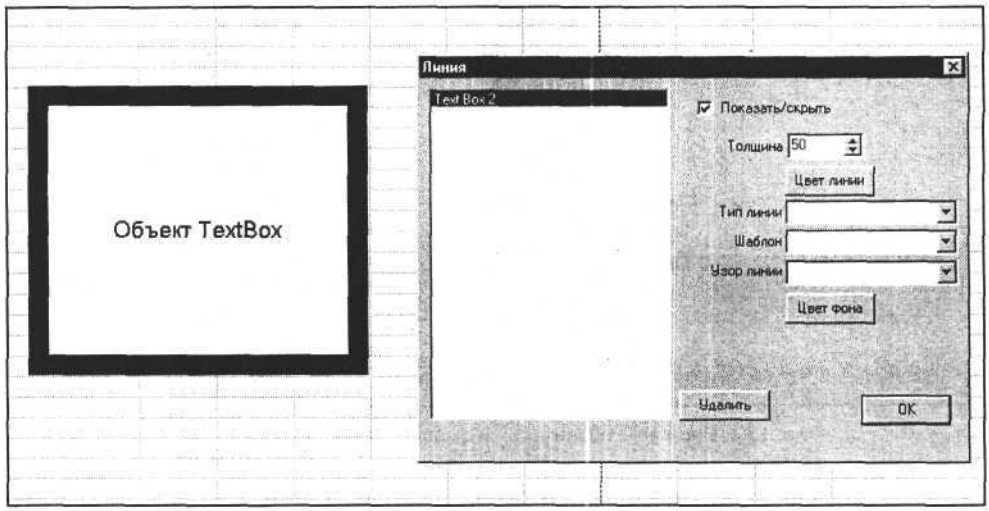


Рис. 13.4. Задание толщины линии границы для надписи

Цвет линии границы задается выбором из палитры цветов Excel, при этом значение индекса цвета записывается в свойство ForeColor.SchemeColor. Например, оператор

```
Line.ForeColor.SchemeColor:=1;
```

определяет для линии цвет как значение, соответствующее выбранному в палитре цвету. Такой метод определения цвета зависит от палитры и ее изменений. Если требуется задать цвет как комбинацию трех цветов (красный, зеленый, синий), то используем запись значения цвета в формате RGB непосредственно в свойство ForeColor.RGB объекта Line.

Изменим для надписи цвет линии границы, используя формат RGB.

Задание цвета линии границы

```
procedure TOKBottomDlg3.ForeColorClick(Sender: TObject);
begin
  if not ColorDialog1.Execute then exit;
  Line.ForeColor.RGB:=ColorDialog1.Color;
end;
```

Результат выполнения процедуры изменения цвета представлен на рис. 13.5.

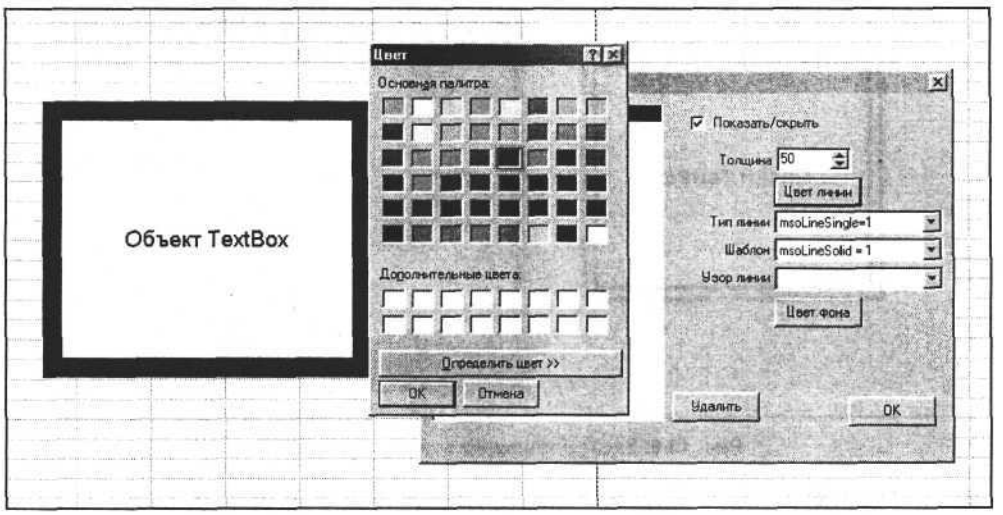


Рис. 13.5. Изменяем цвет линии границы надписи

Следующее свойство линии границы, которое мы изменим, это ее тип (или стиль). Тип линии определяется значением, которое содержит свойство Style объекта Line. Свойство Style линии границы надписи может принимать целые числовые значения из списка определенных констант, значения которых находятся в диапазоне от 1 до 5. Их применение определяет, будет линия сплошной или сочетанием из двух-трех толстых и тонких линий.

Рассмотрим следующую процедуру, позволяющую изменять тип линии.

Задание типа линии границы

```
procedure TOKBottomDlg3.LineStyleChange(Sender: TObject);
begin
  case LineStyle.ItemIndex of
  0:Line.Style:=msoLineSingle;
  1:Line.Style:=msoLineThinThin;
```

```
  2:Line.Style:=msoLineThinThick;
  3:Line.Style:=msoLineThickThin;
  4:Line.Style:=msoLineThickBetweenThin;
  end;
end;
```

Результат выполнения этой процедуры и применения типа линии, заданного константой msoLineThinThin=2, представлен на рис. 13.6.

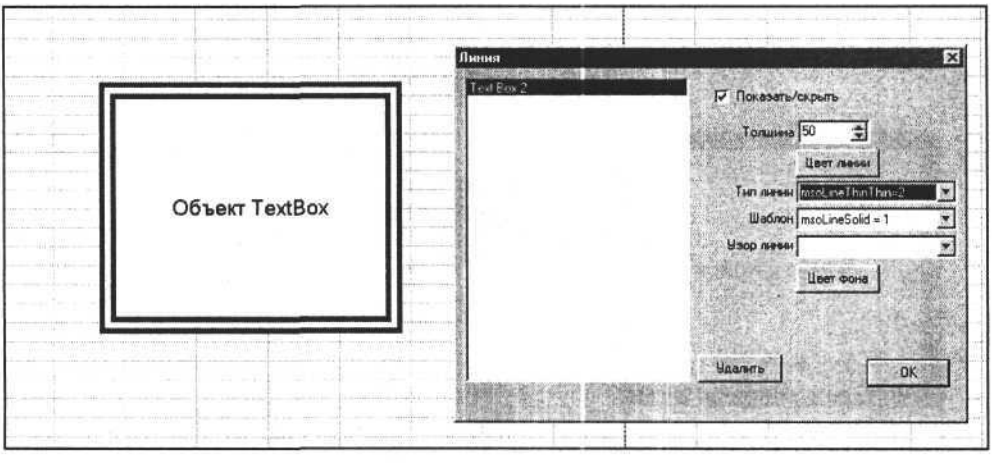


Рис. 13.6. Задаем тип линии границы для надписи

Если задать тип линий константами msoLineThickBetweenThin и msoLineThickThin, то получим другой результат (рис. 13.7).

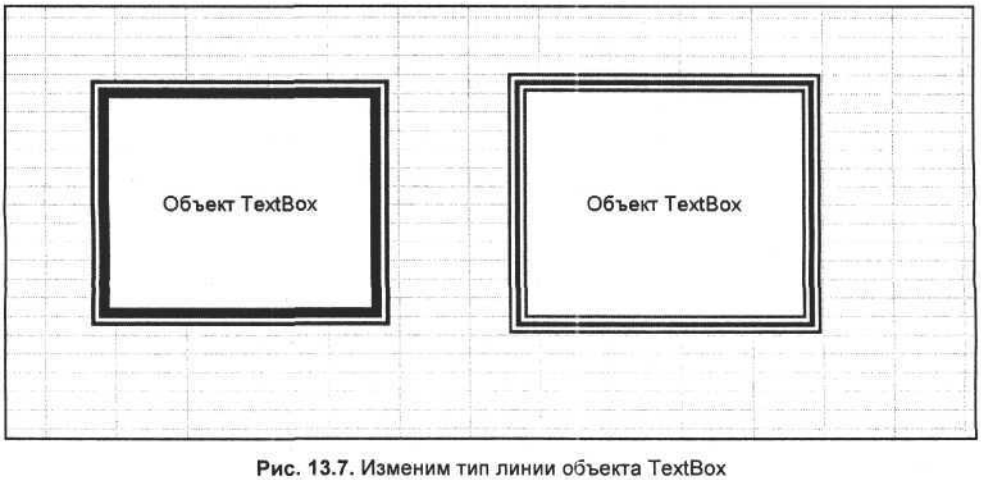


Рис. 13.7. Изменим тип линии объекта TextBox

Независимо от типа, линия границы может быть представлена в виде точек, пунктирной линии или сочетания того и другого. Шаблон (вид разрыва линии) определяется значением свойства DashStyle объекта Line. Рассмотрим пример.

Задание шаблона линии границы

```
procedure TOKBottomDlg3.LineDashStyleChange(Sender: TObject);
begin
  Line.DashStyle:=LineDashStyle.ItemIndex+1;
end;
```

В этом примере мы изменяем свойство DashStyle, присваивая ему значения констант из заданного списка. Все эти константы являются целыми числами в диапазоне от 1 до 8. Этим объясняется простота исходного текста для процедуры выбора шаблона линии границы.

Установим шаблон линии с помощью этой процедуры, задав для свойства DashStyle значение константы msoLineSquareDot. Результат выполнения процедуры представлен на рис. 13.8.

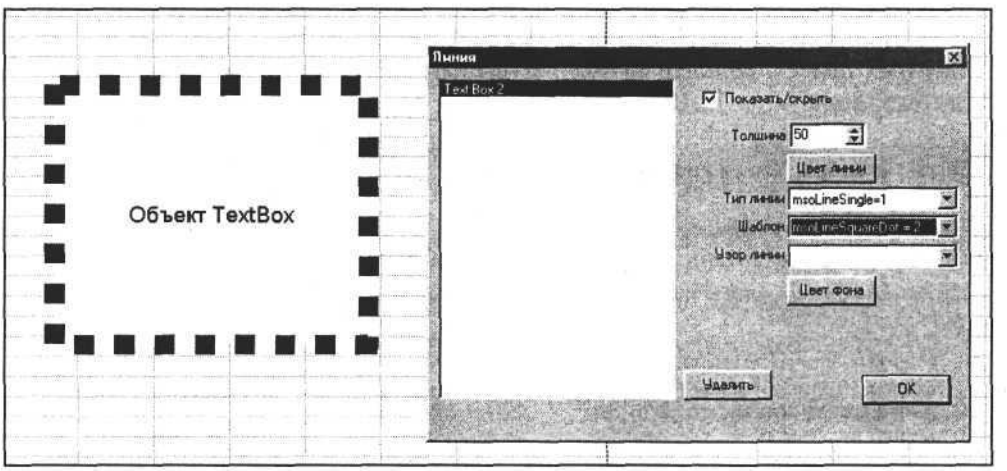


Рис. 13.8. Задаем шаблон линии границы для надписи

Воспользовавшись той же процедурой, изменим шаблон, присваивая значения других констант свойству DashStyle объекта Line. Результаты этих преобразований представлены на рис. 13.9.

Если по каким-то причинам нас не устраивает вид линии границы, получаемый комбинированием значений свойств Style и DashStyle объекта Line, то дополнительно можно воспользоваться свойством Pattern объекта Line и

задать нужный рисунок узора. Сочетая различные значения этих трех свойств, можно получить множество вариантов и выбрать среди них подходящий.

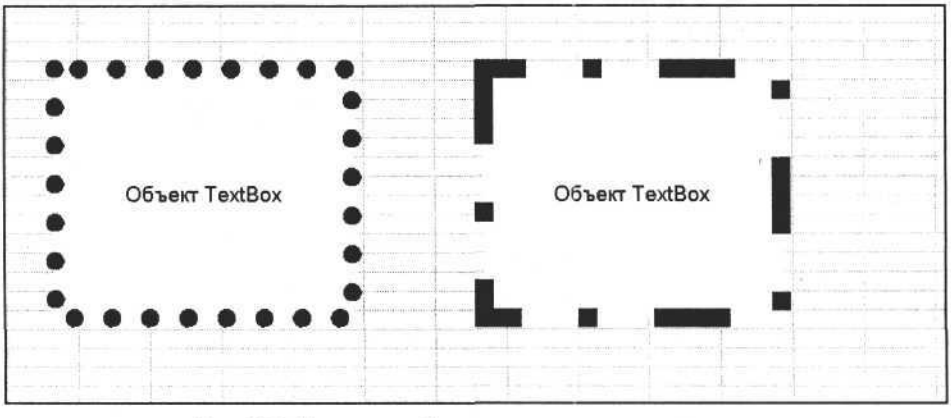


Рис. 13.9. Изменим шаблон линии границы для надписи

При задании узора появляется дополнительная возможность манипуляции с цветом. Дополнительно к цвету линии (узора) можно задать цвет фона. Цвет фона определяется свойством BackColor объекта Line. Первая из приведенных далее процедур позволяет выбрать узор линии, а вторая — задать цвет его фона (рис. 13.10 и 13.11).

Задание узора линии границы

```
procedure TOKBottomDlg3.PatternChange(Sender: TObject);
begin
  Line.Pattern:=msoPatternHorizontalBrick;
end;
```

Задание цвета фона узора

```
procedure TOKBottomDlg3.BackColorClick(Sender: TObject);
begin
  Line.BackColor.RGB:=ColorDialog1.Color;
end;
```

В процессе работы программы мы всегда можем изменить узор линии границы надписи. Для этого достаточно свойству Pattern присвоить новое значение из списка допустимых значений (более 40), которое является целым числом.

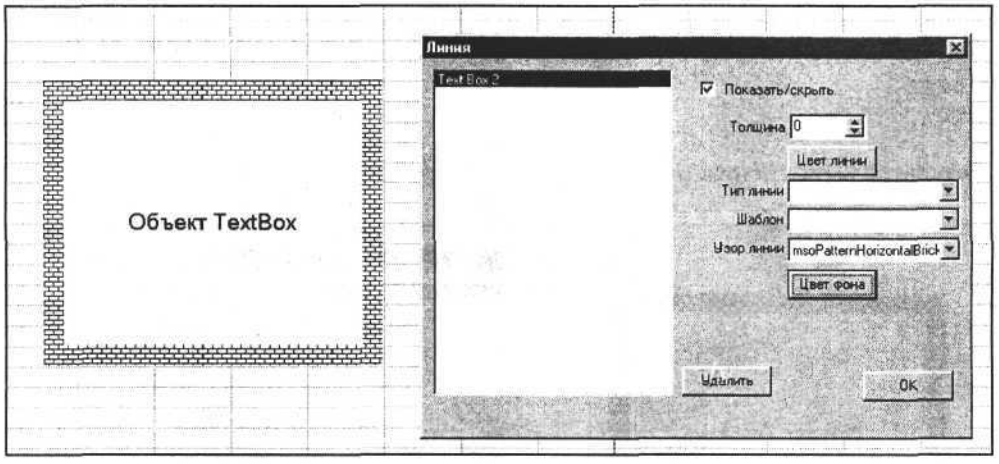


Рис. 13.10. Задаем узор линии границы для надписи

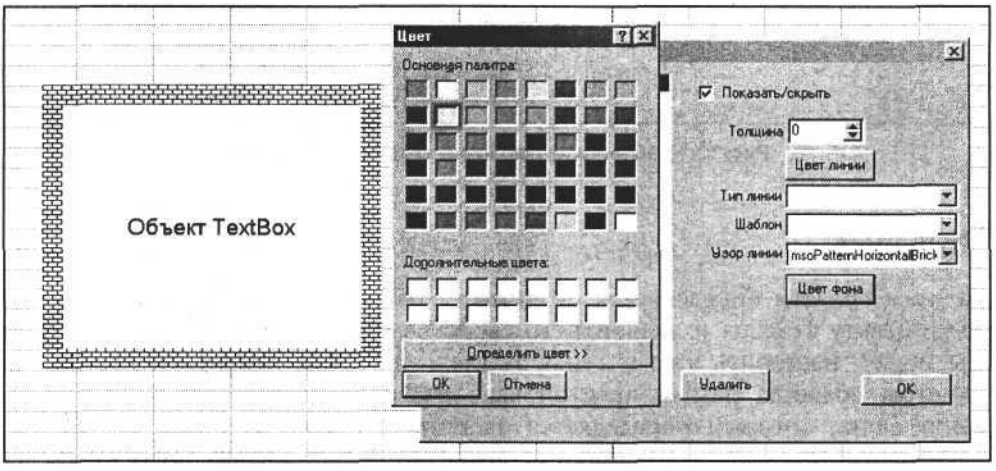


Рис. 13.11. Задаем цвет фона узора для линии границы

Следующая процедура позволяет изменить узор.

Изменение узора линии границы

```
procedure TOKBottomDlg3.PatternChange(Sender: TObject);
begin
  case Pattern.ItemIndex of
  0:Line.Pattern:=msoPatternDottedGrid;
  1:Line.Pattern:=msoPatternHorizontalBrick;
  2:Line.Pattern:=msoPatternLargeCheckerBoard;
```

```
  3:Line.Pattern:=msoPatternLargeConfetti;
  4:Line.Pattern:=msoPatternLargeGrid;
  end;
end;
```

Результат выполнения процедуры представлен на рис. 13.12.

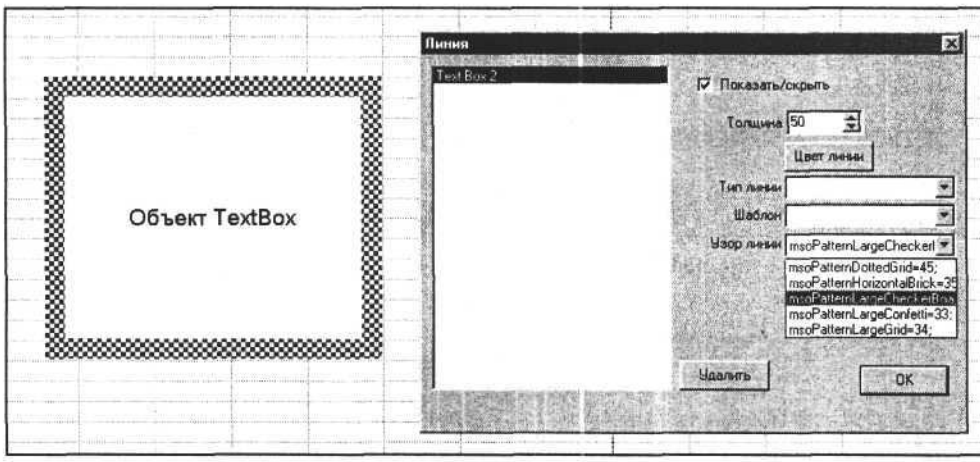


Рис. 13.12. Изменяем узор линии границы для надписи

Заливка

Заливка области определяется свойствами объекта Fill, который принадлежит объекту TextBox и является его свойством. Манипулируя объектом Fill, мы можем настроить цвет и стиль заливки, фоновый рисунок и другие параметры области. Самый простой вариант заливки заключается в выборе цвета фона, который определяется свойством ForeColor объекта Fill. Для более сложного варианта заливки необходимо использование других свойств и методов объекта Fill, часть из которых представлены в табл. 13.1.

***Таблица 13.1.** Свойства и методы объекта Fill*

Свойство или метод	Тип	Назначение
Type	Integer	Определяет тип заливки — одноцветная или градиентная заливка, узор или рисунок
Visible	Boolean	True/False — показать/скрыть заливку
Transparency	Extended	0..1 — определяет прозрачность заливки
BackColor	Integer	Определяет цвет заливки

Таблица 13.1 *(окончание)*

Свойство или метод	Тип	Назначение
Solid	Метод	Сбрасывает пользовательские настройки заливки и задает однотонную заливку
GradientStyle	Integer	Определяет направление градиента и тип штриховки
GradientVariant	Integer	Выбор варианта градиента для данного типа штриховки
GradientColorType	Integer	Определяет для выбранного типа штриховки градиента один или два цвета, или заготовку
GradientDegree	Extended	Определяет яркость двух цветов для градиентной заливки
PresetGradientType	Integer	Задает одну из доступных заготовок градиентов
OneColorGradient	Метод	Создает одноцветную градиентную заливку
TwoColorGradient	Метод	Создает двухцветную градиентную заливку
PresetGradient	Метод	Создает один из вариантов выбранной заготовки градиентной заливки
Pattern	Integer	Определяет тип узора для заливки
ForeColor	Integer	Определяет цвет фона для заливки узором
Patterned	Метод	Задает тип узора для заливки
PresetTextured	Метод	Задает для текстурной заливки образец текстуры
UserTextured	Метод	Задает для текстурной заливки рисунок из графического файла
UserPicture	Метод	Задает графический файл для заливки в виде рисунка

Задание цвета заливки

```
procedure TOKBottomDlg4.ForeColorClick(Sender: TObject);
begin
  Fill.ForeColor.RGB:=ColorDialog1.Color;
end;
```

Представленная процедура устанавливает для надписи одноцветную заливку (рис. 13.13).

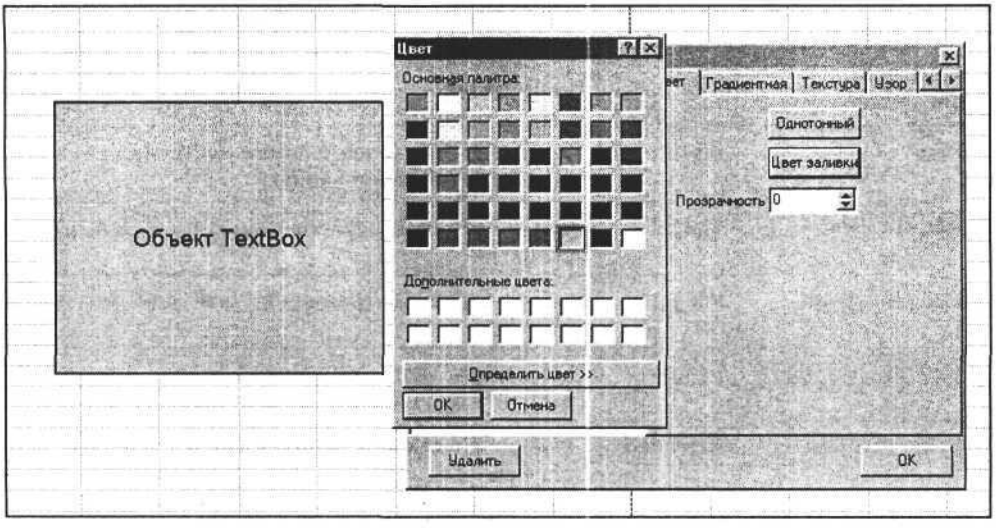


Рис. 13.13. Устанавливаем цвет заливки для надписи

Изменим значение свойства Transparency объекта Fill. Это придаст надписи эффект прозрачности по отношению к листу рабочей книги Excel. Значение свойства Transparency должно иметь тип Extended и находиться в пределах от 0,00 до 1,00. Следующая процедура и результат ее выполнения (рис. 13.14) демонстрируют этот эффект.

Задание прозрачности заливки

```
procedure TOKBottomDlg4.TransparencyChange(Sender: TObject);
begin
  Fill.Transparency:=Transparency.Value*0.01;
end;
```

При градиентной заливке получается плавный переход между двумя и более тонами одного цвета или разными цветами. При этом дополнительно задаются тип штриховки градиента и варианты ее исполнения из числа возможных.

Для градиентной заливки с одним цветом используем метод OneColorGradient, первый аргумент которого задает тип, а второй — вариант штриховки. Последний аргумент имеет тип Extended и определяет яркость фона заливки. Двухцветная градиентная заливка области выполняется методом TwoColorGradient, первый аргумент которого определяет тип, а второй — вариант штриховки.

Использование методов OneColorGradient и TwoColorGradient представлено на следующем примере.

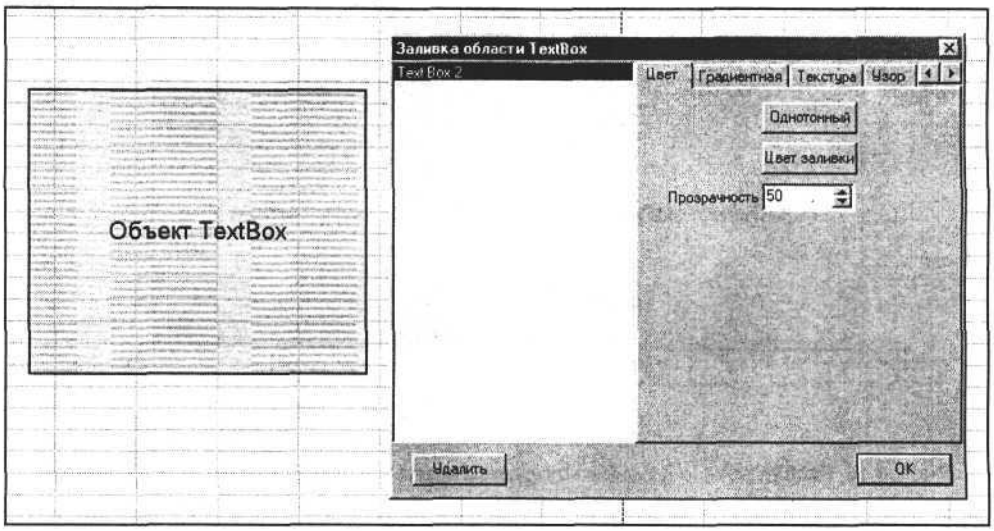


Рис. 13.14. Эффект прозрачности заливки для надписи

Задание одноцветной и двухцветной градиентной заливки

```
procedure TOKBottomDlg4.GradientStyleChange(Sender: TObject);
begin
  Fill.OneColorGradient(msoGradientHorizontal,Variant.ItemIndex+1,
          Degree.Position*0.01);
end;

procedure TOKBottomDlg4.GradientStyleChange(Sender: TObject);
begin
  Fill.TwoColorGradient(msoGradientHorizontal,Variant.ItemIndex+1);
end;
```

На рис. 13.15 представлены два варианта заливки надписи, выполненные с применением описанных процедур. Левый объект имеет одноцветную, правый — двухцветную градиентную заливку.

Выбор градиента из списка заготовок осуществляется с помощью метода PresetGradient, первый и второй аргументы которого также определяют тип и вариант штриховки. Вариант штриховки определяется числом в диапазоне от 1 до 4. Последний аргумент определяет заготовку из списка MS Office.

Используя метод PresetGradient в приложении Delphi, выберем заготовку градиента с помощью следующей процедуры. Полный исходный текст приложения представлен на сопроводительном компакт-диске книги.

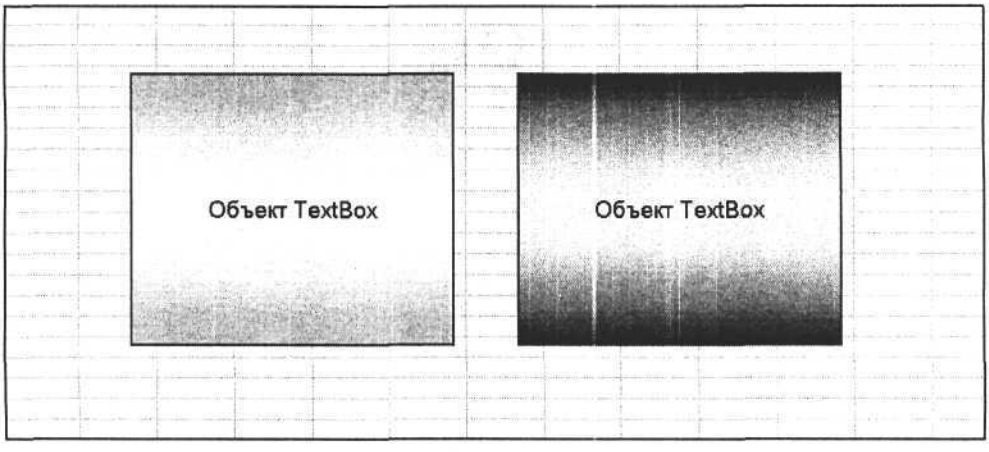


Рис. 13.15. Использование одноцветной и двухцветной градиентной заливки для надписи

Выбор заготовки градиентной заливки из списка

```
procedure TOKBottomDlg4.GradientStyleChange(Sender: TObject);
begin
  Fill.PresetGradient(msoGradientHorizontal,Variant.ItemIndex+1,
          PresetGradientType_);
end;
```

Результат выбора варианта заготовки показан на рис. 13.16.

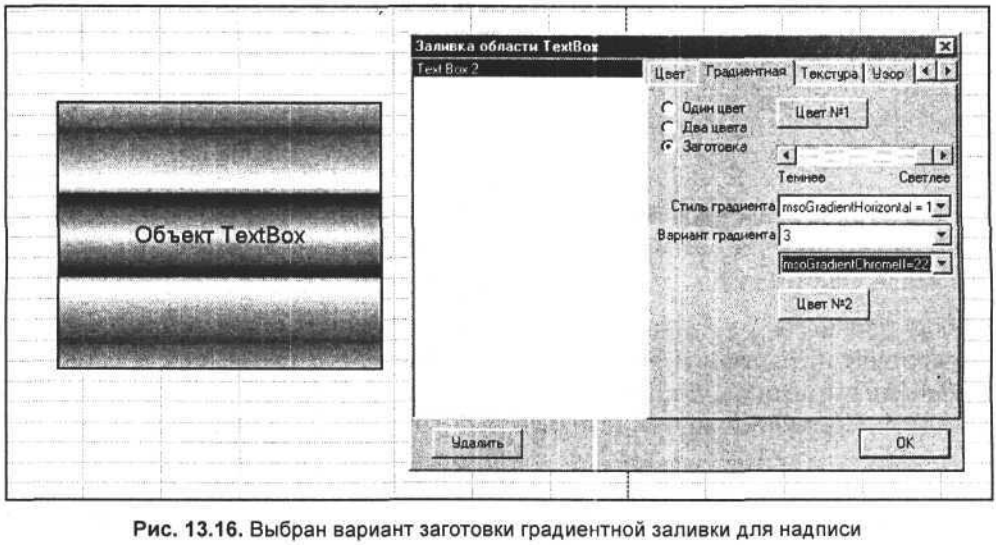


Рис. 13.16. Выбран вариант заготовки градиентной заливки для надписи

Еще один способ заливки — использование текстуры. Этот способ основан на использовании рисунка, который, многократно повторяясь, заполняет всю область объекта TextBox. Если размеры рисунка превосходят размер объекта, то заливкой объекта служит часть этого рисунка. Рисунок выбирается из списка или загружается из графического файла. Поэтому есть два метода задания текстуры:

- метод PresetTextured основан на выборе текстуры из списка и его единственный аргумент — индекс текстуры в палитре текстур;
- метод UserTextured основан на выборе и использовании в качестве текстуры рисунка из графического файла.

Рассмотрим две процедуры, применяющие эти методы.

Задание текстурной заливки

```
procedure TOKBottomDlg4.PresetTexturedChange(Sender: TObject);
begin
  case PresetTextured.ItemIndex of
  0:Fill.PresetTextured(msoTextureBrownMarble);
  1:Fill.PresetTextured(msoTextureCork);
  2:Fill.PresetTextured(msoTextureGranite);
  3:Fill.PresetTextured(msoTextureGreenMarble);
  end;
end;

procedure TOKBottomDlg4.Button4Click(Sender: TObject);
begin
  if not OpenPictureDialog1.Execute then exit;
  Fill.UserTextured(OpenPictureDialog1.FileName);
end;
```

Первая процедура позволяет выбрать один из четырех вариантов текстуры. Вторая, используя диалог выбора графического файла, загружает текстуру из файла. Варианты текстурной заливки представлены на рис. 13.17.

Использование узора в качестве заливки реализуется с помощью метода Patterned объекта Fill, единственным аргументом которого является индекс, определяющий вид узора. Для заливки узором задаются цвет узора и цвет фона (свойства ForeColor и BackColor объекта Fill, используемые и для градиентной заливки).

Следующие процедуры позволяют выбрать один из четырех способов штриховки, цвет штриховки и фона для заливки надписи узором. Результат выполнения процедур показан на рис. 13.18.

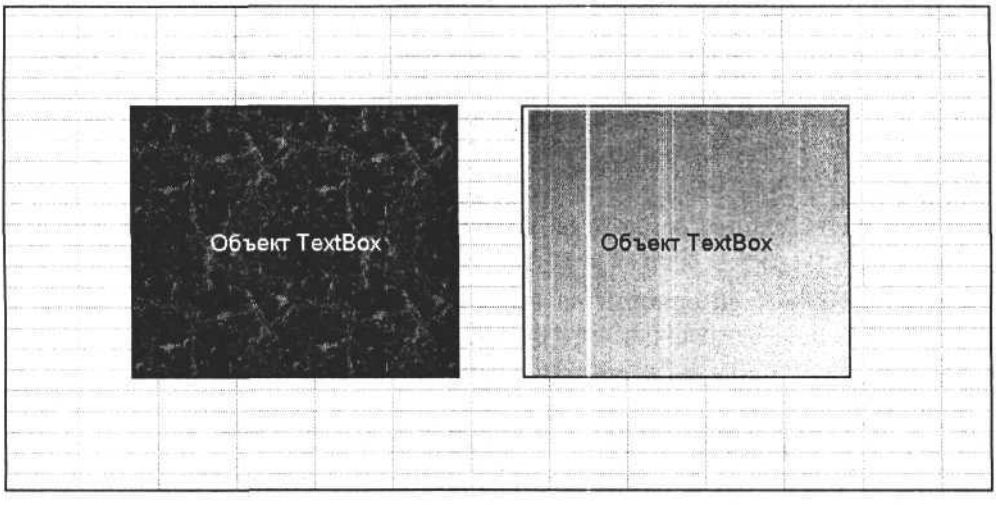


Рис. 13.17. Текстурная заливка надписи

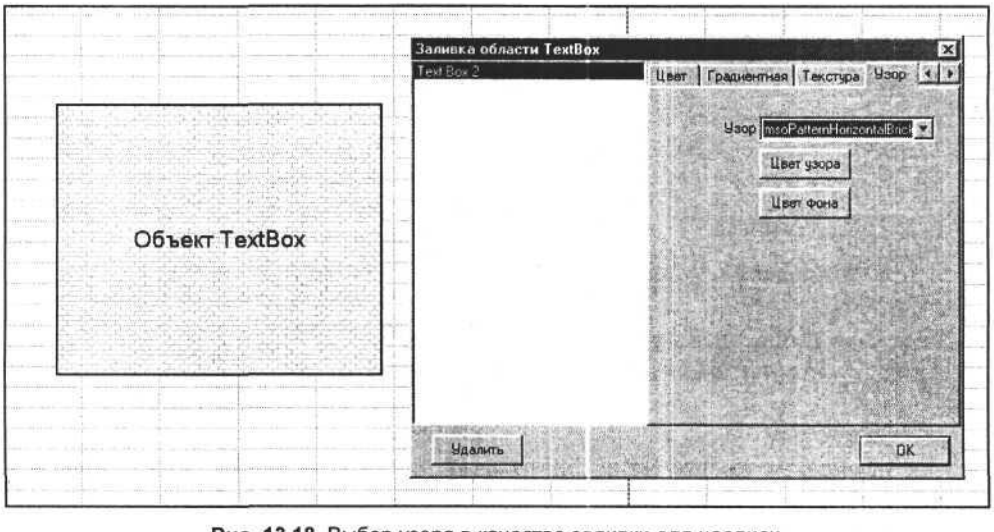


Рис. 13.18. Выбор узора в качестве заливки для надписи

Задание заливки в виде узора

```
procedure TOKBottomDlg4.PatternChange(Sender: TObject);
begin
  case Pattern.ItemIndex of
  0:Fill.Patterned(msoPatternDottedGrid);
  1:Fill.Patterned(msoPatternHorizontalBrick);
  2:Fill.Patterned(msoPatternLargeCheckerBoard);
```

```
  3:Fill.Patterned(msoPatternLargeConfetti);
  end;
end;

procedure TOKBottomDlg4.Button6Click(Sender: TObject);
begin
  Fill.ForeColor.RGB:=ColorDialog1.Color;
end;

procedure TOKBottomDlg4.BackColorClick(Sender: TObject);
begin
  Fill.BackColor.RGB:=ColorDialog1.Color;
end;
```

Рассмотрим способ заливки, при котором фоном для надписи является рисунок. Метод UserPicture объекта Fill позволяет реализовать этот способ. Аргумент данного метода содержит путь и имя графического файла.

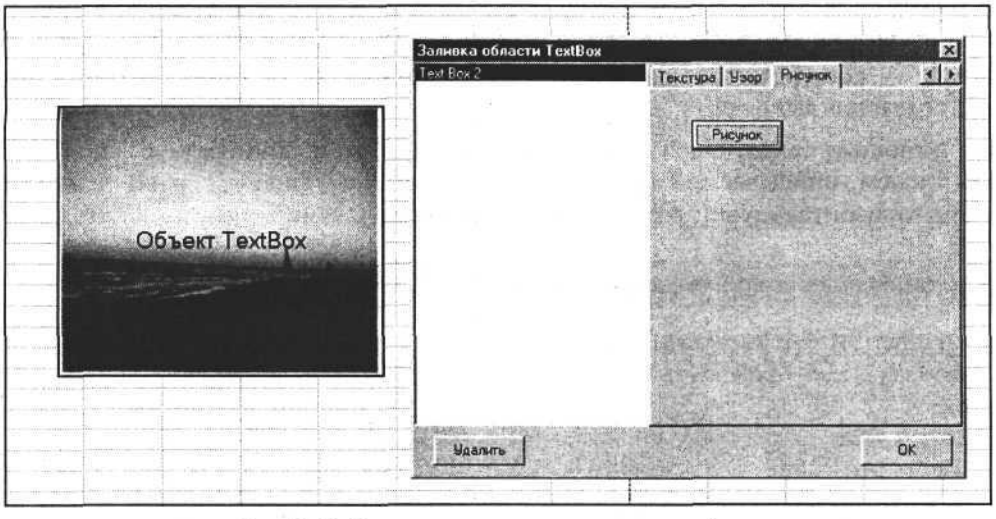


Рис. 13.19. Использование рисунка в качестве фона

Следующая процедура, применяющая метод UserPicture, позволяет выбрать рисунок для заливки.

Задание заливки в виде рисунка

```
procedure TOKBottomDlg4.Button5Click(Sender: TObject);
begin
  if not OpenPictureDialog1.Execute then exit;
```

```
  Fill.UserPicture(OpenPictureDialog1.FileName);
end;
```

На рис. 13.19 представлен пример результата выполнения этой процедуры.

Выноски

По сути, выноска (объект Callout) является объектом, во многом сходным с надписью, поэтому многие их свойства идентичны. Выноска отличается от надписи дополнительной ломаной линией-указателем. У некоторых вариантов выносок нет линий, ограничивающих текстовую область. Во всех случаях выносок характеристики заливки, расположения текста и линий описываются такими же объектами и свойствами, как и для надписи (у выноски свойства линии применяются и к ломаной линии-указателю). У выноски есть дополнительный объект Adjustments для описания линии-указателя, содержащий описание параметров линии Adjustments.Item(i), где i изменяется от 1 до Adjustments.Count (общее количество точек перелома линии). Создается объект-выноска с помощью метода AddCallout(Type, Left, Top, Width, Height, Anchor), где Type:integer — тип выноски, Left, Top, Width, Height — координаты и размеры (имеют тип Extended), Anchor — область, где создается выноска.

С помощью следующей процедуры мы создаем выноску с линией-указателем, описываемой тремя точками (msoCalloutThree=3), а затем изменяем горизонтальную и вертикальную координаты конца линии-указателя.

Создание выноски и изменение координат конца линии-указателя

```
procedure TForm1.Button8Click(Sender: TObject);
 var left_,top_:extended;
begin
  left_:=40;
  top_:=40;
  Callout:=E.ActiveWorkBook.ActiveSheet.Shapes.AddCallout(msoCalloutTwo,
                                                  left_,top_, 150, 100);
  Callout.Adjustments.Item(1):=-0.1;  // Горизонтальное смещение
  Callout.Adjustments.Item(2):=2;     // Вертикальное смещение
end;
```

На рис. 13.20 показано, как стала выглядеть выноска, после того как она была создана, а затем были изменены ее координаты конца линии-указателя и заливка (при создании заливка имеет свойства "по умолчанию"), при этом использовались общие для объектов Shapes методы и свойства, описанные ранее.

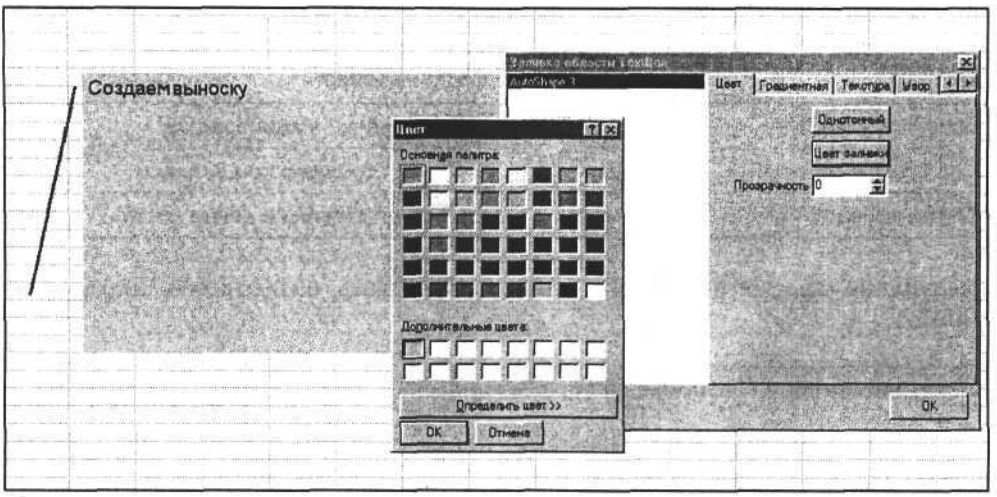


Рис. 13.20. Использование рисунка в качестве фона

Линии

Линия в рабочей книге Excel создается с использованием метода AddLine коллекции Shapes. Аргументами метода AddLine являются начальные и конечные координаты (BeginX, BeginY, EndX и EndY типа Extended). Толщина, цвет и другие характеристики линии задаются, в основном, так же, как для линии границы надписи (табл. 13.2).

***Таблица 13.2.** Свойства линии*

Свойство	Тип	Назначение
Visible	Boolean	Отобразить/скрыть линию
Weight	Integer	Толщина линии
ForeColor.RGB	TColor	Цвет линии
BackColor.RGB	TColor	Цвет фона (для узора)
Style	Integer	Тип линии
DashStyle	Integer	Шаблон линии
Pattern	Integer	Узор
BeginArrowheadStyle	Integer	Вид стрелки в начале линии
BeginArrowheadLength	Integer	Длина стрелки в начале линии
BeginArrowheadWidth	Integer	Ширина стрелки в начале линии

Таблица 13.2 *(окончание)*

Свойство	Тип	Назначение
EndArrowheadStyle	Integer	Вид стрелки в конце линии
EndArrowheadLength	Integer	Длина стрелки в конце линии
EndArrowheadWidth	Integer	Ширина стрелки в конце линии

Следующие процедуры Delphi позволяют создать и настроить параметры линии.

Создание линии и настройка ее параметров

```
procedure TForm1.Button9Click(Sender: TObject);
begin
  Line:=Shapes.AddLine(10,10,300,250);
end;

procedure TForm1.ForeColorClick(Sender: TObject);
begin
  Line.ForeColor.RGB:=ColorDialog1.Color;
  Line.Weight:=2.5;
  Line.DashStyle:= msoLineRoundDot;
end;
```

Результат выполнения процедур представлен на рис. 13.21.

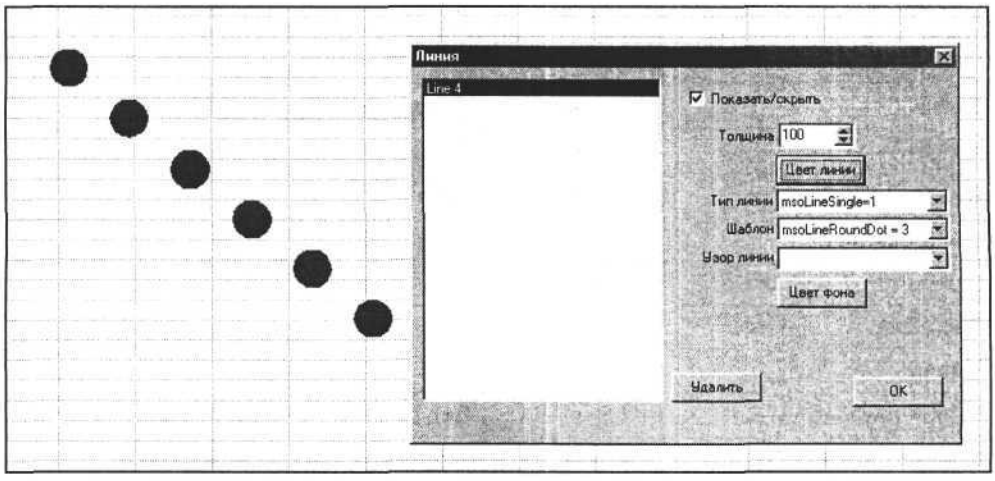


Рис. 13.21. Настройка свойств линии

Произвольные фигуры

Метод AddShape коллекции Shapes предназначен для создания произвольных объектов Shape, в том числе геометрических фигур. Прямые, ломаные и кривые линии создаются посредством специально предназначенных для этого методов, часть которых мы рассмотрели ранее. Синтаксис вызова метода AddShape:

```
AddShape:=AddShape(Type, Left, Top, Width, Height);
```

Первый аргумент представляет собой целое число и определяет тип создаваемого объекта, например Type:=msoShapeOvalCallout; соответствует овальной выноске. Число всех возможных типов геометрических фигур составляет несколько десятков. Величины этих констант можно найти в редакторе Visual Basic. Значения аргументов Left, Top, Width, Height определяют положение и размеры создаваемого объекта и имеют тип Extended.

Создадим овальную выноску и запишем в нее текст с помощью следующих процедур.

Создание произвольного объекта Shape — выноски

```
procedure TForm1.ShapeTypeChange(Sender: TObject);
 const msoShapeOvalCallout= 107;
 var Shape:variant;
begin
Shape:=Shapes.AddShape(msoShapeOvalCallout, left_,top_, 300, 200);
Shape.TextFrame.Characters.Text:='Вставляем текст в графический объект';
end;
```

Выноска, созданная с применением метода AddShape, представлена на рис. 13.22.

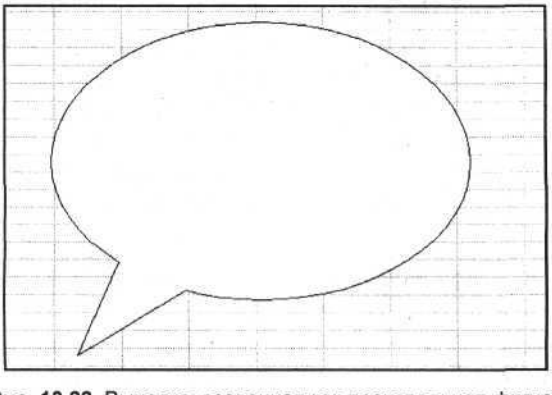


Рис. 13.22. Выноска, созданная как произвольная фигура

Текст в созданный объект добавляем с помощью тех же методов, что и при работе с объектом Callout, т. к. методы и свойства области текста (TextFrame) для двух этих объектов полностью идентичны (рис. 13.23).

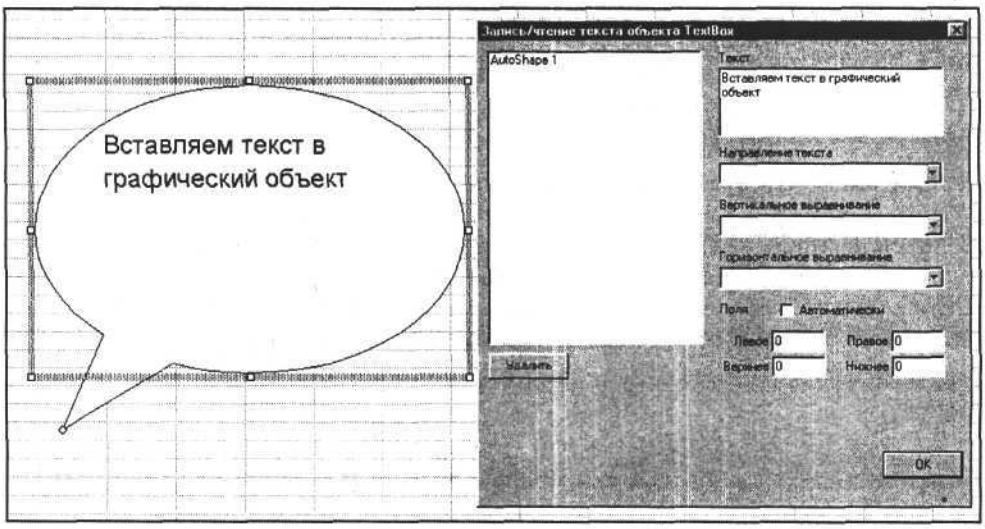


Рис. 13.23. Запись текста в созданную выноску

Объекты WordArt

Элементами коллекции Shapes могут быть объекты WordArt, создаваемые с помощью метода AddTextEffect. Синтаксис вызова метода:

```
WordArt:=Shapes.AddTextEffect(PresetTextEffect, Text, FontName, FontSize,
  FontBold, FontItalic, Left, Top);
```

где PresetTextEffect имеет тип Integer и может принимать значение константы из списка типов объекта. Аргумент Text имеет тип String и представляет собой текстовую строку. Наименование шрифта, которым должен быть отображен текст, передается как строка (аргумент FontName, тип String). Размер шрифта задается аргументом FontSize, имеющим тип Integer. Посредством аргументов FontBold и FontItalic задается начертание шрифта как комбинация значений светлый/полужирный и прямой/курсив. Аргументы Left и Top (тип Extended) определяют положение объекта относительно начала координат листа.

Метод AddTextEffect возвращает ссылку на созданный объект, используя которую можно манипулировать свойствами объекта после его создания.

Следующий пример процедуры демонстрирует использование метода AddTextEffect в приложениях Delphi; результат ее выполнения показан на рис. 13.24.

Создание объекта WordArt

```
Var WordArt:variant;
procedure TOKBottomDlg5.Button2Click(Sender: TObject);
 var Left, Top:extended;
begin
Left:=10, Top:=100;
WordArt:=Shapes.AddTextEffect(PresetTextEffect:=msoTextEffect1,
      Text:='Объект WordArt', FontName:='Tahoma', FontSize:=48,
      FontBold:=true, FontItalic:=false, Left:=Left, Top:=Top);
end;
```

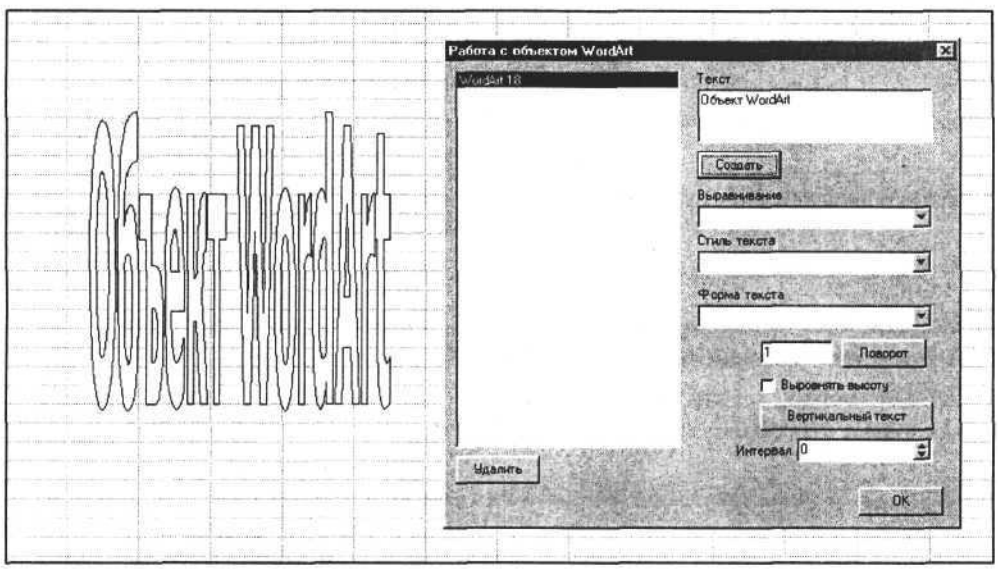


Рис. 13.24. Создаем объект WordArt

Создав объект WordArt, мы получаем доступ к его свойствам и методам, определяемым его дочерним объектом TextEffect (табл. 13.3).

Таблица 13.3. *Свойства и методы объекта TextEffect*

Свойство или метод	Тип	Назначение
Alignment	Integer	Выравнивание
FontBold	Boolean	True/False — полужирное/светлое начертание
FontItalic	Boolean	True/False — курсивное/прямое начертание

Таблица 13.3 *(окончание)*

Свойство или метод	Тип	Назначение
FontName	String	Наименование шрифта
FontSize	Integer	Размер шрифта
KernedPairs	Boolean	Кернинг пар символов[1]
NormalizedHeight	Boolean	True — выравнивание символов по высоте
PresetShape	Integer	Форма текста объекта WordArt
PresetTextEffect	Integer	Стиль объекта WordArt
RotatedChars	Boolean	Поворот символов на 90 градусов против часовой стрелки
Text	String	Содержание текста объекта WordArt
Tracking	Extended	Определяет межсимвольное расстояние; рекомендуемые значения — в диапазоне от 0,8 до 1,5
ToggleVerticalText	Метод	Переключает способ расположения символов текста от вертикального к горизонтальному и наоборот

Изменить цвет символов текста можно с помощью того же метода, что и при установке заливки надписи (объекта TextBox).

Изменим форму текста с помощью свойства PresetShape объекта TextEffect. Реализовать это позволяет следующая процедура.

Изменение формы текста для объекта WordArt

```
procedure TOKBottomDlg5.PresetShapeChange(Sender: TObject);
begin
case PresetShape.ItemIndex of
0:WordArt.TextEffect.PresetShape:=msoTextEffectShapeArchDownCurve;
1:WordArt.TextEffect.PresetShape:=msoTextEffectShapeArchDownPour;
2:WordArt.TextEffect.PresetShape:=msoTextEffectShapeArchUpCurve;
3:WordArt.TextEffect.PresetShape:=msoTextEffectShapeArchUpPour;
4:WordArt.TextEffect.PresetShape:=msoTextEffectShapeButtonCurve;
5:WordArt.TextEffect.PresetShape:=msoTextEffectShapeButtonPour;
6:WordArt.TextEffect.PresetShape:=msoTextEffectShapeCanDown;
7:WordArt.TextEffect.PresetShape:=msoTextEffectShapeCanUp;
8:WordArt.TextEffect.PresetShape:=msoTextEffectShapeCascadeDown;
```

[1] Кернинг регулирует интервалы между отдельными парами букв в зависимости от их формы.

```
9:WordArt.TextEffect.PresetShape:=msoTextEffectShapeCascadeUp;
10:WordArt.TextEffect.PresetShape:=msoTextEffectShapeCircleCurve;
end;
end;
```

Результаты изменения цвета текста и последующего применения этой процедуры представлены на рис. 13.25.

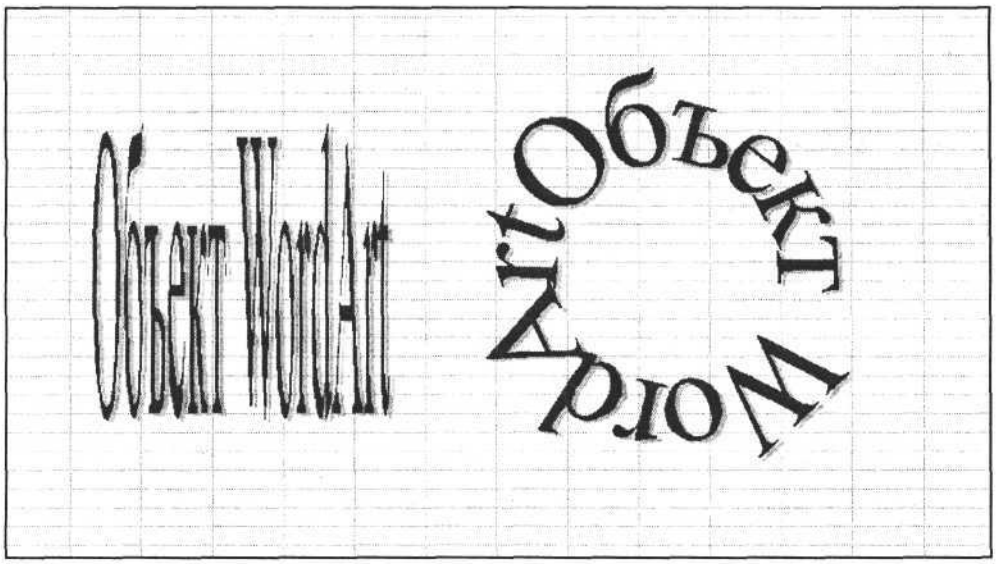


Рис. 13.25. Изменение цвета и формы текста для объекта WordArt

Затем изменим способ расположения символов текста и интервал между символами (выполним трекинг).

Изменение способа расположения символов и межсимвольного интервала

```
procedure TOKBottomDlg5.ToggleVerticalTextClick(Sender: TObject);
begin
  WordArt.TextEffect.ToggleVerticalText;
end;

procedure TOKBottomDlg5.TrackingChange(Sender: TObject);
begin
  WordArt.TextEffect.Tracking:=Tracking.Value*0.1;
end;
```

Результат выполнения этих процедур представлен на рис. 13.26.

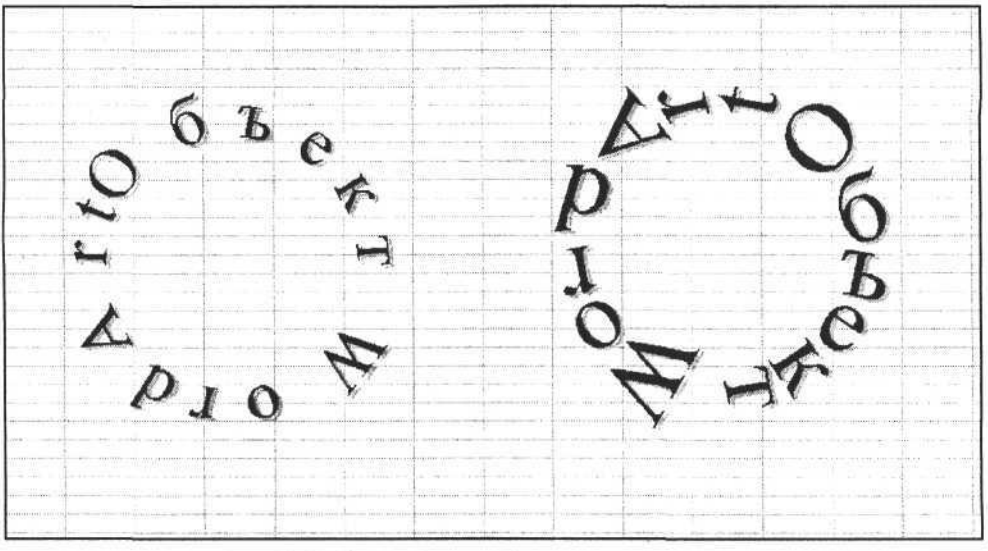


Рис. 13.26. Изменяем способ расположения символов и интервал между символами для объекта WordArt

Коллекция Shapes позволяет создавать и работать с внешними объектами (OLE-объектами), свойства и методы работы с которыми применительно к документам Word были рассмотрены в предыдущих главах. За исключением незначительных особенностей для рабочих книг Excel все они аналогичны.

ГЛАВА 14

Диаграммы в рабочей книге Excel

В этой главе рассмотрены следующие темы:

- программирование диаграмм Excel в Delphi;
- коллекция Charts, размещение диаграммы и исходных данных;
- тип диаграммы;
- объектная модель диаграммы;
- особенности некоторых типов диаграмм;
- некоторые дополнительные элементы рядов диаграмм.

Программирование диаграмм Excel в Delphi

При решении задач отображения информации мы всегда сталкиваемся с проблемой нахождения наиболее эффективного способа, при котором значительный массив данных можно быстро представить пользователю для понимания и принятия правильного решения. Одним из способов решения этой проблемы является представление информации в графическом виде, особенно в объемном. Это связано с тем, что человек живет в трехмерном мире и взаимодействует с объемными объектами, отражая в своем сознании окружающий мир как связь пространства и времени.

Очевидно, что представление набора данных в объемном виде может донести до сознания больше информации за более короткий промежуток времени. Отсюда можно сделать простой вывод: если разрабатываемое приложение позволяет отображать данные в графическом виде, то за счет этого оно обладает существенными преимуществами по сравнению с приложениями, не предоставляющими пользователю такую возможность.

Приложение MS Excel обладает целым набором инструментов отображения информации в графическом виде, часть из которых мы рассмотрели ранее. В этой главе мы рассмотрим построение диаграмм. Диаграмма на рабочем листе Excel представляет собой объект. Как все объекты Excel, она имеет некоторые общие свойства, характеризующие и другие типы объектов (например, размеры и положение), а также свойства, которыми обладают только диаграммы.

Мы рассмотрим только особые свойства диаграмм.

Коллекция Charts, размещение диаграммы и исходных данных

Все диаграммы рабочей книги объединены в коллекцию Charts, свойства и методы которой позволяют добавлять новые диаграммы, хранить, удалять и обеспечивать доступ к любой диаграмме коллекции. Коллекция Charts имеет такие же свойства, как любая другая коллекция объектов Excel, но есть и некоторые отличия, обусловленные свойствами диаграммы, т. к. диаграмма может находиться на листе вместе с другими данными или занимать отдельный лист. Поэтому диаграмма и лист рабочей книги, а также коллекции, которым они принадлежат, могут иметь общие свойства и методы, которые придется учитывать при программировании приложений[1].

Для того чтобы создать новую диаграмму, а затем манипулировать общим списком диаграмм и получать доступ к отдельной диаграмме, достаточно воспользоваться одним свойством и двумя методами коллекции Charts. Свойство Count имеет тип Integer, его значение равно количеству диаграмм в рабочей книге. Метод Add создает новую диаграмму и возвращает ссылку на нее. Метод Item() возвращает ссылку на созданную ранее диаграмму, принадлежащую коллекции Charts. Аргументом метода Item() является порядковый номер (индекс) или имя диаграммы в коллекции. Во втором случае аргументом метода Item является строка, содержащая имя диаграммы. Индексом может быть целое число со значением в диапазоне от 1 до Charts.Count.

Следующие две процедуры демонстрируют, как можно создать диаграмму, получив ссылку на нее, или получить доступ к существующей диаграмме с помощью индекса.

---

[1] Диаграмма может располагаться на отдельном листе, поэтому доступ к ней может быть таким же, как к листу. Например: если диаграмма расположена на первом листе, то ее можно удалить как лист (оператор Application.ActiveWorkbook.Sheets.Item(1).Delete;) или как диаграмму (оператор Application.ActiveWorkbook.Charts.Item(1).Delete;). Результат будет одним и тем же.

Создание диаграммы и получение доступа к существующей диаграмме

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Chart:=E.Charts.Add;
  Chart.ChartType:=xl3DColumn;
  Chart.SetSourceData(Source:=E.ActiveWorkbook.Sheets.Item[2].
                      Range['A1:F5'],PlotBy:=xlColumns);
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Chart:=E.Charts.Item[1];
end;
```

Результат выполнения первой процедуры, использующей метод Add, представлен на рис. 14.1.

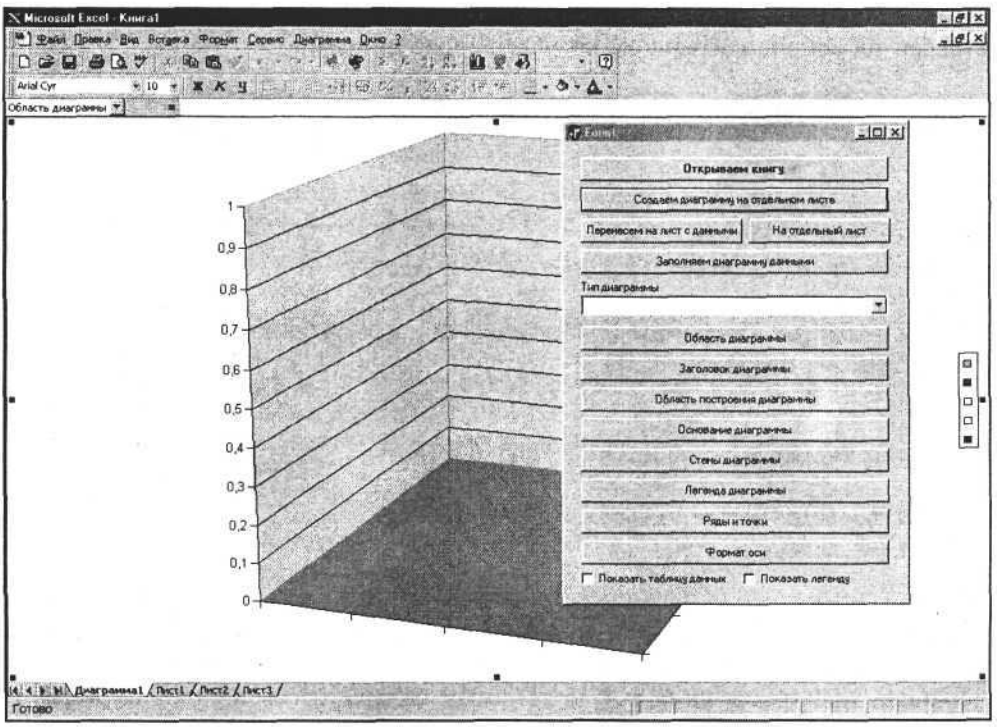


Рис. 14.1. Диаграмма, созданная на отдельном листе

Первая процедура состоит из трех операторов: первый создает диаграмму и возвращает ссылку нее, два других, используя полученную ссылку, опреде-

ляют область значений для построения диаграммы и ее тип. Без этих двух операторов мы получили бы только пустую область диаграммы. Тип диаграммы определяется значением свойства ChartType объекта Chart, имеющим целый тип и выбираемым из нескольких десятков констант. Метод SetSourceData определяет диапазон данных (область ячеек) листа рабочей книги, по которым будет строиться диаграмма. Первый аргумент этого метода определяет адрес области, второй — порядок использования ячеек в заданной области.

Важной характеристикой диаграммы также является место, где расположена сама диаграмма, определяемое с помощью метода Location объекта Chart. У этого метода два аргумента. Первый аргумент (типа Integer) определяет, будет диаграмма находиться на отдельном листе или на обычном листе с данными. Если выбран второй вариант, то второй аргумент метода Location определяет имя листа, на котором будет располагаться диаграмма.

В нашем случае диаграмма создана на отдельном листе. С помощью следующих процедур переместим ее на первый лист рабочей книги — туда, где располагаются данные для построения диаграммы, затем переместим обратно на отдельный лист (полный пример исходного текста приложения представлен на сопроводительном компакт-диске книги).

Перемещение диаграммы на лист с данными

```
const
 xlLocationAsNewSheet=1;
 xlLocationAsObject=2;
procedure TForm1.Button2Click(Sender: TObject);
begin
  Chart.Location(Where:=xlLocationAsObject, Name:='Лист1');
  Chart:=E.ActiveChart;
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
  Chart.Location(Where:=xlLocationAsNewSheet);
  Chart:=E.ActiveChart;
end;
```

Обратите внимание на то, что после вызова метода Location необходимо снова получать ссылку на объект ActiveChart, т. к. переменная Chart после переноса диаграммы на новый лист уже не содержит ссылку на диаграмму.

В имеющемся на сопроводительном компакт-диске книги примере приложения эти процедуры будут правильно работать, если создана только одна диаграмма. Если их будет больше, то вам придется изменить исходный текст приложения.

На рис. 14.2 представлена диаграмма, которая была перенесена с отдельного листа. Обратите внимание на область ячеек в верхнем левом углу листа — она определена как область исходных данных.

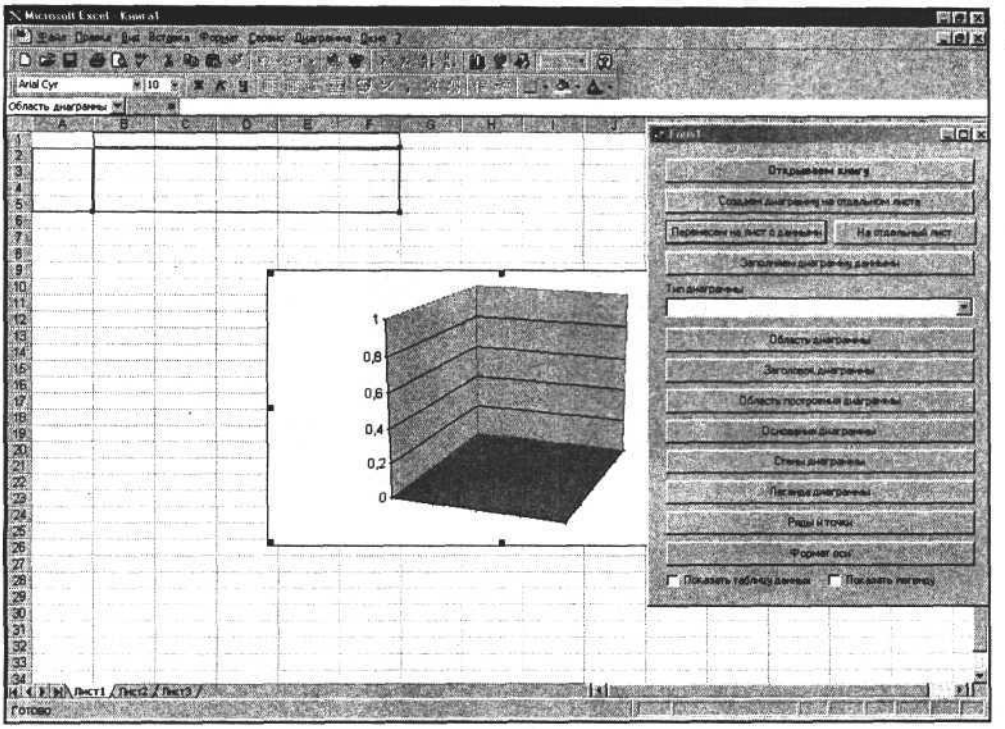


Рис. 14.2. Диаграмма, перемещенная на лист с данными

Заполним первую строку области исходных данных диаграммы (т. е. область подписей делений) определенными значениями, а ячейки, определяющие значения точек данных, — случайными числами. Для этого можно использовать следующие процедуры.

Заполнение области исходных данных для диаграммы

```
function SetValueRange(sheet:variant; range:string;
                       value_:variant):boolean;
begin
  SetValueRange:=True;
  try
    E.ActiveWorkbook.Sheets.Item[sheet].Range[range]:=value_;
    except
```

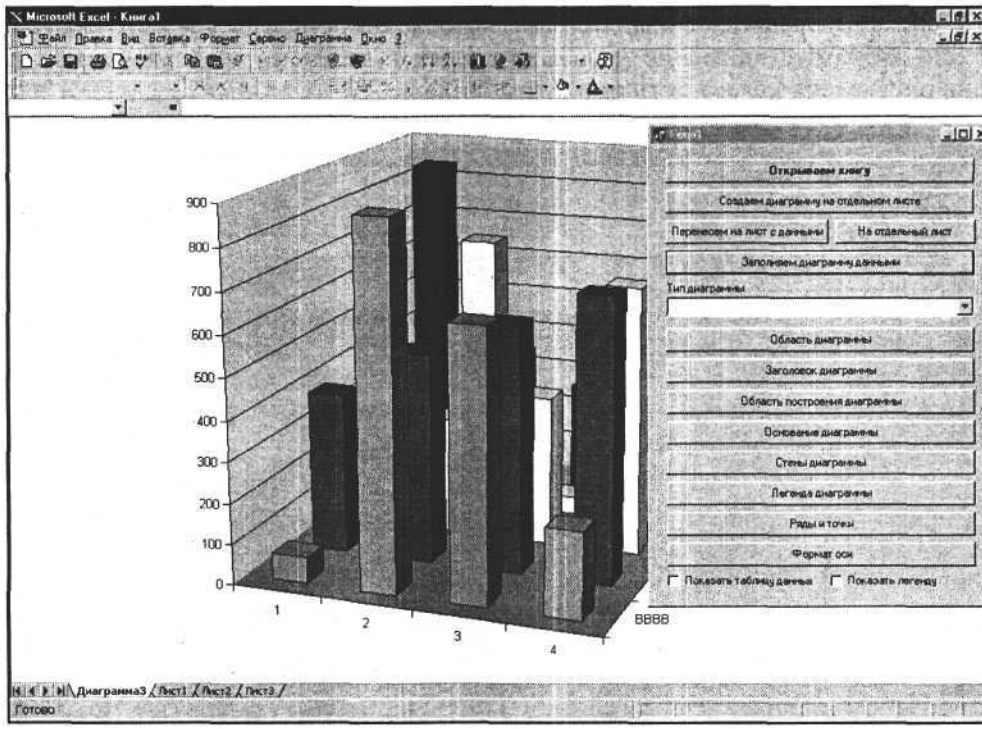


Рис. 14.3. Заполняем диаграмму исходными данными

```
    SetValueRange:=False;
  end;
end;

procedure TForm1.Button4Click(Sender: TObject);
 var a_:integer;
begin
  randomize;
  SetValueRange('Лист1','A1','AAAA');
  SetValueRange('Лист1','B1','BBBB');
  SetValueRange('Лист1','C1','CCCC');
  SetValueRange('Лист1','D1','AAAA');
  SetValueRange('Лист1','E1','BBBB');
  SetValueRange('Лист1','F1','CCCC');
  for a_:=2 to 5 do begin
    SetValueRange('Лист1','A'+inttostr(a_),a_-1);
    SetValueRange('Лист1','B'+inttostr(a_),random(1000));
    SetValueRange('Лист1','C'+inttostr(a_),random(1000));
    SetValueRange('Лист1','D'+inttostr(a_),random(1000));
```

```
    SetValueRange('Лист1','E'+inttostr(a_),random(1000));
    SetValueRange('Лист1','F'+inttostr(a_),random(1000));
  end;
end;
```

Результат выполнения данных процедур представлен на рис. 14.3.

Тип диаграммы

Обычно тип диаграммы задается сразу после ее создания, но в процессе работы приложения иногда требуется изменить внешний вид диаграммы. Для этого используем свойство ChartType объекта Chart, позволяющее изменять тип диаграммы путем записи в него значения константы из заданного списка возможных значений.

Следующая процедура демонстрирует способ применения свойства ChartType. На рис. 14.4 и 14.5 показаны два из возможных типов диаграмм.

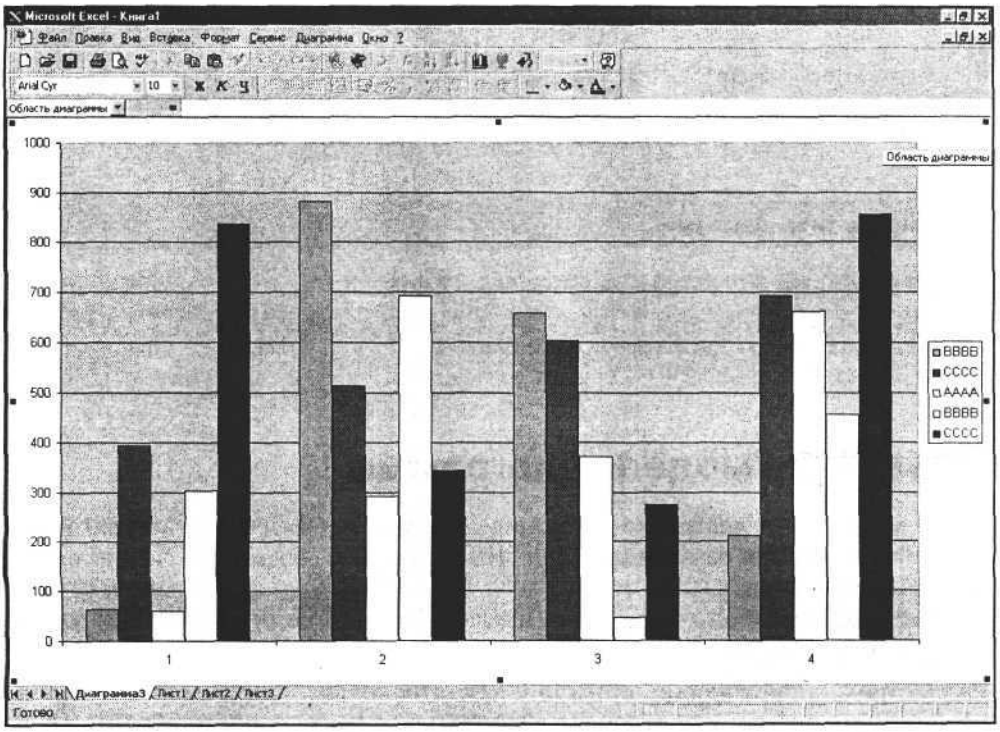


Рис. 14.4. Обычная гистограмма

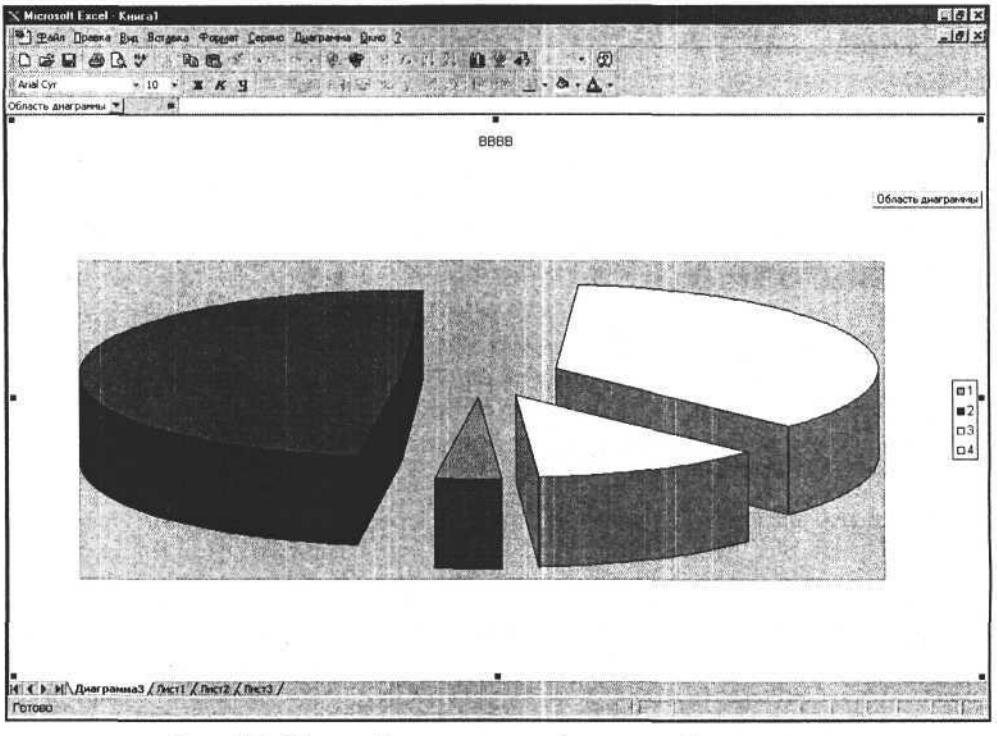


Рис. 14.5. Объемный вариант круговой разрезанной диаграммы

Изменение типа диаграммы

```
procedure TForm1.ChartTypeClick(Sender: TObject);
begin
  Chart.ChartType:=ChartType.ItemIndex+51;
end;
```

Объектная модель диаграммы

Итак, мы создали диаграмму на основе некоторого массива значений и попытались изменить ее внешний вид. Но диаграмма, возможно, является одним из самых сложных объектов Excel. Поэтому, чтобы далее рассматривать ее свойства, необходимо получить представление о ее структуре, соответствующей объектной модели диаграммы.

Рассмотрим составляющие объекта Chart. Поскольку этот объект состоит из графических элементов, очевидно, что бóльшая часть свойств объекта Chart предназначена для их описания. На рис. 14.6 представлен состав объемного варианта обычной гистограммы.

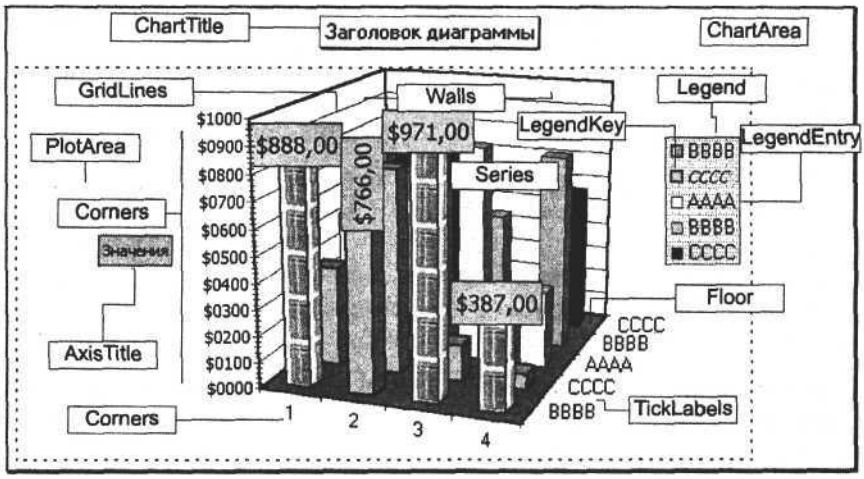


Рис. 14.6. Состав диаграммы

Диаграмма состоит из области диаграммы, на которой расположены заголовок диаграммы, область построения диаграммы, сама диаграмма, легенда, заголовки (названия) осей диаграммы, подписи данных и др. Диаграмма, показанная на рис. 14.6, содержит основание, стены, оси и ряды значений. Чтобы представить, как эти объекты взаимодействуют между собой, рассмотрим краткое описание объектной модели диаграммы.

ChartArea — область диаграммы; представляет собой объект, описывающий поверхность, на которой находятся все элементы диаграммы.

ChartTitle — заголовок диаграммы; его основные свойства связаны с текстом и прямоугольной областью, в которой расположен этот текст.

PlotArea — область построения диаграммы, которая представляет собой геометрическую фигуру с присущими ей свойствами.

Сама диаграмма ограничена тремя прямоугольниками: Floor — основание и Walls — две стены, которые также имеют свойства обычных геометрических фигур.

Corners — углы, представляют собой точки по краям диаграммы.

Axes (коллекция, Axis — элемент коллекции) — оси диаграммы; представляют собой линии, которые имеют заголовок оси (AxisTitle), могут иметь метки делений (TickLabels) и линии сетки (MajorGridLines — основные, MinorGridLines — промежуточные). Заголовок (или название) оси AxisTitle представляет собой объект, имеющий свойства обычной геометрической фигуры и свойства отображения текста. Линии сетки и осей имеют такие свойства, как цвет, тип и толщина.

Legend — легенда; представляет собой прямоугольную область, включающую в себя элементы легенды (коллекцию LegendEntries, элемент коллекции —

LegendEntry), которые в свою очередь содержат ключ легенды — LegendKey. Очевидно, что легенда, как и ее составные элементы, описывается свойствами, аналогичными свойствам надписи (объекта TextBox).

Рассмотрим объекты, описывающие содержание диаграммы.

SeriesCollection — ряды, включают в себя следующие объекты: коллекцию DataLabels (элемент коллекции — DataLabel) — подписи данных, LeaderLines — линии выноски, ErrorBars — линии (планки) погрешностей, коллекцию TrendLines (элемент коллекции — TrendLine) — линии тренда (сглаживания и аппроксимации) и коллекцию Points (элемент коллекции — Point) — точки.

Точка (объект Point) может содержать объект DataLabel — подпись данных.

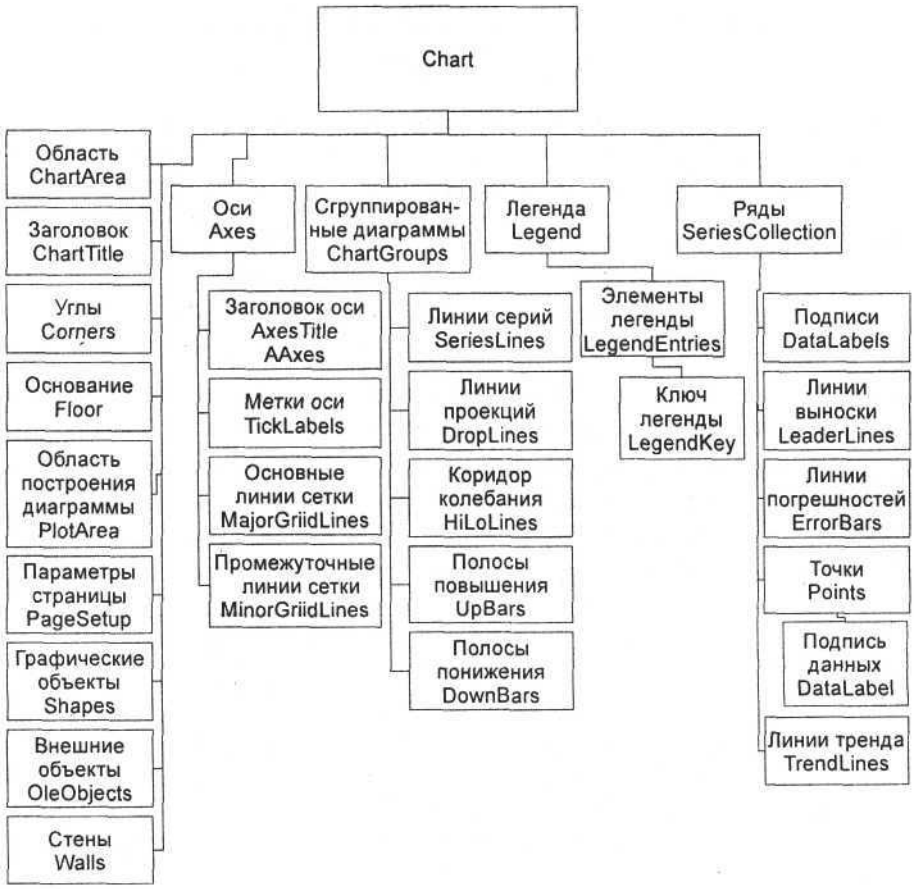


Рис. 14.7. Объектная модель диаграммы

В некоторых диаграммах, например в двухмерных, ряды данных можно группировать по некоторым признакам. Для этого используется коллекция ChartGroups (элемент коллекции — ChartGroup). Дополнительную информацию могут нести элементы этой коллекции, которые в свою очередь содержат следующие объекты: SeriesLines — линии серий (рядов), DropLines — линии проекций, HiLoLines — коридор колебания, UpBars — полосы повышения, DownBars — полосы понижения.

Диаграмма обладает еще и свойствами, знакомыми вам по предыдущим главам: Shapes — коллекция графических объектов, которые можно разместить непосредственно на области диаграммы; OLEObjects — коллекция внешних объектов (OLE-объектов), также размещаемых на области диаграммы; PageSetup — параметры страницы, это свойство предназначено для подготовки вывода диаграммы на печать.

Данное описание диаграммы лучше представить в графическом виде. На рис. 14.7 представлена объектная модель объекта Chart.

Данное представление является общим и статичным. Для более детального изучения объектной модели диаграммы нужно ознакомиться с тем, как взаимодействуют ее элементы в динамике, как используются свойства и методы элементов диаграммы, и установить взаимосвязь между изменениями свойств объектов и визуальным отображением происходящих изменений. Для этого продолжим изучение модели на основе конкретных программных примеров. Поскольку мы рассматриваем применение среды визуального программирования Delphi, все исходные тексты приведены на языке Pascal. Полные исходные тексты приложений представлены на сопроводительном компакт-диске книги.

Область диаграммы

Визуальными параметрами области диаграммы являются заливка, линия границы, шрифт, наличие тени. Чтобы манипулировать этими свойствами, достаточно получить к ним доступ. Поскольку они принадлежат области диаграммы, а значит, объекту ChartArea, изменение заливки обеспечивается изменением свойства Fill или Interior объекта, изменение толщины, цвета и типа линии границы — изменением свойства Border объекта, а изменение шрифта для текста диаграммы — настройкой свойств объекта Font. Свойство Shadow:Boolean определяет наличие тени. Рассмотрим примеры процедур.

Настройка свойств заливки области диаграммы

```
 var Variant:integer; Degree:real;
procedure TOKBottomDlg3.ForeColor1Change(Sender: TObject);
```

```
begin
  ChartArea.Fill.ForeColor.SchemeColor:=ForeColor1.Value;
end;

procedure TOKBottomDlg3.GradientStyleChange(Sender: TObject);
begin
  ChartArea.Fill.OneColorGradient(msoGradientHorizontal,Variant,Degree);
end;
```

На рис. 14.8 приведен результат выполнения этих процедур.

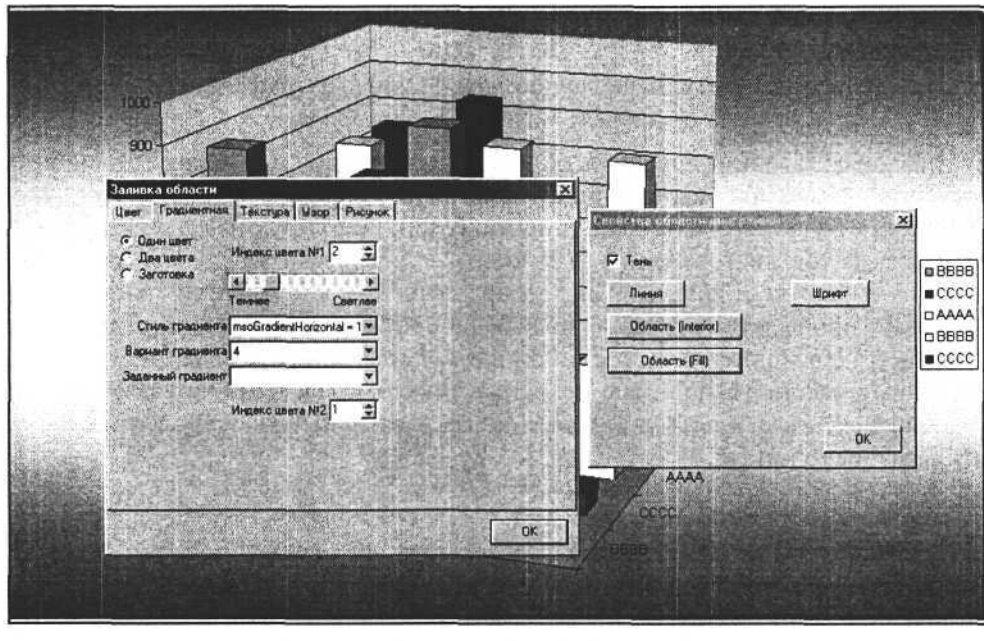


Рис. 14.8. Изменение свойств заливки области диаграммы

Заголовок диаграммы

Чтение и изменение свойств заголовка диаграммы производятся с помощью объекта ChartTitle, но для доступа к этому объекту необходимо активизировать заголовок, иначе объект ChartTitle и сам заголовок будут недоступны. Это делается путем установки свойства Chart.HasTitle в значение True (для скрытия заголовка этому свойству присваивается значение False). Получив доступ к заголовку, мы можем изменить и его содержание и параметры области, в которой он размещен. Для этого используем свойства Font

(шрифт), Border (линия границы), Interior и Fill (свойства заливки), Shadow (наличие тени), Left и Top (координаты расположения заголовка) объекта ChartTitle.

Изменение свойств заголовка диаграммы

```
procedure TOKBottomDlg2.HasTitleClick(Sender: TObject);
begin
  Chart.HasTitle:=True;
  Chart.ChartTitle.Text:='Заголовок диаграммы';
end;

procedure TOKBottomDlg2.ShadowClick(Sender: TObject);
begin
  Chart.ChartTitle.Shadow:=Shadow.Checked;
end;
```

На рис. 14.9 представлен заголовок диаграммы, измененный при выполнении приведенных процедур.

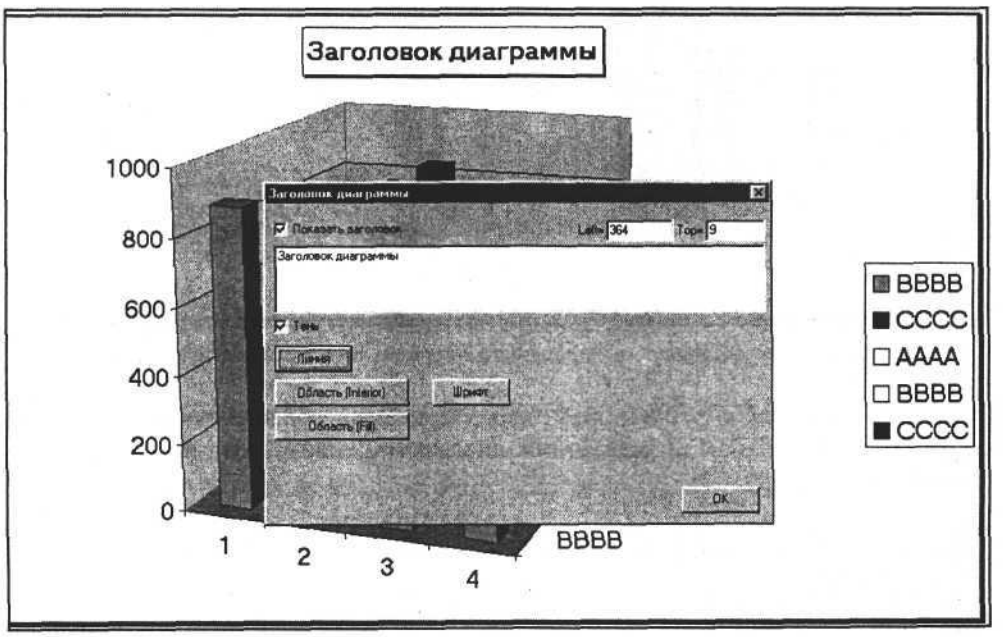


Рис. 14.9. Настройка заголовка диаграммы

Область построения диаграммы, основание и стены диаграммы

Область построения диаграммы, основание и стены диаграммы представляют собой поверхности, ограниченные линией. Следовательно, свойствами этих объектов являются параметры линии и заливки. В дополнение у области построения диаграммы имеются свойства, позволяющие задать ее координаты и размеры. Свойство Border определяет толщину, цвет и тип линии границы, свойства Interior и Fill — цвет и способ заливки внутренней области.

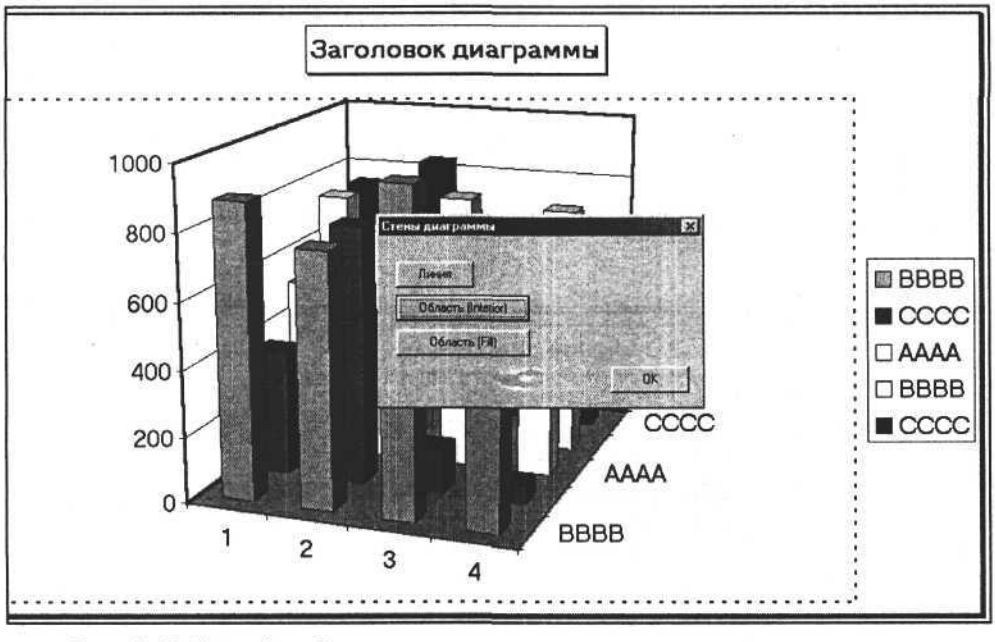


Рис. 14.10. Настройка области построения диаграммы, стен и основания диаграммы

Изменим с помощью этих свойств характеристики области построения диаграммы, а также стен и основания диаграммы, например, выполнив следующую процедуру.

Изменение свойств области построения диаграммы, основания и стен диаграммы

```
const XlDot=-4118;
begin
  Chart.PlotArea.Border.Weight:=1;
  Chart.PlotArea.Border.LineStyle:=xlDot;
```

```
  Chart.Floor.Border.Color:=RGB(0,200,0);
  Chart.Floor.Border.Weight:=4;
  Chart.Walls.Border.Color:=RGB(0,0,200);
  Chart.Walls.Border.Weight:=4;
  Chart.Walls.Interior.Color:=RGB(255,255,255);
end;
```

Результат выполнения этой процедуры может быть примерно таким, как показано на рис. 14.10.

Легенда

Доступ к свойствам легенды диаграммы осуществляется посредством объекта Legend, являющегося свойством диаграммы Chart. Для открытия доступа к объекту Legend необходимо присвоить свойству HasLegend объекта Chart значение True, после чего легенда диаграммы отобразится в области диаграммы и будет доступна для изменений.

Легенда представляет собой прямоугольную область, для которой можно изменить визуальные свойства заливки и линии границы. Оператор

```
Legend:=Chart.Legend;
```

возвращает в переменную Legend:variant ссылку на объект "легенда". После этого можно непосредственно обращаться к этой легенде. Доступ к линии границы легенды обеспечивает свойство Border, к заливке — свойство Interior или Fill. Используя эти свойства, изменим внешний вид легенды.

Настройка заливки и линии границы легенды

```
const msoPatternHorizontalBrick=35;
var Legend:variant;
procedure TForm.WeightChange(Sender: TObject);
begin
  Legend.Border.Weight:=Weight.Value;
end;

procedure TForm.PatternChange(Sender: TObject);
begin
  Legend.Fill.Patterned(Pattern:=msoPatternHorizontalBrick);
end;

procedure TForm.ForeColor3Change(Sender: TObject);
begin
  Legend.Fill.ForeColor.SchemeColor:=ForeColor3.Value;
end;
```

```
procedure TForm.BackColor3Change(Sender: TObject);
begin
  Fill.BackColor.SchemeColor:=BackColor3.Value;
end;
```

Результат использования представленных процедур показан на рис. 14.11.

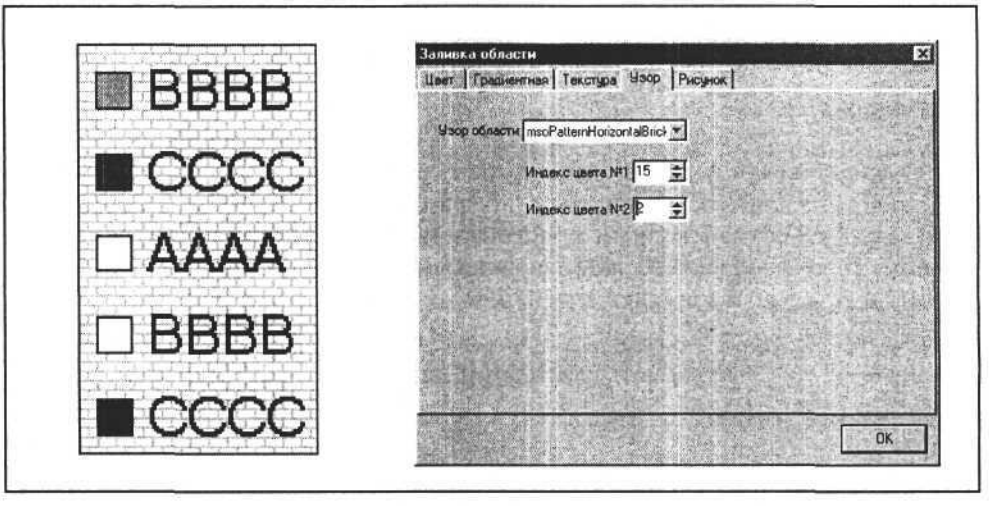


Рис. 14.11. Изменены заливка и линия границы легенды

Легенда включает в себя дополнительные объекты, объединенные в коллекцию LegendEntries. Это элементы легенды, их количество совпадает с количеством рядов диаграммы. У элемента легенды есть свойство Font для настройки шрифта текста и единственный дополнительный объект — ключ легенды. Ключ имеет свойства линии границы и заливки.

Доступ к элементам легенды осуществляется посредством коллекции LegendEntries с помощью индекса элемента (целого числа из диапазона от 1 до LegendEntries.Count, где Count — количество элементов легенды).

Настроим свойства легенды, выполнив следующие процедуры.

Изменение свойств элемента легенды

```
procedure TForm1.FormShow(Sender: TObject);
 var a_:integer;
begin
// Загружаем список элементов легенды в ListBox
  ListBox1.Items.Clear;
  for a_:=1 to Legend.LegendEntries.Count do begin
```

```
    ListBox1.Items.Add(inttostr(a_));
  end;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
// Получаем ссылку на элемент легенды
  I:=2;
  LegendEntry:=Legend.LegendEntries.item[i];
// Получаем ссылку на ключ легенды
  LegendKey:= LegendEntry.LegendKey;
// Изменяем толщину линии.
  LegendKey.Border.Weight:=4;
// Изменяем цвет заливки
  LegendKey.Interior.Color:=RGB(200,50,0);
// Получаем доступ к шрифту надписи и изменяем его.
  LegendEntry.Font.Size:=16;
end;
```

Результат выполнения этих процедур представлен на рис. 14.12 и 14.13.

Примечание

При программировании объектов, изменяющих визуальные свойства ключа легенды, необходимо учитывать тот факт, что это приводит к соответствующим изменениям свойств рядов диаграммы.

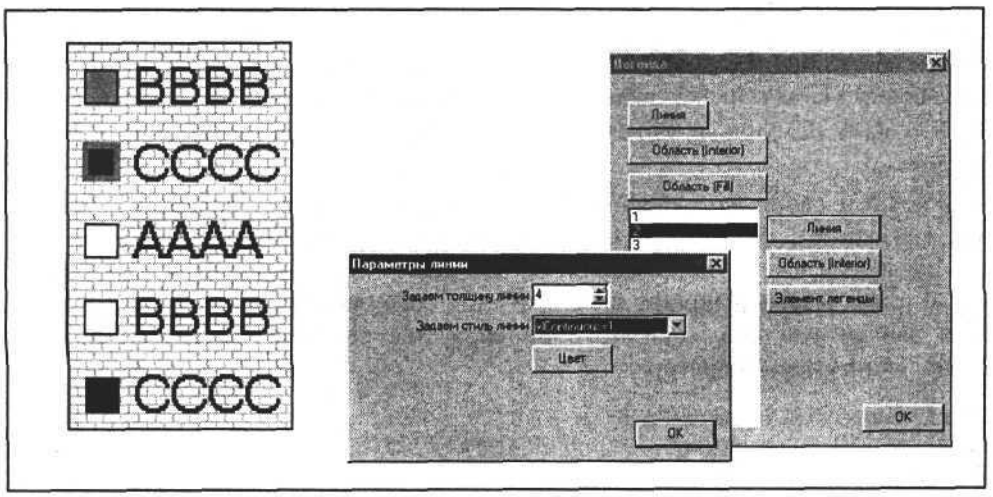


Рис. 14.12. Изменена линия и заливка ключа легенды

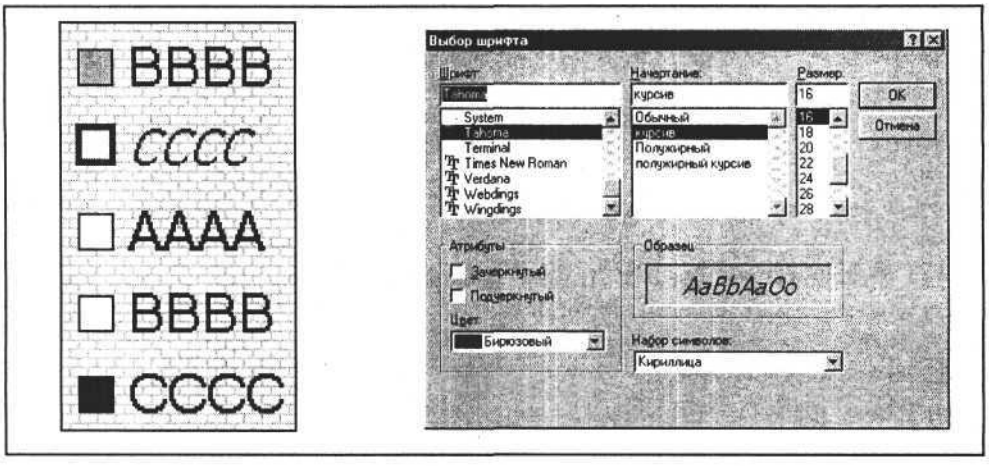


Рис. 14.13. Изменен шрифт текста элемента легенды

Оси

Оси диаграммы представляют собой линии с метками. Двухмерные диаграммы имеют ось значений и ось категорий, а трехмерные — еще и ось ряда данных. Оси предназначены для более наглядного отражения содержимого диаграммы. Вся информация об осях содержится в коллекции Axes объекта Chart. Понятно, что эта коллекция может содержать не более трех элементов-осей. Элемент коллекции Axes представляет собой объект Axis, объединяющий в себе все свойства выбранной оси. Для выбора объекта Axis из коллекции осей используется метод Item(i). Аргумент i метода представляет собой числовую константу, выбранную из трех возможных (xlCategory=1, xlValue=2, xlSeriesAxis=3). Соответственно мы можем получить ссылку на один из объектов: ось категорий, ось значений или ось ряда данных.

Рассмотрим свойства выбранной оси. Каждая ось может содержать заголовок (название), метки оси, основные и промежуточные линии сетки. Все эти элементы имеют такие свойства, как цвет, тип и толщина линий, заливка, ширина меток и шрифт текста. Кроме того, можно удалить или отобразить выбранную ось. Для этого используется свойство HasAxis(i) объекта Chart. Например, оператор

```
Chart.HasAxis(xlValue):=True;
```

отобразит, а оператор

```
Chart.HasAxis(xlValue):=False;
```

скроет ось значений. Сама ось значений имеет такие же свойства, как обычная линия, их можно настраивать; доступ к ее свойствам осуществляется посредством объекта Axis.Border.

Доступ к заголовку оси осуществляется посредством объекта Axis.AxisTitle. Заголовок оси состоит из линии границы, заливки и текста. Поэтому у объекта AxisTitle есть свойства Border, Interior, Fill, Caption и Font, связанные с соответствующими визуальными элементами заголовка.

Метки оси представляют собой текст, отображающий соответствующее значение. Доступ к свойствам меток осуществляется посредством объекта TickLabels, принадлежащего объекту Axis. Свойства NumberFormat, Orientation, Font объекта TickLabels позволяют задавать формат, направление и шрифт текста меток.

Ось имеет основные и промежуточные линии сетки, доступ к которым осуществляется посредством свойств MajorGridlines и MinorGridlines объекта Axis. Эти свойства сами представляют собой объекты, обладающие свойством линии (Border). Показать/скрыть линии сетки позволяют свойства HasMajorGridlines и HasMinorGridlines объекта Axis.

Линия оси также может содержать линии основных и промежуточных меток. Тип этих линий определяется путем записи числовых констант в свойства MajorTickMark и MinorTickMark объекта Axis.

Выберем ось значений и настроим ее свойства с помощью следующих процедур.

Задание свойств оси значений

```
procedure TForm1. BorderAxesClick(Sender: TObject);
 var Axis, Border:variant;
begin
  Axis:=Chart.Axes.Item(xlValue); // Получаем доступ к оси значений
  Border:=Axis.Border;          // Получаем доступ к линии оси значений
  Border.Color:=RGB(200,0,0); // и настраиваем ее свойства
  Border.Weight:=4;
end;

// Показываем заголовок оси значений, получаем доступ
// к нему и настраиваем его свойства
procedure TForm1.HasTitleClick(Sender: TObject);
 var AxisTitle:variant;
begin
  Axis.HasTitle:=True;
  AxisTitle:=Axis.AxisTitle;
```

```
  AxisTitle.Border.Color:=RGB(0,200,200);
  AxisTitle.Interior.Color:=RGB(200,200,0);
  AxisTitle.Caption:='Значения';
  AxisTitle.Font.Size:=24;
end;

// Получаем доступ к меткам делений оси и настраиваем их свойства
// (шрифт, формат числовых меток и направление текста (угол))
procedure TForm1.Button1Click(Sender: TObject);
 var TickLabels:variant;
begin
  TickLabels:=Axis.TickLabels;
  TickLabels.Font.Size:=20;
  TickLabels.NumberFormat:='$0000';
  TickLabels.Orientation:=0;
end;

// Получаем доступ к меткам делений оси и настраиваем их свойства
// (толщину и цвет основных линий меток)
procedure Tform1.HasMajorGridlinesClick(Sender: TObject);
 var MajorGridlines:variant;
begin
  Axis.HasMajorGridlines:=True;
  Axis.HasMinorGridlines:=False;
  MajorGridlines:=Axis.MajorGridlines;
  MajorGridlines.Border.Weight:=2;
  MajorGridlines.Border.Color:=GRB(0,0,0);
end;

// Показываем линии основных и промежуточных меток.
procedure TForm1.Button2(Sender: TObject);
const
  xlTickMarkCross=4;
  xlTickMarkInside=2;
begin
  Axis.MajorTickMark:=xlTickMarkCross;
  Axis.MinorTickMark:=xlTickMarkInside;
end;
```

Результат всех этих преобразований может иметь вид, показанный на рис. 14.14.

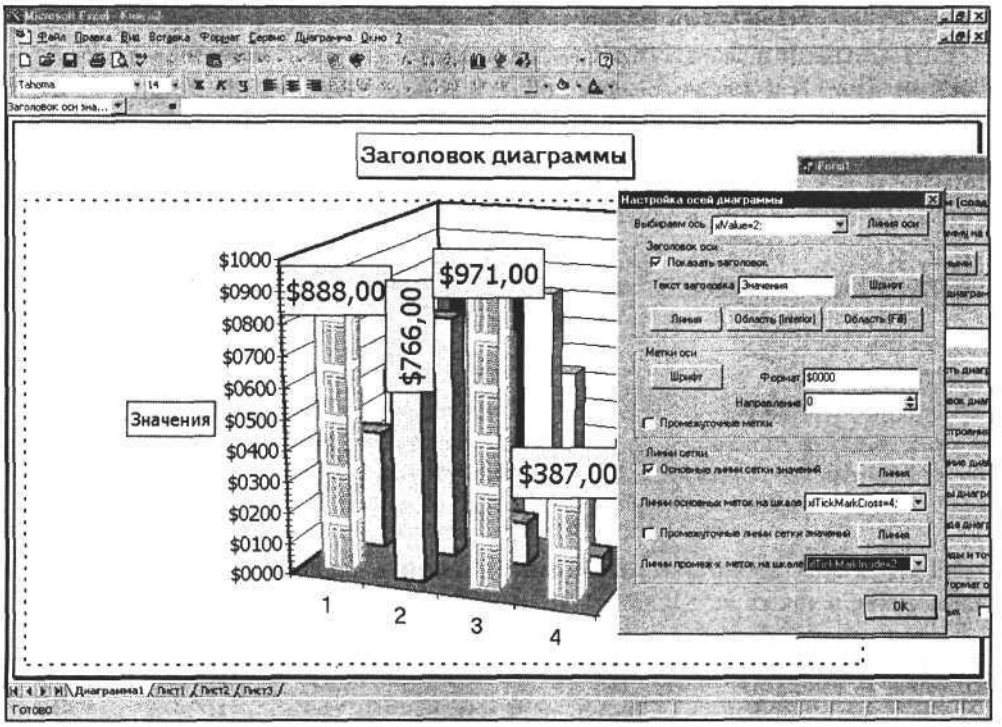


Рис. 14.14. Настройка оси значений диаграммы

Ряды и точки

Мы подошли к самому главному — отображению значений диаграммы. Известно, что диаграмма может содержать не одну последовательность значений. Каждая такая последовательность представляет собой ряд, состоящий из точек.

Точка отражает одно значение из последовательности значений определенного ряда. Точки одного ряда могут быть соединены между собой линиями или представлять собой отдельные плоские или объемные фигуры, это зависит от выбора типа диаграммы. В любом случае точки имеют между собой логическую связь и образуют *ряды* значений. Таким образом, при программировании значений диаграммы мы имеем дело как минимум с двумя типами объектов: это объекты, объединенные в коллекцию рядов SeriesCollection, и объекты, объединенные в коллекцию точек Points. Коллекция точек принадлежит ряду.

Как и любая коллекция, коллекция SeriesCollection обладает свойствами, позволяющими определить количество элементов этой коллекции, и мето-

дами, обеспечивающими доступ к любому объекту этой коллекции. Метод Item(i) коллекции SeriesCollection возвращает доступ к элементу Series коллекции. Получив доступ к элементу коллекции, т. е. к ряду диаграммы, мы имеем возможность настроить его внешний вид. Для этого, как и для аналогичных объемных или плоских элементов, используем объекты Border, Interior и Fill, связанные с параметрами границ и внутренней области объектов, отображающих точки ряда диаграммы.

Доступ к ряду и его свойствам

```
var i:integer;
i:=1;
Series:=SeriesCollection(i);   // Ссылка на первый ряд диаграммы
Border:=Series.Border;         // Ссылка на линию границы
Interior:=Series.Interior;     // Ссылка на внутреннюю область
Fill:=Series.Fill;             // Ссылка на заливку
```

Используя эти свойства, мы можем изменить параметры линии границы элементов, а также их заливку (рис. 14.15).

Рис. 14.15. Настройка элементов диаграммы

Подписи данных для точек ряда диаграммы также представляют собой геометрические фигуры, содержащие текст. Доступ к ним обеспечивается посредством объекта DataLabels, который принадлежит объекту Series из коллекции SeriesCollection. Их внешний вид определяется свойствами линии и заливки, а также свойствами текста. Свойства текста определяются его типом, шрифтом, способом выравнивания в рамках прямоугольной области объекта DataLabels, форматом отображения значений и углом поворота (направлением).

Настройка свойств подписей данных для точек ряда диаграммы

```
DataLabels:variant;
// Выбор типа подписи
procedure TOKBottomDlg8.LabelsTypeChange(Sender: TObject);
begin
  case LabelsType.ItemIndex of
  0: XlDataLabelsType:=xlDataLabelsShowNone;
  1: XlDataLabelsType:=xlDataLabelsShowValue;
  2: XlDataLabelsType:=xlDataLabelsShowPercent;
  3: XlDataLabelsType:=xlDataLabelsShowLabel;
  4: XlDataLabelsType:=xlDataLabelsShowLabelAndPercent;
  end;
  Series.ApplyDataLabels(Type:=XlDataLabelsType);
end;

// Выбор числового формата для отображения значений
procedure TOKBottomDlg8.sNumberFormatChange(Sender: TObject);
begin
  DataLabels:=Series.DataLabels;
  DataLabels.NumberFormat:=sNumberFormat.Text;
end;

// Горизонтальное выравнивание текста
procedure TOKBottomDlg8.sHorizontalAlignmentChange(Sender: TObject);
begin
  case sHorizontalAlignment.ItemIndex of
  0: DataLabels.HorizontalAlignment:=xlHAlignCenter;
  1: DataLabels.HorizontalAlignment:=xlHAlignLeft;
  2: DataLabels.HorizontalAlignment:=xlHAlignRight;
  end;
end;

// Вертикальное выравнивание текста
procedure TOKBottomDlg8.sVerticalAlignmentChange(Sender: TObject);
```

```
begin
  case sHorizontalAlignment.ItemIndex of
  0: DataLabels.VerticalAlignment:=xlVAlignBottom;
  1: DataLabels.VerticalAlignment:=xlVAlignCenter;
  2: DataLabels.VerticalAlignment:=xlVAlignTop;
  end;
end;

// Угол (направление) текста
procedure TOKBottomDlg8.sOrientationChange(Sender: TObject);
begin
  DataLabels.Orientation:=sOrientation.Value;
end;
```

Настраивая эти свойства, можно существенно изменить внешний вид диаграммы, а возможность сочетать варианты выбора позволяет представить информацию пользователю наиболее наглядно (рис. 14.16).

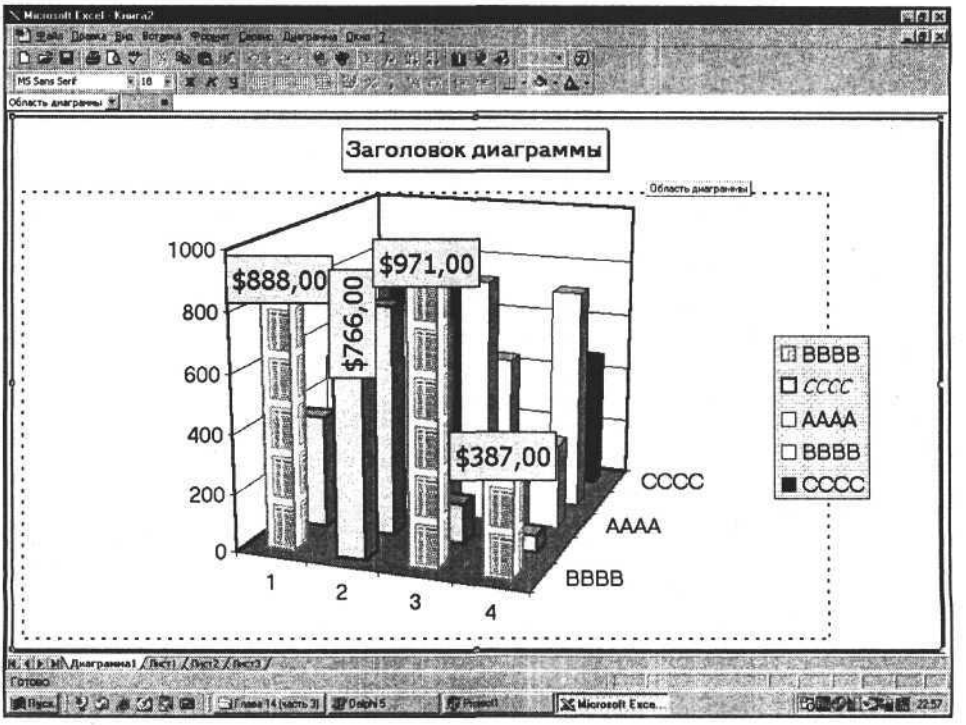


Рис. 14.16. Настройка подписей ряда диаграммы

Интересные возможности предоставляет применение формата, задаваемого строкой символов и позволяющего задать необходимый вид для числового

значения. Для задания формата чисел можно использовать символы 0, #, ?, $ и т. д. (их список ограничивается только вашей фантазией и требованиями пользователей). Формат чисел, показанных на рис. 14.16, задан следующим оператором:

```
DataLabels.NumberFormat:='$000,00';
```

В рамках выбранного типа диаграммы объект Series дает возможность изменять фигуру для точек ряда. Это осуществляется записью определенной константы в свойство BarShape объекта Series. Следующая процедура позволяет это сделать.

Изменение вида фигуры для точек ряда диаграммы

```
procedure TOKBottomDlg8.sBarShapeChange(Sender: TObject);
 const xlPyramidToPoint=1;
begin
  Series.BarShape:= xlPyramidToPoint;
end;
```

В данном случае мы устанавливаем отображение точки ряда в виде пирамиды (рис. 14.17).

Если внимательно рассмотреть диаграммы на рис. 14.16 и 14.17, то мы обнаружим отклонение стиля второй точки от остальных точек этого ряда. Это достигается путем изменения свойств выбранной точки данного ряда. Точки ряда диаграммы объединены в коллекцию Points, свойства и методы которой обеспечивают доступ к любой точке ряда. Свойство Count этой коллекции содержит количество элементов коллекции, а метод Item(i:integer) возвращает ссылку на элемент коллекции. Чтобы не повторяться, достаточно отметить, что свойства объекта Point в большей степени идентичны свойствам объекта Series. Программирование свойств этого объекта более детально представлено в исходном тексте приложения на сопроводительном компакт-диске книги.

Рассмотрим еще одно свойство объекта Series, являющееся альтернативой задания области исходных данных для построения диаграммы. Это свойство имеет строковый тип и содержит формулу, по которой строится ряд диаграммы. Свойство Formula принадлежит непосредственно объекту Series.

Следующая процедура иллюстрирует его использование для получения выражения формулы (рис. 14.18).

Получение строки, содержащей формулу для заданного ряда

```
procedure TOKBottomDlg8.Button16Click(Sender: TObject);
var eee_:string;
```

Рис. 14.17. Изменение вида фигуры для выбранного ряда

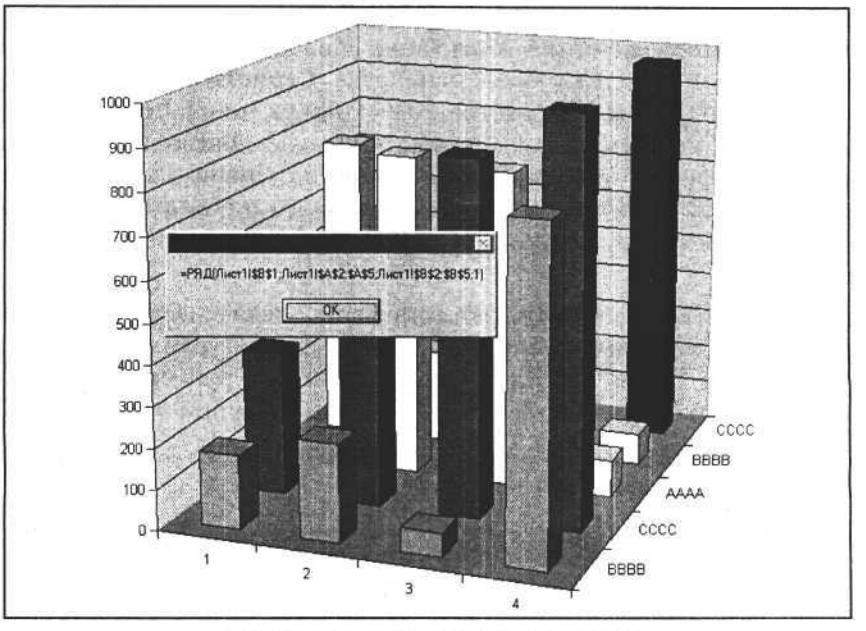


Рис. 14.18. Формула для построения ряда

```
begin
  eee_:=Series.Formula;
  MessageBox(handle,pchar(eee_),'',0);
end;
```

Используя формулы для объекта Series, можно изменить область исходных данных для построения ряда.

Следующий оператор изменяет границы области исходных данных построения для выбранного ряда диаграммы:

```
Series.FormulaLocal = '=SERIES(,,Лист1!R5C1:R5C7,2)';
```

Объемные диаграммы

Объемный вид диаграммы определяется горизонтальным и вертикальным углами поворота, изометрией и другими параметрами, влияющими на вид трехмерной проекции. Объект Chart обладает следующими свойствами, позволяющими изменить диаграмму объемного вида: Rotation — поворот (в горизонтальной плоскости), Elevation — возвышение (поворот в вертикальной плоскости), Perspective — перспектива, RightAngleAxes — изометрия, AutoScaling — автомасштаб, HeightPercent — высота в процентах от нормальной высоты. Все перечисленные свойства имеют числовой тип, за исключением свойства AutoScaling, имеющего логический тип. Значения этих свойств можно считывать и изменять.

Следующие операторы позволяют определить значения параметров объемной диаграммы.

Задание свойств объемной диаграммы

```
Rotation.Value:=Chart.Rotation;
Elevation.Value:=Chart.Elevation;
Perspective.Value:=Chart.Perspective;
RightAngleAxes.Checked:=Chart.RightAngleAxes;
AutoScaling.Checked:=Chart.AutoScaling;
HeightPercent.Value:=Chart.HeightPercent;
```

На рис. 14.19 представлены диаграмма и форма приложения, в которой отображены считанные значения свойств, определяющих объемный вид диаграммы.

Используя эту форму, изменим некоторые параметры, например возвышение и поворот диаграммы.

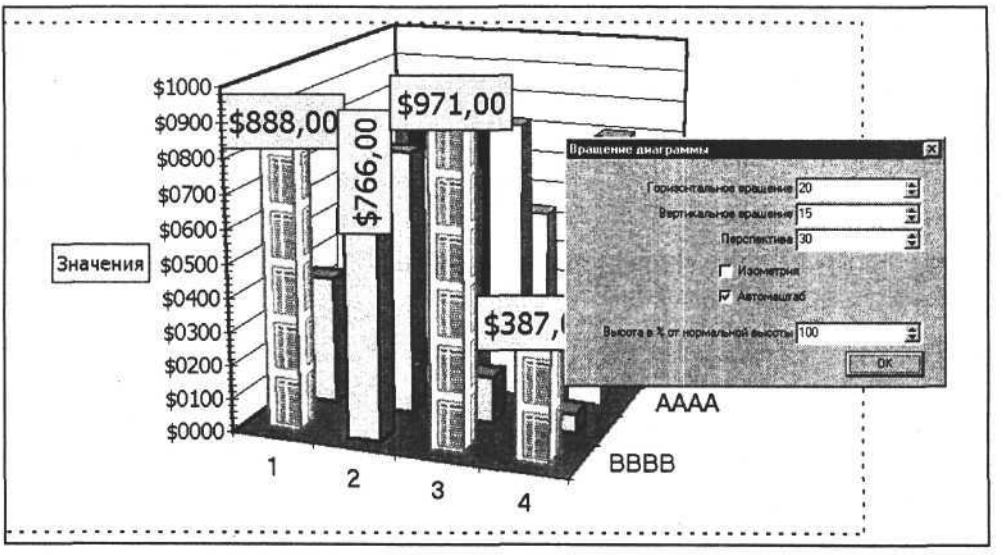


Рис. 14.19. Значения свойств объемной диаграммы

Изменение возвышения и поворота диаграммы

```
procedure TOKBottomDlg10.ElevationChange(Sender: TObject);
begin
  Chart.Elevation:=Elevation.value;
end;

procedure TOKBottomDlg10.RotationChange(Sender: TObject);
begin
  Chart.Rotation:=Rotation.Value;
end;
```

На рис. 14.20 видны результаты этих изменений.

Используя свойство RightAngleAxes объекта Chart, изменяем изометрию объемной диаграммы.

Задание изометрии диаграммы

```
procedure TOKBottomDlg10.RightAngleAxesClick(Sender: TObject);
begin
  Chart.RightAngleAxes:=RightAngleAxes.Checked;
end;
```

Внешний вид измененной диаграммы представлен на рис. 14.21.

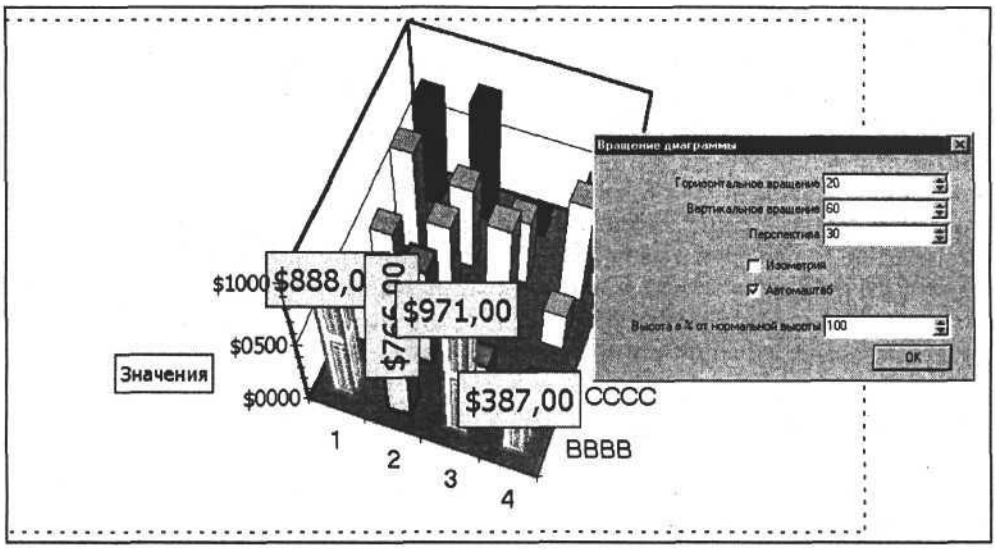


Рис. 14.20. Вращение диаграммы в двух плоскостях

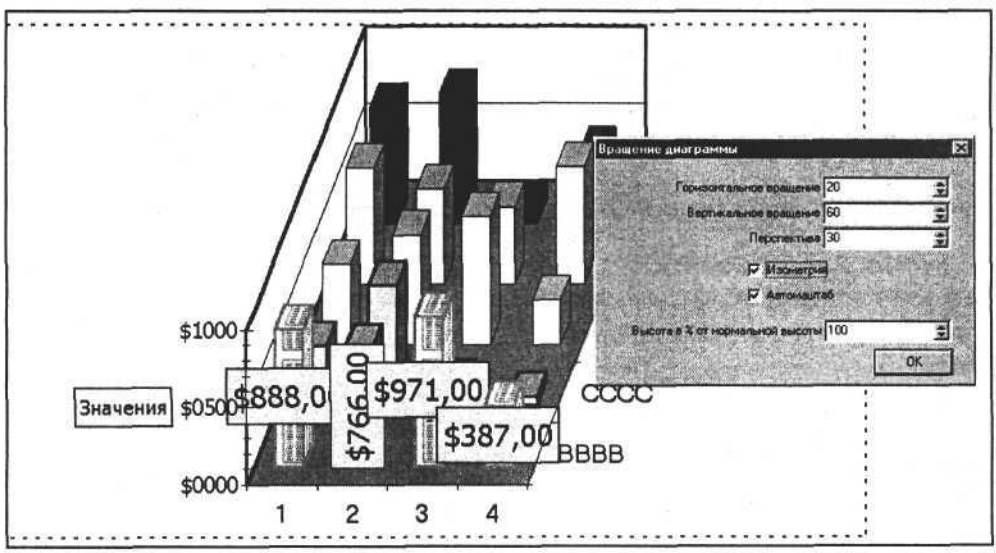


Рис. 14.21. Изометрический вид диаграммы

Особенности некоторых типов диаграмм

Очень часто диаграммы предназначены для изучения сравнительных характеристик каких-либо процессов, которые они отражают. Обычно в этих случаях диаграмма должна содержать не менее двух рядов. Для сравнения ря-

дов на одной диаграмме можно использовать дополнительные графические элементы, причем для разных типов диаграмм можно использовать присущие только им специфические элементы диаграмм. Очевидно, что создавать такие элементы и управлять ими можно, когда между основными рядами диаграммы есть логическая связь.

Коллекция ChartGroups объекта Chart позволяет создавать дополнительные элементы, отражающие взаимосвязь и сравнительные характеристики рядов диаграммы, и управлять ими. Коллекция ChartGroups имеет свойство Count (количество ее элементов) и метод Item(i:integer) (ссылка на элемент коллекции). У нее нет методов, позволяющих создавать новые элементы — значение свойства Count определяется типом и составом диаграммы.

Линии серий (рядов)

Для плоских гистограмм с накоплением есть возможность отобразить линии серий (рядов). Они представляют собой прямые, соединяющие границы областей значений соседних рядов. Если изменить тип диаграммы, приведя ее к объемному виду, то линии серий, отображавшиеся на плоской диаграмме, исчезнут. Для включения режима отображения линий серий используем свойство HasSeriesLines объекта ChartGroup, установив его в значение True. Цвет и другие свойства линий серий определяются свойством Border объекта SeriesLines. Объект SeriesLines в свою очередь принадлежит объекту ChartGroup. Рассмотрим следующую процедуру.

Отображение линий серий на диаграмме

```
const xlColumnStacked=52;
procedure TOKBottomDlg11.HasSeriesLinesClick(Sender: TObject);
 var ChartGroup:variant;
begin
  Chart.ChartType:=xlColumnStacked;
  ChartGroup:=Chart.ChartGroups.Item(1);
  ChartGroup.HasSeriesLines:=HasSeriesLines.Checked;
  if HasSeriesLines.Checked then begin
    ChartGroup.SeriesLines.Border.ColorIndex:=32;
    ChartGroup.SeriesLines.Border.Weight:=4;
  end;
end;
```

Первый оператор представленной процедуры устанавливает нужный тип диаграммы, второй возвращает ссылку на элемент коллекции ChartGroups. Далее устанавливаем режим отображения линий серий и настраиваем их параметры. Результат показан на рис. 14.22.

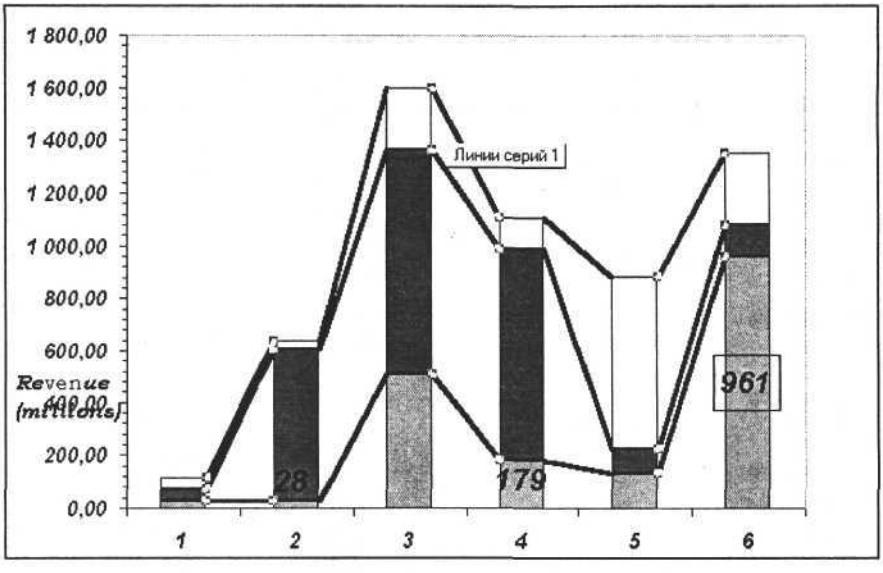


Рис. 14.22. Линии серий на диаграмме

Линии проекции

Линиями проекции называются прямые линии, соединяющие точки данных с осью категорий. Для отображения линий проекции используются элементы коллекции ChartGroups объекта Chart. Чтобы отобразить проекцию, нужно присвоить значение True свойству HasDropLines объекта ChartGroup, что дает возможность доступа к объекту DropLines элемента коллекции.

Примечание

Не все типы диаграмм поддерживают линии проекции, поэтому предварительно необходимо задать допустимый тип диаграммы (линейчатые диаграммы, диаграммы с накоплением).

Рассмотрим создание линий проекции на следующем примере.

Отображение линий проекции на диаграмме

```
const xlLine=4;
procedure TOKBottomDlg11.HasDropLinesClick(Sender: TObject);
 var ChartGroup:variant;
begin
  Chart.ChartType:=xlLine;
  ChartGroup:=Chart.ChartGroups.Item(1);
  ChartGroup.HasDropLines:=HasDropLines.Checked;
```

```
  if HasDropLines.Checked then ChartGroup.DropLines.Border.ColorIndex:=3;
end;
```

На рис. 14.23 показаны линии проекции для диаграммы в виде графиков.

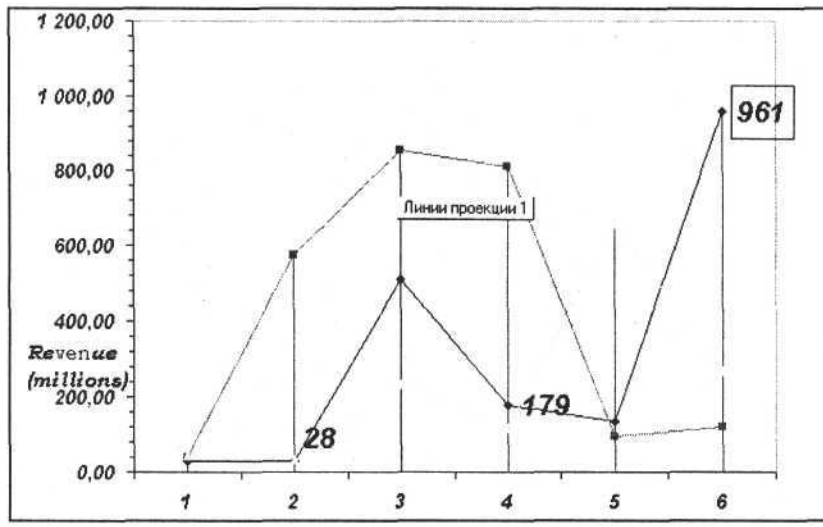


Рис. 14.23. Линии проекций

Коридор колебания (изменения)

Коридор колебания (изменения) на плоских диаграммах отображается линиями, идущими от максимального до минимального значения в каждой категории.

Для отображения коридора колебания нужно установить свойство HasHiLoLines объекта ChartGroup в значение True. После этого настройка свойств линий коридора будет доступна через свойства объекта HiLoLines, принадлежащего элементу коллекции ChartGroup.

Рассмотрим построение линий коридора колебания на примере графиков. Следующая процедура позволяет отобразить и настроить цвет линий коридора колебания.

Отображение настройка цвета линий коридора колебания

```
procedure TOKBottomDlg11.HasHiLoLinesClick(Sender: TObject);
 var ChartGroup:variant;
begin
  Chart.ChartType:=xlLine;
  ChartGroup:=Chart.ChartGroups.Item(1);
```

```
  ChartGroup.HasHiLoLines:=HasHiLoLines.Checked;
  if HasHiLoLines.Checked then
    ChartGroup.HiLoLines.Border.ColorIndex:=5;
end;
```

Результат выполнения этой процедуры представлен на рис. 14.24.

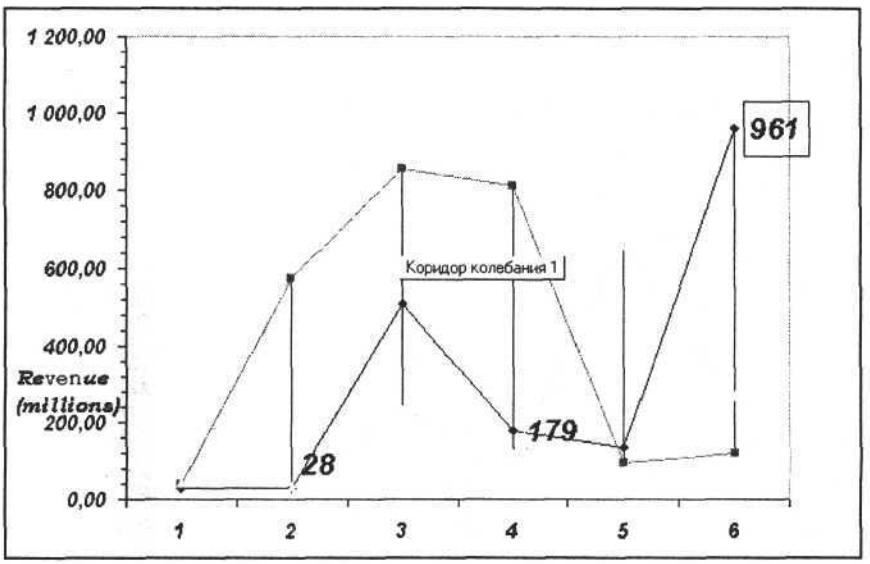


Рис. 14.24. Коридор колебания для диаграммы из трех графиков

Полосы понижения и повышения

Полосы понижения и повышения отображают сравнительную характеристику двух (и более) рядов данных диаграммы в точках построения. Для настройки этих элементов предназначены объекты UpBars и DownBars, принадлежащие элементу коллекции ChartGroups, а отображение этих элементов обеспечивается записью значения True в свойство HasUpDownBars объекта ChartGroup.

Следующая процедура позволяет отобразить полосы понижения и повышения на диаграмме в виде графиков, а также настроить цвет их линий границы.

Отображение полос понижения и повышения и настройка цвета их линий границы

```
procedure TOKBottomDlg11.HasUpDownBarsClick(Sender: TObject);
 var ChartGroup:variant;
```

```
begin
  Chart.ChartType:=xlLine;
  ChartGroup:=Chart.ChartGroups.Item(1);
  ChartGroup.HasUpDownBars:=HasUpDownBars.Checked;
  if HasUpDownBars.Checked then begin
    ChartGroup.UpBars.Border.ColorIndex:=7;
    ChartGroup.DownBars.Border.ColorIndex:=12;
  end;
end;
```

Результат представлен на рис. 14.25.

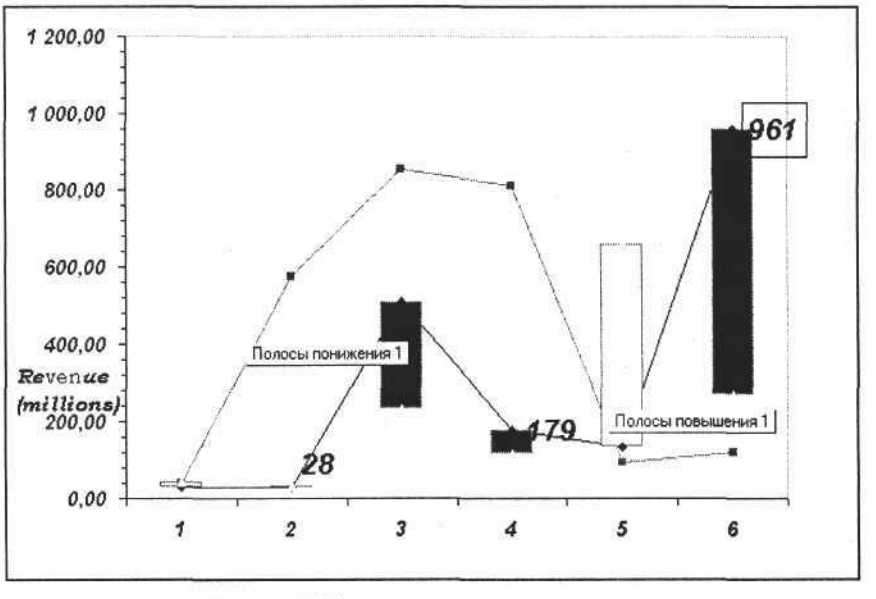


Рис. 14.25. Полосы повышения и понижения

Некоторые дополнительные элементы рядов

В предыдущих разделах мы рассмотрели элементы, отражающие сравнительные характеристики двух и более рядов данных диаграммы. Здесь мы рассмотрим некоторые дополнительные элементы, относящиеся к отдельным рядам данных диаграммы.

Линии выноски для подписей данных

Линии выноски для подписей данных являются элементом ряда данных диаграммы, поэтому их отображение определяется значением свойства

HasLeaderLines объекта Series, являющегося элементом коллекции SeriesCollection. Если значение этого свойство равно True, то линии выноски подписей будут отображены, но только в случае, если будет выбран и соответствующий тип диаграммы, поскольку эти элементы можно отобразить не для всех типов диаграмм. Например, для круговой объемной диаграммы линии выноски подписей могут быть отображены и доступны.

Доступ к свойствам самих линий осуществляется посредством объекта LeaderLines, свойства и методы которого позволяют выделить эти элементы (Select), удалить их (Delete) или настроить свойства линии (тип, толщину и цвет), которые идентичны свойствам обычных линий и определяются объектом Border.

Следующий пример позволяет изменить тип диаграммы, отобразить линии выноски для подписей данных и настроить их свойства.

Задание линий выноски подписей данных диаграммы

```
Const xl3DPieExploded=70;
procedure TOKBottomDlg11.HasLeaderLinesClick(Sender: TObject);
 var Series:variant;
begin
  Chart.ChartType:=xl3DPieExploded;
  Series:=Chart.SeriesCollection(1);
  Series.HasDataLabels:=HasLeaderLines.Checked;
  if HasLeaderLines.Checked then begin
    Series.Points(1).DataLabel.Left:=SeriesPoints(1).DataLabel.Left+20;
    Series.HasLeaderLines:=HasLeaderLines.Checked;
    Series.LeaderLines.Border.ColorIndex:=5;
  end;
end;
```

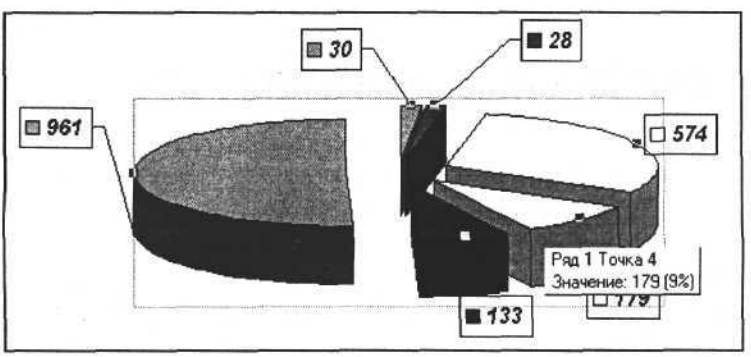


Рис. 14.26. Линии выноски для подписей данных на круговой объемной диаграмме

На рис. 14.26 отображен результат выполнения представленной процедуры — круговая объемная диаграмма, подписи данных которой соединены линиями выноски с графическими элементами самой диаграммы.

Полоса погрешностей

Полоса погрешностей (планки погрешностей) представляет собой линии в точках построения, отображающие потенциальную ошибку (или степень недостоверности) каждой точки данных ряда данных. Полосы погрешностей строятся в точках ряда, могут иметь различные тип, толщину и цвет линий, а также величину отклонения, которая выражается как расстояние между точкой данных и конечными точками линии полосы погрешностей. Доступ к полосе погрешностей обеспечивает объект ErrorBars, принадлежащий элементу коллекции Series. Отображение полос погрешностей обеспечивается путем записи значения True в свойство HasErrorBars и последующей настройкой свойств объекта ErrorBars или использованием метода ErrorBar, принадлежащего элементу коллекции Series. Синтаксис вызова метода ErrorBar:

```
Series.ErrorBar(Direction, Include, Type, Amount, MinusValues);
```

где Direction:Integer — направление построения линий; Include:Integer — признак наличия только отрицательных, только положительных, обоих типов или отсутствие полос погрешностей; Type:Integer — тип отклонения — фиксированный, в процентах (%) от значения и т. д.; Amount, MinusValues:Extended — абсолютные или относительные величины положительного и отрицательного отклонений.

Рассмотрим следующие процедуры.

Построение полосы погрешностей

```
procedure TOKBottomDlg11.Button3Click(Sender: TObject);
begin
  Chart.ChartType:=xlLine;
  Chart.SeriesCollection(1).ErrorBar(Direction:=xlY,
                    Include:=xlErrorBarIncludeBoth,
                    Type:=xlErrorBarTypeFixedValue,
                    Amount:=200);
end;

procedure TOKBottomDlg11.HasErrorBarsClick(Sender: TObject);
begin
  Chart.ChartType:=xlLine;
  Chart.SeriesCollection(1).HasErrorBars:=HasErrorBars.Checked;
```

```
  if HasErrorBars.Checked then begin
    Chart.SeriesCollection(1).ErrorBars.EndStyle:=xlNoCap;
    Chart.SeriesCollection(1).ErrorBars.Border.ColorIndex:=13;
    Chart.SeriesCollection(1).ErrorBars.Border.Weight:=3;
  end;
end;
```

Обе процедуры обеспечивают построение полос погрешностей, которые могут выглядеть так, как показано на рис. 14.27.

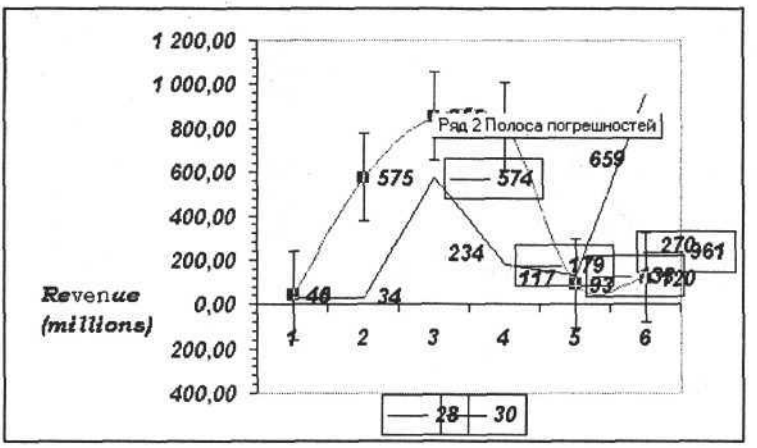


Рис. 14.27. Построение полосы погрешностей для графика

Линия тренда

Диаграмма строится по точкам, которые соединяются между собой прямыми или кривыми линиями, и дает нам представление о процессе, который она описывает. Но не всегда такое представление является наглядным и не всегда дает общую картину законов развития этого процесса. Известно, что точки любого графика, если он описывает реальный процесс, связаны между собой и поддаются математическому описанию. Например: мы хотим выяснить, какова общая тенденция описываемого процесса, как он будет изменяться, как наиболее точно определить промежуточные значения графика? Для этого используем методы аппроксимации и строим на этой основе дополнительный график, отвечающий на наши вопросы. Для диаграмм Excel такие дополнительные графики называются *линиями тренда*. Линию тренда, построенную на одном из методов аппроксимации, можно добавить к двухмерной диаграмме без накопления, например, к графику или гистограмме.

Линии тренда диаграммы объединены в коллекцию Trendlines. Метод Add этой коллекции позволяет добавить новую линию тренда. Используем этот

метод и добавим две линии тренда с помощью следующей процедуры. Результат ее выполнения показан на рис. 14.28.

Добавление линии тренда

```
const xlLinear=-4132;
xlPolynomial=3;
procedure TOKBottomDlg11.Button1Click(Sender: TObject);
begin
Chart.ChartType:=xlLine;
Chart.SeriesCollection(1).Trendlines.Add(Type:=xlLinear, Forward:=1,
                                         Backward:=0.5,
                                         DisplayEquation:=False,
                                         DisplayRSquared:=False);
Chart.SeriesCollection(1).Trendlines.Add(Type:=xlPolynomial, Order:=2,
                                         Forward:=1, Backward:=0.5,
                                         DisplayEquation:=False,
                                         DisplayRSquared:=False);
Chart.SeriesCollection(1).Trendlines(1).Border.ColorIndex:=3;
Chart.SeriesCollection(1).Trendlines(2).Border.ColorIndex:=11
end;
```

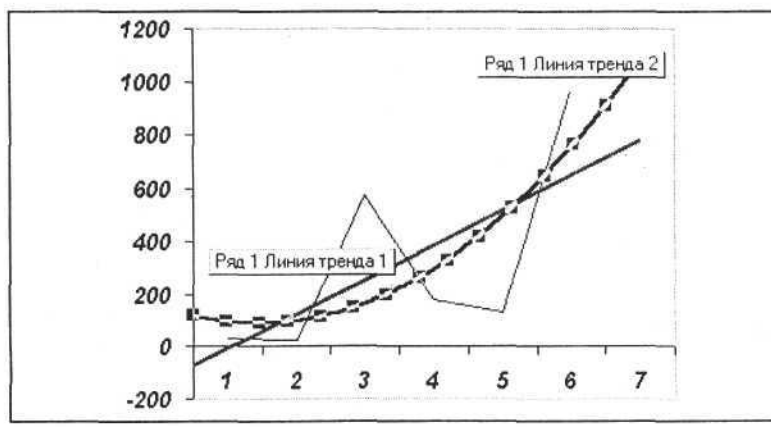


Рис. 14.28. Линейная и полиномиальная линии тренда

Можно добавить несколько пояснений. В нашем примере мы создали две линии тренда — линейную и полиномиальную, при этом для задания типа аппроксимации использовали первый аргумент метода Add. Второй и третий аргументы определяют прогноз поведения описываемого процесса после последней и до первой точек ряда. Следующие два аргумента позволяют отобразить формулы, на основании которых строились линии тренда. Мы

имеем возможность создать несколько аппроксимаций одного и того же процесса (см. рис. 14.28). Для удаления любой линии тренда достаточно вызвать метод Delete для выбранного элемента коллекции Trendlines, как показано в следующей процедуре.

Удаление линии тренда

```
procedure TOKBottomDlg11.Button2Click(Sender: TObject);
begin
  Chart.SeriesCollection(1).Trendlines(1).Delete;
end;
```

В этой главе мы постарались рассмотреть основные принципы построения диаграмм в Excel. Более детальную и полную информацию по этой теме можно получить только практическим путем — применяя диаграммы в ваших приложениях, создаваемых в Delphi.

ГЛАВА 15

Печать

В этой главе мы рассмотрим, как и какими методами осуществляются настройка, предварительный просмотр и печать документов в формате рабочих книг Excel. Особенностью печати рабочей книги Excel является необходимость согласования параметров страницы и свойств конкретного принтера, установленного в системе. Эти особенности учитываются и находят свое отражение в свойствах (параметрах) страницы, доступ к которым обеспечивается путем использования объекта PageSetup. Сам этот объект принадлежит объекту "лист", в свою очередь принадлежащему объекту "рабочая книга". После настройки страницы можно просмотреть, как она будет выглядеть на бумаге, и приступить к печати, задав имя принтера и другие параметры печати.

Сначала рассмотрим формирование той информации, которая будет включена в печатную страницу. Информация для печатной страницы задается путем разделения листа рабочей книги. Для этого используется понятие "разрыв страницы".

Разрыв страницы

Лист рабочей книги можно "поделить" на печатные страницы с помощью линий разрыва страницы, обозначающих, каким образом информация листа будет перенесена на печатные страницы при печати документа. Для того чтобы отобразить эти линии, достаточно переключить "вид" документа в режим "разметка страницы".

В приложениях Delphi это можно сделать путем изменения значения свойства View объекта ActiveWindow (см. следующую процедуру).

Переключение вида документа в режим "разметка страницы"

```
const
xlNormalView=1;
xlPageBreakPreview=2;
```

```
procedure TOKBottomDlg3.ViewChange(Sender: TObject);
begin
  case View.ItemIndex of
  0:Form1.E.ActiveWindow.View:=xlNormalView;
  1:Form1.E.ActiveWindow.View:=xlPageBreakPreview;
  end;
end;
```

Используя константы xlNormalView и xlPageBreakPreview, изменяем вид документа и преобразуем его из обычного в тот вид, на котором отображена разметка страниц. На рис. 15.1 показан внешний вид листа рабочей книги в режиме "разметка страницы".

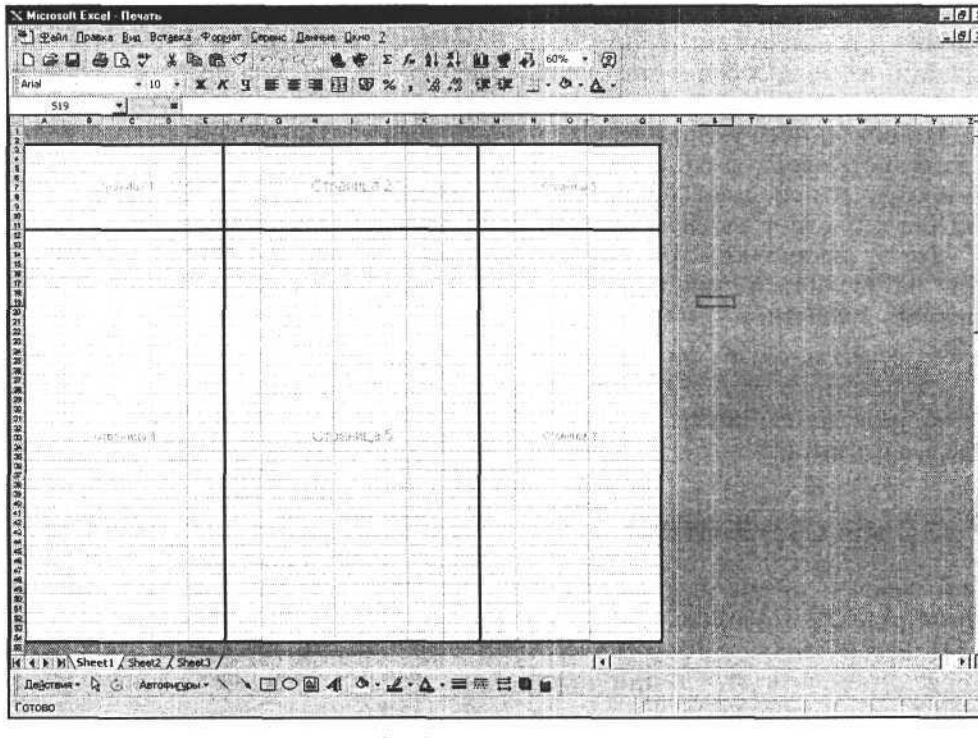


Рис. 15.1. Лист рабочей книги в режиме "разметка страницы"

Используя свойства и методы коллекций HPageBreaks и VPageBreaks листа рабочей книги, мы получаем доступ и возможность манипулировать разметкой листа. Элементы этих коллекций содержат информацию о разрывах страниц и позволяют изменять или удалять их. Добавление новых элементов в коллекцию осуществляется с помощью метода Add, аргументом которого является ссылка на ячейку, до которой вставляется разрыв. Следующая

процедура позволяет добавить разрыв в точке расположения ячейки с координатами 10:30, после чего загрузить в компонент ListBox1 номера всех элементов коллекции, в том числе и вновь созданного.

Добавление разрыва страницы

```
procedure TOKBottomDlg3.Button2Click(Sender: TObject);
 var a_:integer;
begin
  HPageBreaks.Add(Before:=Form1.E.ActiveSheet.Cells[10,30]);
  ListBox1.Items.Clear;
  for a_:=1 to HPageBreaks.Count do ListBox1.Items.Add(inttostr(a_));
  ListBox1.ItemIndex:=0;
end;
```

Для задания точки разрыва можно использовать не только ссылку на ячейку Cells, но и ссылку на область ячеек Range. Тогда первый оператор описанной выше процедуры будет выглядеть так:

```
HPageBreaks.Add (Before:=Form1.E.ActiveSheet.Range[Range.Text]);
```

Здесь объект Range.Text задает адрес области ячеек. Результат добавления новой точки разрыва страницы в обоих случаях будет одним и тем же (рис. 15.2).

Для создания и манипулирования вертикальными разрывами используются элементы коллекции VPageBreaks. Методы этой коллекции идентичны методам коллекции HpageBreaks. Рассмотрим еще одно свойство элементов коллекций VpageBreaks и HpageBreaks. Это свойство определяет положение линий разрыва и позволяет изменять его. Свойство Location является ссылкой на ячейку. Изменение ссылки с одной на другую ячейку приведет к изменению положения линии разрыва.

Перенос линии разрыва страницы

```
procedure TOKBottomDlg3.ButtonLocationClick(Sender: TObject);
 var a_:integer;
begin
  HPageBreaks.Item[1].Location:=Form1.E.ActiveSheet.Range[Range.Text];
  VPageBreaks.Item[1].Location:=Form1.E.ActiveSheet.Range[Range.Text];
end;
```

Использование этой процедуры позволит изменять границы листа, выводимого на печать. После того как мы определили области, которые будут размещаться на отдельных печатных страницах, перейдем к настройке свойств самой страницы.

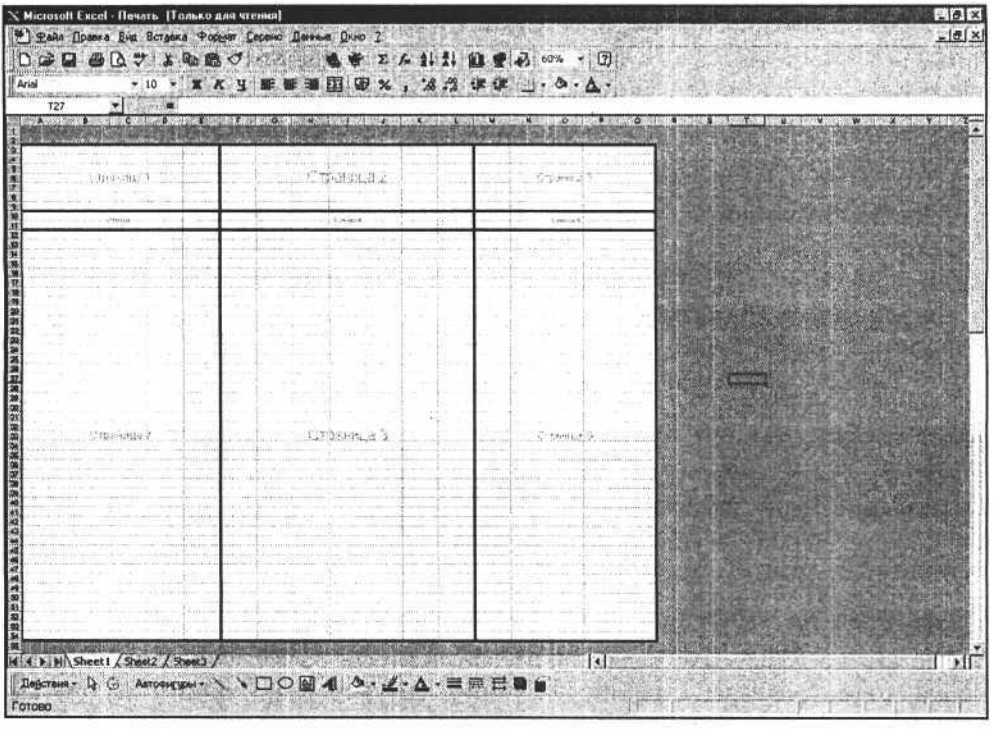


Рис. 15.2. Добавляем горизонтальный разрыв

Объект PageSetup

Для настройки свойств страницы предназначен объект PageSetup.

Чтобы настроить страницу для печати, мы используем диалоговое окно **Параметры страницы**, позволяющее пользователю задать размер бумаги, полей и другие параметры, необходимые для согласования настроек документа с параметрами принтера, на котором он будет распечатан. Все эти свойства объединены в объекте PageSetup и могут быть заданы и изменены программным путем, в том числе и из приложений Delphi (табл. 15.1).

Таблица 15.1. *Свойства объекта PageSetup*

Свойство	Тип	Значение
PrintTitleRows	String	Сквозные строки
PrintTitleColumns	String	Сквозные столбцы
PrintArea	String	Область печати

Таблица 15.1 *(окончание)*

Свойство	Тип	Значение
LeftHeader	String	Верхний колонтитул (левая часть)
CenterHeader	String	Верхний колонтитул (центральная часть)
RightHeader	String	Верхний колонтитул (правая часть)
LeftFooter	String	Нижний колонтитул (левая часть)
CenterFooter	String	Нижний колонтитул (центральная часть)
RightFooter	String	Нижний колонтитул (правая часть)
LeftMargin	Extended	Левое поле
RightMargin	Extended	Правое поле
TopMargin	Extended	Верхнее поле
BottomMargin	Extended	Нижнее поле
HeaderMargin	Extended	Поле для верхнего колонтитула
FooterMargin	Extended	Поле для нижнего колонтитула
PrintHeadings	Boolean	Печать заголовков
PrintGridlines	Boolean	Печать линий сетки
PrintComments	Integer	Печать примечаний
PrintQuality	Array(X, Y)	Качество печати
CenterHorizontally	Boolean	Центрировать на странице горизонтально
CenterVertically	Boolean	Центрировать на странице вертикально
Orientation	Integer	Книжная/альбомная ориентация страницы
Draft	Boolean	True — черновая печать
PaperSize	Integer	Размер бумаги
FirstPageNumber	Integer	Номер первой страницы
Order	Integer	Последовательность вывода страниц
BlackAndWhite	Boolean	Черно-белый режим печати
Zoom	Boolean/Integer	False — масштаб задается в количестве страниц; число обозначает масштаб в процентах от натуральной величины
FitToPagesWide	Integer	Масштаб печати — максимальное количество страниц в ширину
FitToPagesTall	Integer	Масштаб печати — максимальное количество страниц в высоту

В приложении на сопроводительном компакт-диске книги есть примеры использования всех представленных свойств объекта PageSetup. Здесь мы рассмотрим только те свойства, которые используются наиболее часто и дают четкое представление о настройках параметров страницы.

Задание области печати

Лист рабочей книги может содержать огромное количество информации и достаточно велик. К тому же не всегда нужно печатать всю информацию листа. Достаточно (а часто — требуется) выводить на печать только определенную область ячеек. Значит, нужна возможность задавать эту область. Для этого предназначено свойство PrintArea объекта PageSetup, содержащее строку адреса прямоугольной области. Изменяя значение PrintArea, мы изменяем область печати.

Задание области печати

```
procedure TOKBottomDlg2.PrintAreaExit(Sender: TObject);
begin
  PageSetup.PrintArea:=PrintArea.Text;
end;
```

Задание полей страницы

Используя свойства объекта PageSetup, мы можем задать величину полей печатной страницы, а также положение верхнего и нижнего колонтитулов. Это достигается записью значений этих полей в свойства LeftMargin, RightMargin, TopMargin, BottomMargin, HeaderMargin, FooterMargin. Следующие процедуры позволяют приложению Delphi задать левое и правое поля страницы.

Задание размеров полей страницы

```
procedure TOKBottomDlg2.LeftMarginChange(Sender: TObject);
begin
  PageSetup.LeftMargin:=LeftMargin.Value;
end;

procedure TOKBottomDlg2.RightMarginChange(Sender: TObject);
begin
  PageSetup.RightMargin:=RightMargin.Value;
end;
```

В нашем примере величина полей задается в произвольных единицах, т. е. в пикселах. Чтобы привязать все размеры к реальным единицам измерения, нужно выполнить преобразование. В среде Visual Basic такие преобразования выполняют функции InchesToPoints и CentimetersToPoints, преобразующие величину, выраженную в дюймах и сантиметрах, в количество точек. Если эти функции недоступны из приложений Delphi, то используем следующие соотношения — в одной точке 0,035 см или 1/72 дюйма.

Колонтитулы

Колонтитулы представляют собой надписи на печатной странице в ее верхней части и нижней части. Кроме номера страницы, они могут содержать произвольный текст, текст, связанный с документом и его свойствами, а также текущую дату и время. Колонтитулы могут также представлять собой комбинации этих данных. Печатная страница может содержать верхний и нижний колонтитулы. Доступ к их содержимому осуществляется через шесть свойств объекта PageSetup, т. к. каждый колонтитул может содержать до трех частей: LeftHeader — левая часть верхнего колонтитула, CenterHeader — центральная часть верхнего колонтитула, RightHeader — правая часть верхнего колонтитула, LeftFooter — левая часть нижнего колонтитула, CenterFooter — центральная часть нижнего колонтитула, RightFooter — правая часть нижнего колонтитула. Просмотреть содержимое колонтитулов можно только на распечатанной странице или в режиме "предварительный просмотр", а запрограммировать их возможно в обычном режиме, используя соответствующие свойства объекта PageSetup.

Добавление верхнего колонтитула

```
procedure TOKBottomDlg2.LeftHeaderChange(Sender: TObject);
begin
  PageSetup.LeftHeader:=LeftHeader.Text;
end;

procedure TOKBottomDlg2.CenterHeaderChange(Sender: TObject);
begin
  PageSetup.CenterHeader:=CenterHeader.Text;
end;

procedure TOKBottomDlg2.RightHeaderChange(Sender: TObject);
begin
  PageSetup.RightHeader:=RightHeader.Text;
end;
```

На рис. 15.3 представлена форма, использующая приведенные процедуры, из состава приложения, находящегося на сопроводительном компакт-диске книги. С помощью этой формы можно экспериментировать со шрифтами колонтитулов и их содержанием.

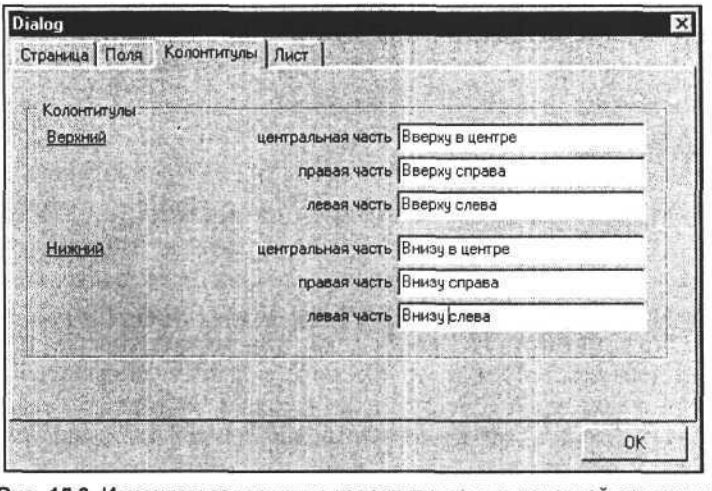


Рис. 15.3. Изменяем содержание колонтитулов для печатной страницы

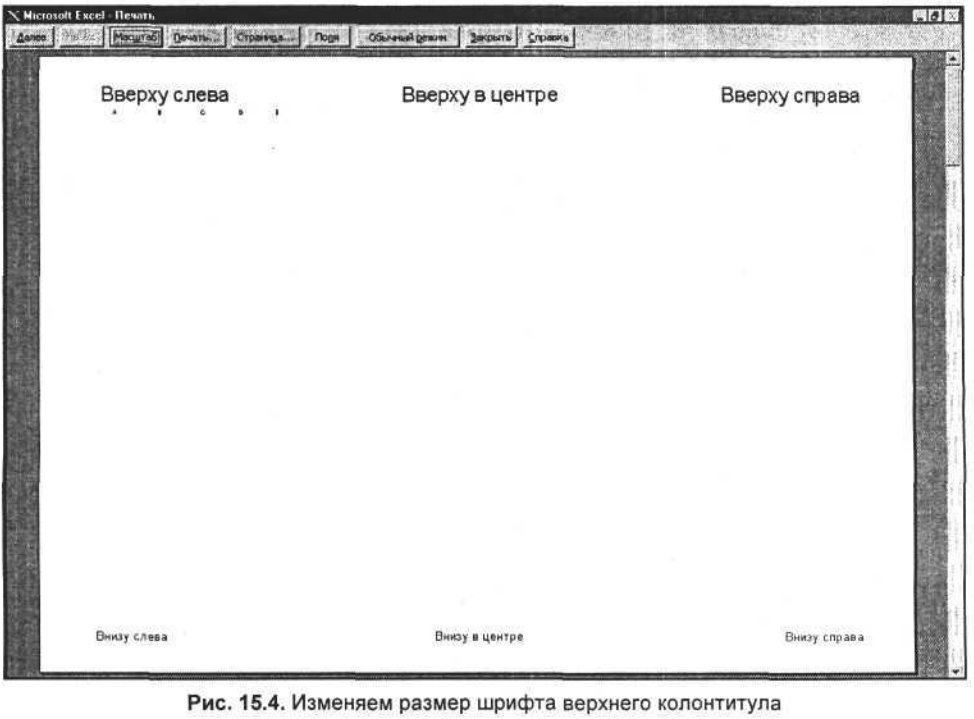


Рис. 15.4. Изменяем размер шрифта верхнего колонтитула

Используя форму, представленную на рис. 15.3, попробуем изменить шрифт текста и тип содержимого колонтитулов. Например, можно записать в части колонтитулов следующие последовательности символов для отображения специальной информации в колонтитуле:

- '&D' — дата;
- '&T' — время;
- '&F' — имя файла.

Комбинация из символа & и числа задает размер шрифта для текста в соответствующей части колонтитула. Например, оператор PageSetup.LeftHeader:= '&28'; установит размер шрифта в 28 пунктов для текста в левой части верхнего колонтитула.

Используя эти особенности, установим для верхнего колонтитула новый размер шрифта, равный 28 пунктов. Результат представлен на рис. 15.4.

Ориентация и размер бумаги, номер первой страницы, масштаб

Ориентация страницы (книжная или альбомная) определяется значением свойства Orientation. Программно такой выбор осуществляется путем записи в свойство Orientation одного из значений — xlPortrait=1 или xlLandscape=2.

Размер бумаги задается свойством PaperSize объекта PageSetup. Список возможных значений этого свойства представляет собой набор из нескольких десятков констант, соответствующих различным типовым размерам бумаги. Полный список констант можно найти в приложении на сопроводительном компакт-диске книги или в справочной системе Visual Basic. В следующем примере представлена только часть этого списка.

Выбор ориентации и размера бумаги

```
const
// Константы, определяющие ориентацию бумаги
xlPortrait=1;
xlLandscape=2;
// Константы, определяющие размер бумаги
xlPaperLetter=1;          // Letter (8-1/2 in. x 11 in.)
xlPaperLetterSmall=2;     // Letter Small (8-1/2 in. x 11 in.)
xlPaperTabloid=3;         // Tabloid (11 in. x 17 in.)
xlPaperLedger=4;          // Ledger (17 in. x 11 in.)
xlPaperLegal=5;           // Legal (8-1/2 in. x 14 in.)
xlPaperStatement=6;       // Statement (5-1/2 in. x 8-1/2 in.)
```

```
xlPaperExecutive=7;        // Executive (7-1/2 in. x 10-1/2 in.)
xlPaperA3=8;               // A3 (297 mm x 420 mm)
xlPaperA4=9;               // A4 (210 mm x 297 mm)
xlPaperEnvelopeDL=27;      // Envelope DL (110 mm x 220 mm)

procedure TOKBottomDlg2.OrientationChange(Sender: TObject);
begin
  case Orientation.ItemIndex of
  0:PageSetup.Orientation:=xlPortrait;
  1:PageSetup.Orientation:= xlLandscape;
  end;
end;

procedure TOKBottomDlg2.PaperSizeChange(Sender: TObject);
begin
  case PaperSize.ItemIndex of
  0:PageSetup.PaperSize:=xlPaperLetter;
  1:PageSetup.PaperSize:=xlPaperLetterSmall;
  2:PageSetup.PaperSize:=xlPaperTabloid;
  3:PageSetup.PaperSize:=xlPaperLedger;
  4:PageSetup.PaperSize:= xlPaperEnvelopeDL;
  end;
end;
```

Используя описанные процедуры, зададим альбомную ориентацию и размер бумаги "Envelope DL (110 mm x 220 mm)" (xlPaperEnvelopeDL). Результат представлен на рис. 15.5.

Номер первой страницы печати определяется свойством FirstPageNumber объекта PageSetup. Если оно имеет значение xlAutomatic, то номер первой страницы определяется "по умолчанию". Чтобы задать конкретный номер первой страницы, нужно в свойство FirstPageNumber записать его значение. Следующая процедура позволяет задать номер первой страницы в приложениях Delphi.

Задание номера первой страницы

```
procedure TOKBottomDlg2.AutomaticClick(Sender: TObject);
begin
  FirstPageNumber.Enabled:= not Automatic.Checked;
  if Automatic.Checked
    then PageSetup.FirstPageNumber:=xlAutomatic
    else PageSetup.FirstPageNumber:=FirstPageNumber.Value;
end;
```

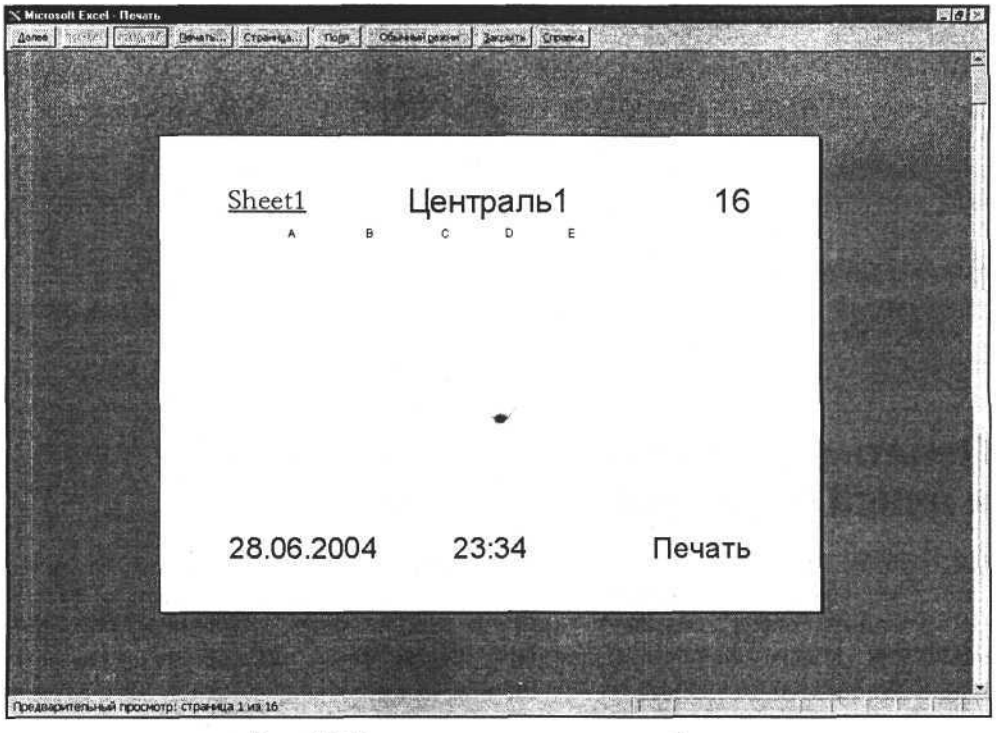


Рис. 15.5. Зададим ориентацию и размер бумаги

Выбор нужного масштаба печати позволяет уменьшить количество страниц заданного размера, необходимых для печати документа. Есть два способа задания масштаба.

Первый основан на задании величины масштаба в процентах от натуральной величины. Свойство Zoom объекта PageSetup позволяет реализовать этот способ. Для этого достаточно записать в него величину масштаба. Обычно масштаб можно задать целым числом в диапазоне от 10 до 400 (%).

При Zoom=False используется второй способ, основанный на задании максимального количества страниц в ширину (свойство FitToPagesWide) и в высоту (свойство FitToPagesTall) для печатаемого документа.

Следующие процедуры позволяют реализовать оба способа в приложениях Delphi.

Задание масштаба печати

```
procedure TOKBottomDlg2.ZoomChange(Sender: TObject);
begin
  if Zoom.Value<>100
```

```
    then PageSetup.Zoom:=Zoom.Value else PageSetup.Zoom:=False;
end;

procedure TOKBottomDlg2.FitToPagesWideChange(Sender: TObject);
begin
  PageSetup.FitToPagesWide:=FitToPagesWide.Value;
end;

procedure TOKBottomDlg2.FitToPagesTallChange(Sender: TObject);
begin
  PageSetup.FitToPagesTall:=FitToPagesTall.Value;
end;
```

Печать заголовков строк и столбцов и линий сетки, черновая печать

Настройки страницы печати позволяют выбрать режим вывода, когда печатается не только содержимое ячеек, но и заголовки столбцов и строк, а также линии сетки. Установив свойство PrintHeadings в значение True, мы выбираем режим печати заголовков. Свойство PrintGridlines позволяет управлять режимом вывода на печать линий сетки. Рассмотрим пример использования этих свойств.

Печать заголовков строк и столбцов и линий сетки

```
procedure TOKBottomDlg2.PrintHeadingsClick(Sender: TObject);
begin
  PageSetup.PrintHeadings:=PrintHeadings.Checked;
end;

procedure TOKBottomDlg2.PrintGridlinesClick(Sender: TObject);
begin
  PageSetup.PrintGridlines:=PrintGridlines.Checked;
end;
```

На рис. 15.6 представлен пример использования представленных процедур.

В режиме "черновая печать" линии сетки, а также заголовки строк и столбцов выводиться на печать не будут. Этот режим определяется состоянием свойства Draft объекта PageSetup, имеющего логический тип. Черно-белая печать определяется свойством BlackAndWhite объекта PageSetup и позволяет осуществлять печать в оттенках серого цвета.

По ходу задания параметров печати страницы можно в любой момент перейти к предварительному просмотру печати. Режим предварительного про-

смотра связан с параметрами конкретного выбранного принтера и недоступен, если не выбран ни один принтер. Для перехода в режим предварительного просмотра используется метод PrintPreview. Этот метод есть не только у листа рабочей книги Excel, но и у других объектов, которые могут быть самостоятельно выведены на печать.

Рис. 15.6. Задан режим печати заголовков и линий сетки

Предварительный просмотр и печать объектов рабочей книги Excel

Excel позволяет вывести на печать или в окно предварительного просмотра объект, у которого есть метод PrintPreview или PrintOut. Таким объектом, например, может быть диаграмма, активный лист рабочей книги или область ячеек листа рабочей книги.

Рассмотрим пример. Следующая процедура позволяет выбрать в качестве объекта диаграмму, а затем с помощью метода ObjectForPrint вывести ее в окно предварительного просмотра.

Выбор и предварительный просмотр диаграммы

```
procedure TForm1.Button9Click(Sender: TObject);
begin
  ObjectForPrint:=E.ActiveChart;
  ObjectForPrint.PrintPreview;
end;
```

Результат выполнения данной процедуры представлен на рис. 15.7.

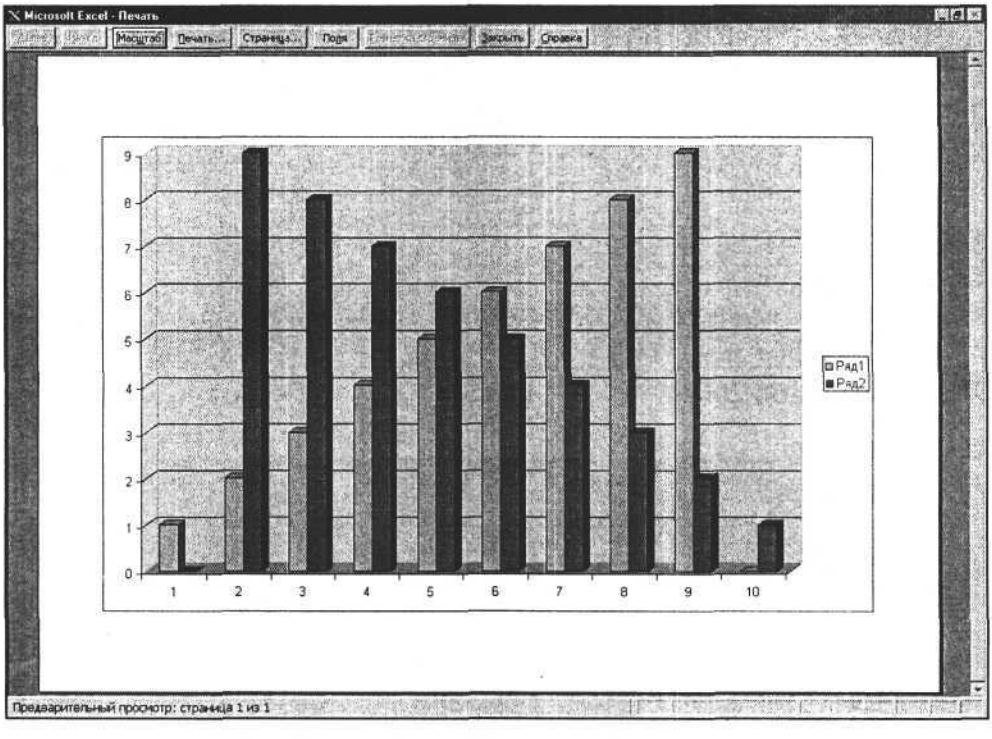


Рис. 15.7. Предварительный просмотр выбранной диаграммы

Печать документа

После настройки и просмотра документа можно перейти к его печати. Печать документа осуществляется методом PrintOut. Синтаксис вызова этого метода:

```
PrintOut(from, To, Copies, Preview, ActivePrinter, PrintToFile, Collate);
```

Аргументы метода, их тип и назначение представлены в табл. 15.2.

Таблица 15.2. *Аргументы метода PrintOut*

Аргумент	Тип	Назначение
From	Integer	Номер страницы, с которой начинается печать
To	Integer	Номер страницы, на которой заканчивается печать
Copies	Integer	Количество копий печати
Preview	Boolean	True — предварительный просмотр перед печатью
ActivePrinter	String	Имя принтера для печати
PrintToFile	Boolean	True — печатать в файл
Collate	Boolean	True — разобрать по копиям

При использовании метода PrintOut без аргументов печать будет выполнена в режиме "по умолчанию". Этот метод можно использовать с любым набором аргументов. Если в системе установлен не один, а несколько принтеров, то, чтобы использовать один из них, нужно при вызове метода PrintOut задать в аргументе ActivePrinter его имя. Чтобы получить список установленных в системе принтеров, используем стандартный объект Delphi TPrinter (см. следующую процедуру).

Получение списка принтеров, установленных в системе

```
procedure TForm1.Button1Click(Sender: TObject);
 var a_:integer;
     pr:TPrinter;
begin
  ListBox1.Items.Clear;
  pr:=TPrinter.Create;
  for a_:=0 to pr.Printers.Count-1 do
     ListBox1.Items.Add(pr.Printers[a_]);
  pr.Free;
end;
```

На рис. 15.8 представлен компонент ListBox (список) с загруженным перечнем установленных в системе принтеров.

Чтобы для печати документа использовать один из принтеров, имя которого находится в списке ListBox, достаточно выполнить следующую процедуру.

Выбор активного принтера и печать документа

```
procedure TForm1.Button11Click(Sender: TObject);
begin
```

```
  E.ActivePrinter:=ListBox1.Items.Strings[ListBox1.ItemIndex];
  E.ActiveWorkBook.PrintOut(ActivePrinter:=
                  ListBox1.Items.Strings[ListBox1.ItemIndex]);
end;
```

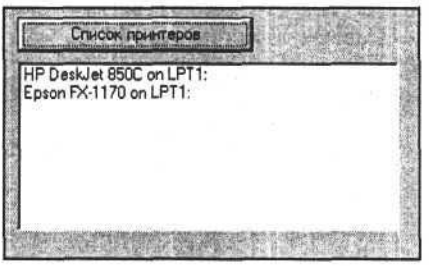


Рис. 15.8. Список установленных в системе принтеров

Важно!

При выполнении этой процедуры возможны ошибки, связанные с отличием форматов представления имени принтера в объекте TPrinter и имени, используемого при вызове метода PrintOut. Например, если имя принтера для объекта TPrinter представляет собой строку:

```
"HP DeskJet 850C on LPT1:",
```

то при вызове метода PrintOut необходимо использовать несколько видоизменненную строку:

```
"HP DeskJet 850C (LPT1:)".
```

ГЛАВА 16

Программирование свойств MS Excel

В этой главе рассматриваются следующие темы:

- элементы управления приложения MS Excel;
- элементы коллекции CommandBars;
- создание пользовательской панели (меню);
- элементы управления и их свойства;
- главное меню;
- создание элемента управления;
- использование Visual Basic Editor;
- коллекция диалогов;
- пример программирования панели.

Элементы управления приложения MS Excel

В предыдущих главах были рассмотрены модели рабочей книги Excel и элементов, которые входят в состав рабочей книги и служат для отображения той или иной информации. Использование этих элементов позволяет не только отображать информацию, но и обмениваться информацией с внешними приложениями. В свою очередь, внешние приложения, используя эти объекты, имеют доступ к воздействию на них и изменению их формы и содержания. Таким образом, подводя итог вышесказанному, можно утверждать, что, используя свойства и объекты коллекции WorkBooks, мы можем из приложений Delphi создавать документы любой сложности, а приложение Excel использовать как генератор и редактор готовых документов, например, отчетов. Конечно, возможности MS Excel этим не исчерпываются.

Панели и элементы управления, различные диалоги также доступны для внешних приложений и могут управляться с их помощью. Внешним приложениям доступны также макросы, которые можно создавать, модифицировать и удалять, используя свойства и методы объекта Excel.Application.

Рассмотрим элементы управления главного окна приложения MS Excel. Все элементы управления организованы в иерархическую структуру, на вершине которой находится коллекция CommandBars. Элементами этой коллекции являются главное меню и панели с кнопками и другими элементами управления. В свою очередь, панели элементов управления содержат коллекции Controls этих элементов. На рис. 16.1 представлена структура объектной модели коллекции CommandBars.

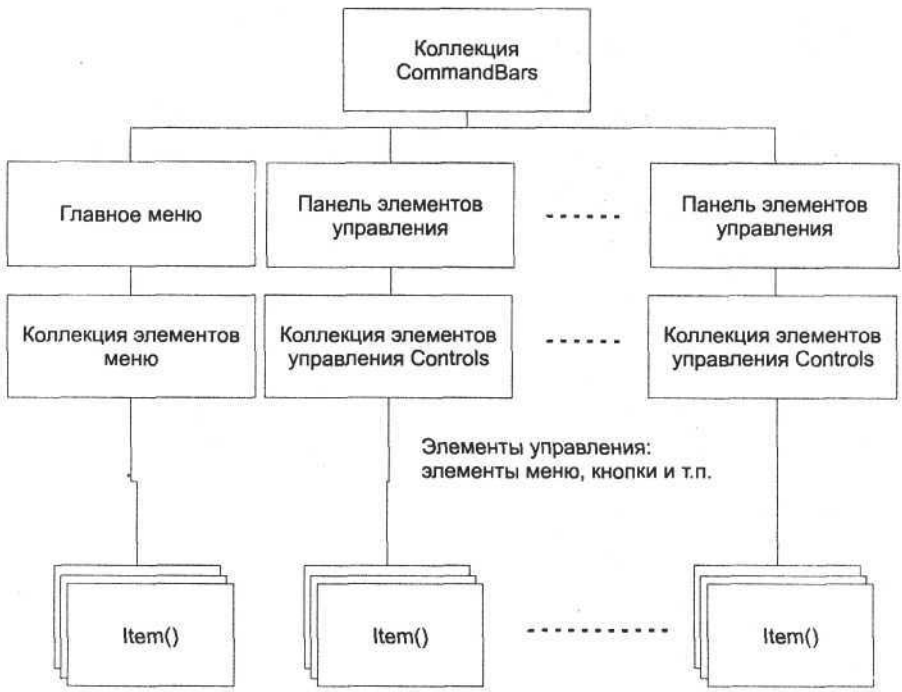


Рис. 16.1. Объектная модель коллекции CommandBars

Элементы коллекции CommandBars

Рассмотрим свойства коллекции CommandBars. Как и любая коллекция, она содержит типичные свойства и методы. Свойство Count содержит количество элементов коллекции, метод Add позволяет добавить новый элемент коллекции (панель элементов управления), метод Item(i) возвращает ссылку на элемент коллекции. К специфическим свойствам относятся те, которые по-

зволяют задавать режим отображения элементов коллекции. В табл. 16.1 представлены свойства и методы коллекции панелей элементов управления.

Таблица 16.1. *Свойства и методы коллекции CommandBars*

Свойство или метод	Тип	Описание
ActionControl	Объект	Ссылка на активный в данный момент элемент управления
ActiveMenuBar	Объект	Ссылка на активный в данный момент элемент меню
Add	Метод	Добавление элемента коллекции
Count	Integer	Количество элементов коллекции
DisplayTooltips	Boolean	True — включить подсказку для кнопок
DisplayKeysInTooltips	Boolean	True — включить в подсказку для кнопок сочетания "горячих" клавиш
FindControl	Метод	Поиск элемента коллекции
Item(i:integer)	Объект	Набор элементов коллекции
LargeButtons	Boolean	Переключение режима отображения больших/маленьких кнопок
MenuAnimationStyle	Integer	Эффект при выводе меню
ReleaseFocus	Метод	Обновление пользовательского интерфейса для всех элементов коллекции

Рассмотрим некоторые свойства коллекции CommandBars, например, свойства, определяющие общие визуальные характеристики панелей и элементов управления. Свойство LargeButtons коллекции позволяет устанавливать размеры кнопок на всех панелях (большие или маленькие). Для выбора большого размера кнопок установим значение этого свойства в True, для выбора малого размера — в False. По умолчанию это свойство имеет значение False.

Следующая процедура позволяет задать большой размер для кнопок.

Изменение размера кнопок

```
procedure TForm1.LargeButtonsClick(Sender: TObject);
begin
  E.CommandBars.LargeButtons:=LargeButtons.Checked;
end;
```

Выполнив данную процедуру, мы получим результат, который не нуждается в комментариях и представлен на рис. 16.2.

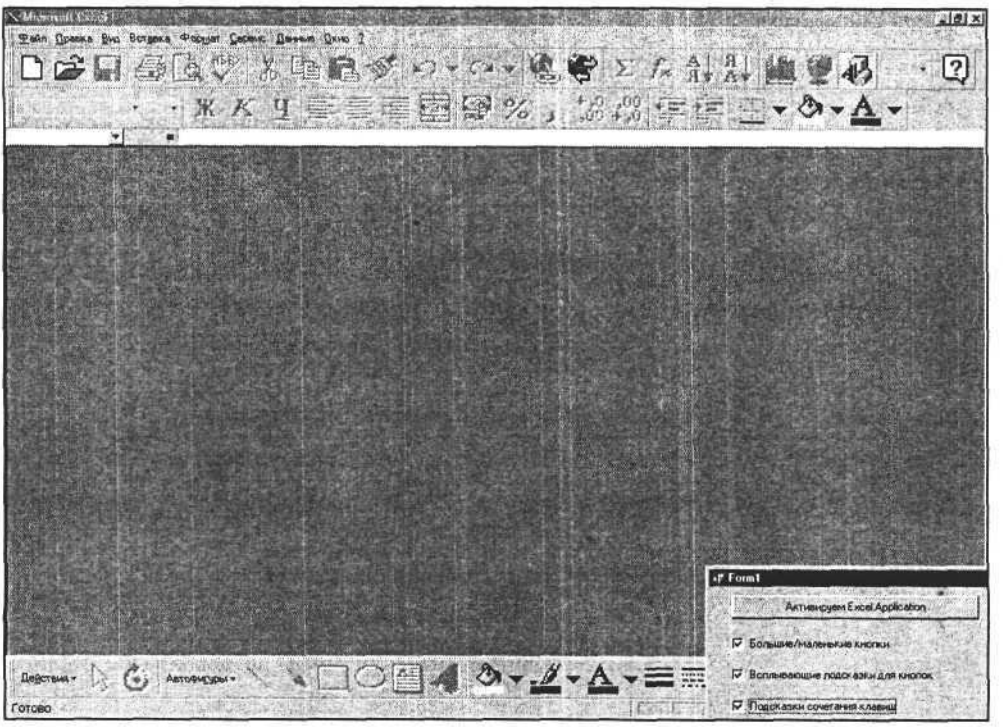


Рис. 16.2. Выбраны большие кнопки для панелей управления

Другими общими свойствами коллекции CommandBars являются два свойства логического типа, управляющие режимом подсказок для кнопок панелей. Свойство DisplayTooltips определяет, включена или отключена всплывающая подсказка. Свойство DisplayKeysInTooltips определяет присутствие в подсказке сочетания "горячих" клавиш (этот режим в Excel не используется).

Изменение отображения режима подсказок определяется установкой этих свойств в определенные состояния, что можно выполнить в приложении Delphi. Рассмотрим следующую процедуру.

Изменение режима подсказок для кнопок

```
procedure TForm1.DisplayTooltipsClick(Sender: TObject);
begin
  E.CommandBars.DisplayTooltips:=DisplayTooltips.Checked;
end;
```

На рис. 16.3 показана форма, использующая данную процедуру, а на рис. 16.4 — главное окно Excel, в котором видна подсказка для кнопки **Открыть**.

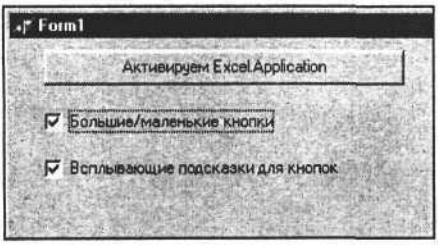


Рис. 16.3. Фрагмент формы приложения Delphi

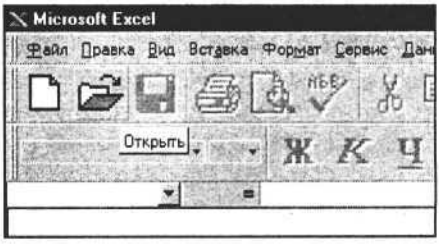


Рис. 16.4. Отображение подсказки для кнопки **Открыть**

Рассмотрим свойства коллекции, отображающие состояние элементов управления, — ActionControl и ActiveMenuBar. Они являются ссылками на объекты-элементы коллекции и позволяют определить, какой элемент панели активен (т. е. какая кнопка запустила на выполнение команду) и какая панель, содержащая меню, активна в настоящее время. Далее представлен исходный текст процедуры, которая возвращает и отображает имя панели, на которой размещено активное меню.

Получение названия активного меню

```
procedure TForm1.Button16Click(Sender: TObject);
 var name_:string;
begin
  name_:=E.CommandBars.ActiveMenuBar.Name;
  messagebox(handle,pchar(name_),'',0);
end;
```

Коллекция CommandBars также включает в себя список панелей управления, доступ к которым осуществляется с помощью метода Item(i), где i — индекс или имя панели. Этот метод возвращает ссылку на элемент коллекции.

Получим список всех элементов коллекции CommandBars. Для этого, используя свойство Count коллекции и свойство Name элемента коллекции, загрузим список элементов в объект типа TCheckListBox (список с флажками). Свойство Name имеет строковый тип и представляет собой имя элемента. Его можно использовать для доступа к любому элементу коллекции так же, как индекс, имеющий числовой тип. Рассмотрим следующую процедуру.

Получение списка панелей

```
procedure TOKBottomDlg2.FormCreate(Sender: TObject);
 var a_:integer;
   eee_:string;
```

```
begin
  COMMANDBARS:=Form1.E.COMMANDBARS;
  for a_:=1 to COMMANDBARS.Count do begin
    eee_:=COMMANDBARS.Item[a_].name+' = '+COMMANDBARS.Item[a_].NameLocal;
    CheckListBox1.Items.Add(eee_);
    CheckListBox1.Checked[a_-1]:=CommandBars.Item[a_].Visible;
  end;
end;
```

Данная процедура, последовательно перебирая все элементы коллекции, загружает их имена в объект CheckListBox1.

После этого можно загрузить значения двух свойств элемента коллекции: NameLocal — определяет имя, отображаемое в заголовке панели, которое соответствует национальной версии Excel (в данном случае — русскоязычной версии); Visible — определяет режим отображения элемента в окне Excel. Далее, после загрузки, перемещаясь по списку имен загруженных в CheckListBox1 элементов, можно получить и изменить значения некоторых свойств выбранной панели. Можно использовать и другие свойства, например Position, Left, Top, Width, Height:Integer (положение и размеры панели) и RowIndex:Integer (номер строки, которую занимает панель, когда панели собраны в какой-либо части главного окна). Если панель содержит кнопки или другие элементы управления, то свойство BuiltIn:Boolean находится в состоянии True, иначе — в состоянии False. Context:String — свойство панели, которое содержит строку со ссылкой на файл шаблона. Свойство Protection:Integer определяет режим защиты панели от изменений со стороны пользователя. Возможные значения этого свойства, а также свойства Position описаны в *Приложении 1*.

Рассмотрим исходный текст процедуры, позволяющей скрывать/отображать выбранные панели с помощью свойства Visible панели.

Выбор панели элементов управления и задание ее отображения

```
procedure TOKBottomDlg2.CheckListBox1Click(Sender: TObject);
begin
  try
  CommandBars.Item[CheckListBox1.ItemIndex+1].Visible:=
               CheckListBox1.Checked[CheckListBox1.ItemIndex];
  except
  CheckListBox1.Checked[CheckListBox1.ItemIndex]:=
            not CheckListBox1.Checked[CheckListBox1.ItemIndex];
  messagebox(handle,'Ошибка изменения свойства Visible!','Внимание!',0);
  end;
end;
```

Обработка исключительных ситуаций позволяет управлять значением свойства Checked в случаях возникновения ошибок. На рис. 16.5 представлена форма, которая отображает список элементов коллекции CommandBars главного окна приложения Excel и позволяет изменять положение панелей управления, выбранных в списке.

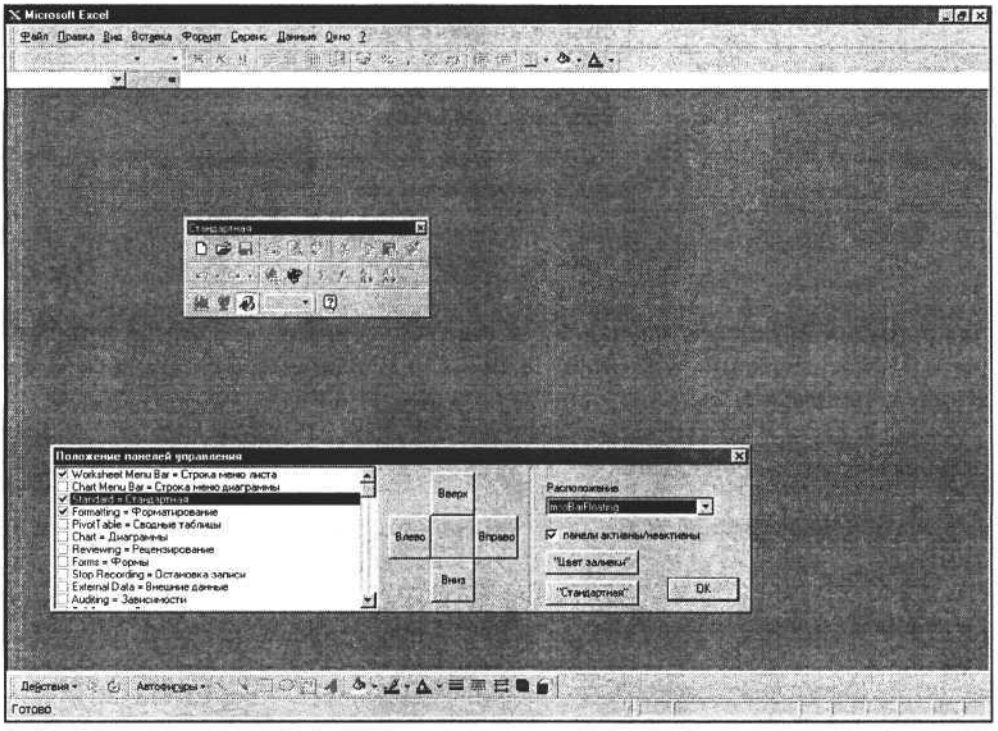


Рис. 16.5. Отображение списка панелей, изменение положения панели

Для управления положением панели предназначено свойство Position. Оно имеет тип Integer, может принимать одно из нескольких значений констант и определяет положение панели в главном окне редактора Excel. Панели могут располагаться вдоль верхней, нижней, левой или правой стороны этого окна, а также в центральной части окна. В табл. 16.2 представлены константы, соответствующие различным положениям панели в главном окне Excel.

Таблица 16.2. *Возможные положения панели*

Константа	Значение	Описание
msoBarFloating	4	Отдельная панель в центральной части главного окна

***Таблица 16.2** (окончание)*

Константа	Значение	Описание
msoBarBottom	3	Панель с кнопками, расположенная вдоль нижней стороны главного окна
msoBarLeft	0	Панель с кнопками, расположенная вдоль левой стороны главного окна
msoBarMenuBar	6	Меню
msoBarPopup	5	Всплывающее меню
msoBarRight	2	Панель с кнопками, расположенная вдоль правой стороны главного окна
msoBarTop	1	Панель с кнопками, расположенная вдоль верхней стороны главного окна

Для изменения положения панели достаточно в свойство Position записать то или иное значение, указанное в табл. 16.2, например, с помощью следующей процедуры.

Изменение положения панели

```
procedure TOKBottomDlg2.PositionClick(Sender: TObject);
begin
try
CommandBars.Item[CheckListBox1.ItemIndex+1].Position:=Position.ItemIndex;
except
Position.ItemIndex:=CommandBars.Item[CheckListBox1.ItemIndex+1].Position;
end;
end;
```

Обратите внимание на то, что в этой процедуре используется обработка исключительной ситуации. Это необходимо в тех случаях, когда положение панели не может быть изменено. В таких случаях состояние объекта Position будет восстановлено в первоначальное состояние. Используя эту процедуру, изменим, положение панели **Стандартная** и поместим ее вдоль левой стороны главного окна (рис. 16.6).

Отметим некоторые особенности расположения панелей в главном окне приложения Excel. Не каждая панель может принимать все возможные положения, для некоторых панелей допустимы только определенные положения, а остальные являются недопустимыми. Для того чтобы выяснить, можно или нельзя изменить положение панели, достаточно проанализировать значение свойства Type элемента коллекции CommandBars. Если значение этого свойства равно msoBarTypeNormal, то панель может быть расположе-

на в центральной части главного окна (как обычное окно) или пристыкована к одной из сторон главного окна (кроме некоторых панелей, например, **Цвет заливки**). Если свойство Type имеет другое значение, то расположить панель можно только определенным образом.

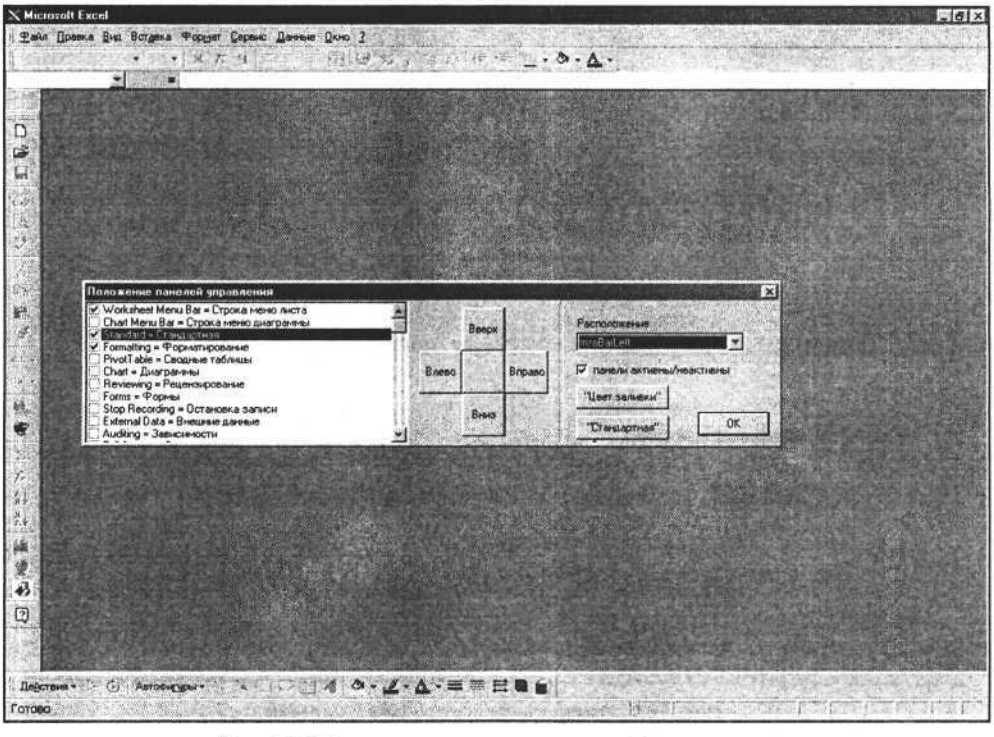


Рис. 16.6. Изменение положения панели **Стандартная**

Панель **Цвет заливки** может находиться только в виде дочернего окна в главном окне приложения (ее нельзя пристыковать к одной из сторон главного окна). При попытке изменить положение этой панели мы получим ошибку выполнения, при этом панель останется на своем месте. В таких случаях желательно обрабатывать ошибку с выводом сообщения пользователю, используя программные скобки try ... except, либо проверять свойство Type и затем изменять позицию панели. В следующем примере используется второй способ.

Использование свойства Type

```
CommandBar:=CommandBars.Item[CheckListBox1.ItemIndex+1];
If CommandBar.Type=msoBarTypeNormal then
                CommandBar.Position:=Position.ItemIndex;
```

На рис. 16.7 отображена панель **Цвет заливки**. Ее нельзя пристыковать к какой-либо стороне главного окна, несмотря на то, что для нее значение свойства Type равно msoBarTypeNormal (т. е. 0).

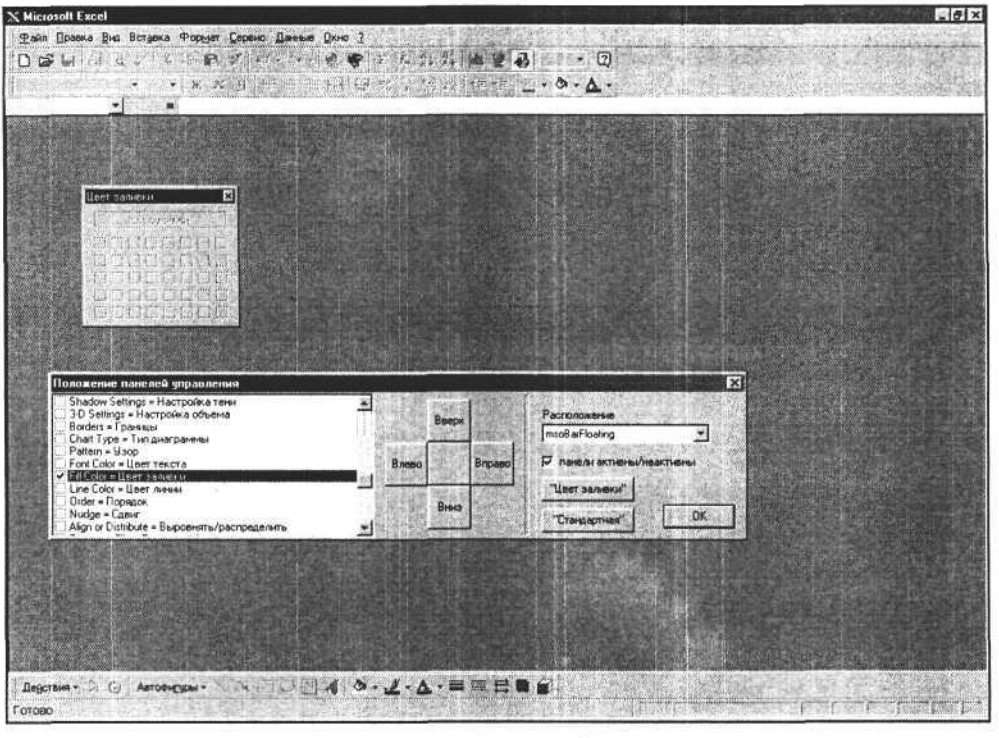


Рис. 16.7. Отображаем панель **Цвет заливки**

В случае, когда панель не может быть размещена вдоль какой-либо стороны главного окна (представляет собой дочернее окно), положение панели определяется только ее координатами. Для изменения координат используем свойства Left и Top.

Изменение горизонтальной координаты панели

```
procedure TOKBottomDlg2.Button3Click(Sender: TObject);
begin
  CommandBar.Left:=CommandBar.Left-10;
end;

procedure TOKBottomDlg2.Button3Click(Sender: TObject);
begin
  CommandBar.Left:=CommandBar.Left+10;
end;
```

Используя представленные процедуры, мы сможем перемещать панель в левую или в правую часть главного окна (рис. 16.8).

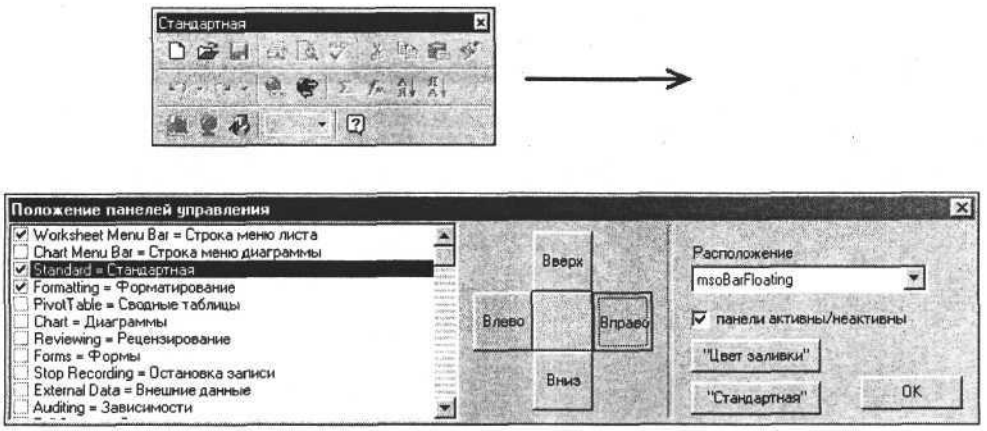


Рис. 16.8. Перемещаем панель вправо

Иногда требуется запретить или ограничить доступ пользователя к той или иной панели. Ограничение доступа к панели может быть как полным, так и частичным. Например, нужно заблокировать для пользователя возможность перемещения панели. Для этих целей используем свойство Protection (тип Integer), при этом способ блокировки определяется значением этого свойства и задается определенными константами или их комбинациями.

Рассмотрим еще одно свойство панели, позволяющее активировать/деактивировать панель в главном окне приложения Excel. Свойство Enabled панелей имеет тип Boolean и определяет возможность доступа пользователя к ним. Когда свойство Enabled имеет значение False, пользователь не может пользоваться данной панелью (она вообще не отображается). Если изменить значение этого поля и установить его в True, то панель будет доступна и займет свое прежнее место в главном окне приложения Excel.

Этими свойствами удобно пользоваться, чтобы блокировать доступ пользователя к некоторым элементам управления. Рассмотрим следующую процедуру.

Отключение отображения панелей управления

```
procedure TOKBottomDlg2.CheckBox1Click(Sender: TObject);
 var a_:integer;
begin
  for a_:=1 to CommandBars.Count do
         CommandBars.Item[a_].Enabled:=CheckBox1.Checked;
end;
```

Данная процедура, перебирая все элементы коллекции CommandBars, устанавливает их свойства Enabled либо в значение True, либо в значение False. Воспользуемся ею и установим для всех панелей свойство Enabled в значение False. Результат представлен на рис. 16.9.

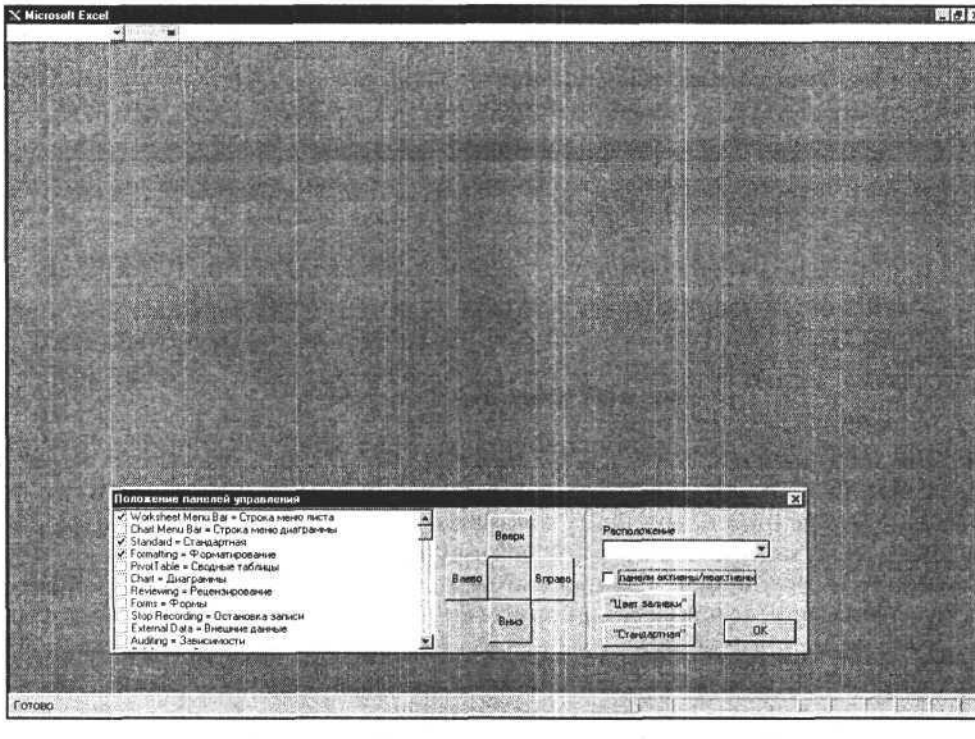


Рис. 16.9. Отключаем отображение панелей управления

Мы рассмотрели некоторые свойства встроенных панелей приложения MS Excel. Есть возможность программно создавать и удалять пользовательские панели и меню. Работа со встроенными панелями и меню имеет некоторые особенности, например их нельзя удалить.

Далее рассмотрим создание пользовательской панели (или меню).

Создание пользовательской панели (меню)

Для создания пользовательской (настраиваемой) панели используем метод Add коллекции CommandBars. У этого метода есть несколько аргументов. Синтаксис вызова метода Add:

```
Add(Name, Position, MenuBar, Temporary);
```

где Name:String — название создаваемой панели, Position:Integer — ее расположение (см. табл. 16.2), MenuBar:Boolean — признак создания меню, Temporary:Boolean — признак создания временной панели. Используя данные аргументы и их комбинации, мы можем создать как обычную панель с кнопками или всплывающее меню, так и заменить существующее главное меню приложения Excel. Любой из создаваемых компонентов коллекции CommandBars может быть создан как временный объект (на время одного сеанса работы приложения) или как постоянный объект. Рассмотрим создание пользовательской панели в приложениях Delphi на примере следующей процедуры.

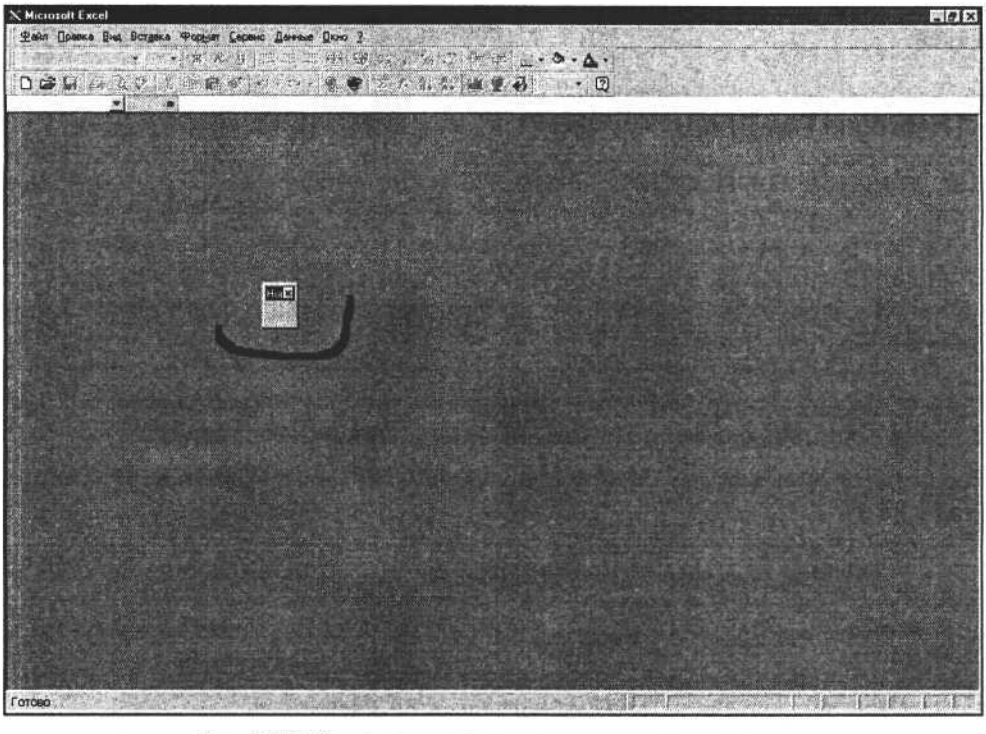


Рис. 16.10. Создана пользовательская панель управления

Создание пользовательской панели

```
procedure TForm1.Button2Click(Sender: TObject);
 const msoBarFloating=4;
 var  CommandBar_:variant;
begin
  CommandBar_:=W.CommandBars.Add('Пользовательская панель',
                          msoBarFloating, False, True);
```

```
  CommandBar_.Enabled:=True;
  CommandBar_.Visible:=True;
end;
```

По конкретным условиям, задаваемым аргументами в данной процедуре, создаваемая пользовательская панель отображается в центральной части главного окна. Результат выполнения процедуры представлен на рис. 16.10. Поскольку на новой панели нет ни одной кнопки, ее размер задан по умолчанию.

Мы исследовали общие свойства панели, присущие как панелям с кнопками, так и меню. Чтобы понять отличия, рассмотрим содержание самих панелей.

Элементы управления и их свойства

Как известно, панель может содержать кнопки или другие элементы управления, например раскрывающиеся списки или подменю. Эти элементы в свою очередь принадлежат самой панели и объединены в коллекцию CommandBarControls. Посредством этой коллекции осуществляется доступ ко всем элементам управления выбранной панели. Коллекция CommandBarControls имеет небольшое количество свойств и один метод. Мы будем использовать свойства Count (количество элементов коллекции) и Item (i:Integer) (набор элементов коллекции, где i — индекс кнопки), а также метод Add,позволяющий добавлять новые элементы на панель.

Рассмотрим некоторые свойства и методы объекта CommandBarControl (табл. 16.3).

Таблица 16.3. *Некоторые свойства и методы объекта CommandBarControl*

Свойство или метод	Тип	Описание
DescriptionText	String	Назначение элемента управления
Caption	String	Надпись на элементе управления
Execute	Метод	Выполнить команду, ассоциированную с элементом управления (кнопкой)
Enabled	Boolean	Доступный/недоступный элемент управления
Visible	Boolean	Видимый/невидимый элемент управления
HelpFile	String	Файл помощи для элемента управления
HelpContextID	Integer	Индекс в файле помощи для элемента управления
ID	Integer	Идентификатор элемента управления

Таблица 16.3 (окончание)

Свойство или метод	Тип	Описание
Index	Integer	Индекс элемента управления на панели
OnAction	String	Имя макроса для элемента управления
Reset	Метод	Сбросить пользовательские настройки для элемента управления
SetFocus	Метод	Активировать элемент управления
TooltipText	String	Всплывающая подсказка для элемента управления
Width	Extended	Ширина элемента управления
Left	Extended	Отступ от левой границы панели
Height	Extended	Высота элемента управления
Top	Extended	Отступ от верхней границы панели

Исследуем содержание панели и некоторые свойства элементов управления. Если рассматривать меню, то его основное отличие от панели с кнопками состоит в том, что элементом управления в составе меню может быть как пункт меню, так и подменю со своими элементами управления и т. д.

Чтобы проанализировать свойства панели, используем свойства Count и Item() коллекции CommandBarControls, а также свойства Caption и TooltipText. Загрузим весь список кнопок и других элементов панели в объект ListBox с помощью следующей процедуры.

Загрузка списка элементов управления панели

```
procedure TOKBottomDlg5.FormCreate(Sender: TObject);
 var a_:integer;
  eee_:string;
begin
  CommandBarControls:=OKBottomDlg4.CommandBars.Item[OKBottomDlg4.
ListBox1.ItemIndex+1].Controls;
  for a_:=1 to CommandBarControls.Count do
    begin
    eee_:=CommandBarControls.Item[a_].Caption+' = '+
          CommandBarControls.Item[a_].TooltipText;
    ListBox1.Items.Add(eee_);
    end;
  CommandBarControl:=CommandBarControls.Item[1];
end;
```

Затем, перемещаясь по списку элементов управления для выбранной панели, загруженному в компонент ListBox1, считаем некоторые их свойства.

Получение ссылки на элемент управления

```
procedure TOKBottomDlg5.ListBox1Click(Sender: TObject);
begin
  CommandBarControl:=CommandBarControls.Item[ListBox1.ItemIndex+1];
  b_visible.Checked:=CommandBarControl.Visible;
  DescriptionText.Text:=CommandBarControl.DescriptionText;
  HelpFile.Text:=CommandBarControl.HelpFile+' |'+
            inttostr(CommandBarControl.HelpContextID)+'|';
end;
```

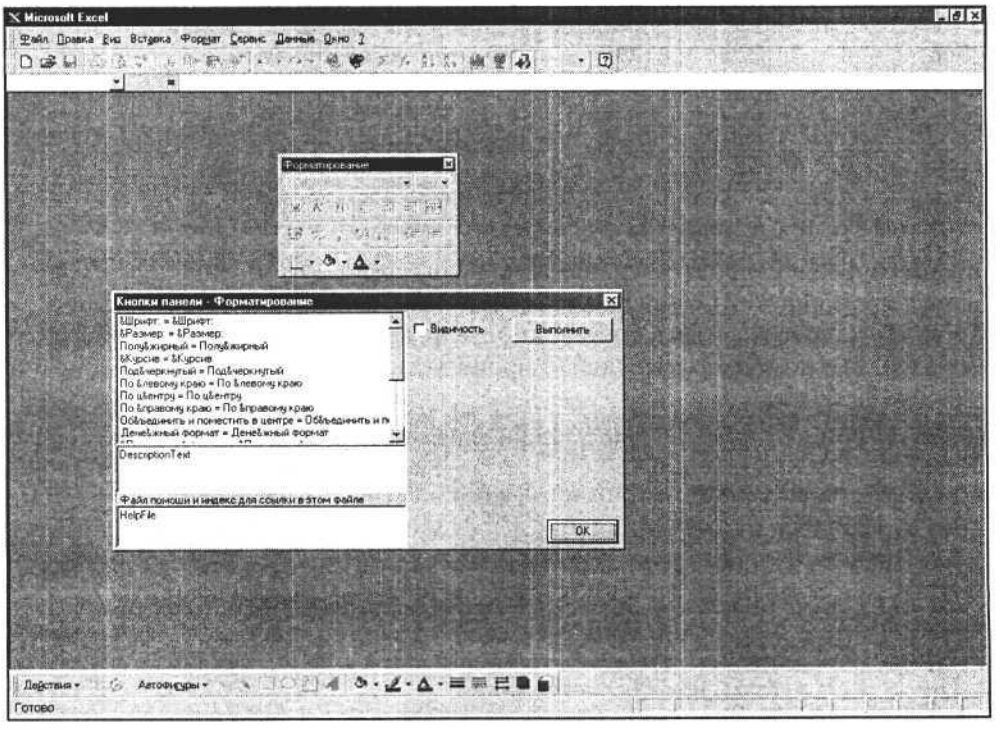


Рис. 16.11. Состав панели **Форматирование**

На рис. 16.11 показан результат выполнения представленных процедур. Верхний список (компонент ListBox1), расположенный в форме нашего приложения, заполнен названиями элементов управления панели **Форматирование**. При перемещении по этому списку в полях формы, расположен-

ных ниже списка, для выбранного элемента управления отображаются назначение и ссылка на файл помощи.

Как видно из рис. 16.10, переходя от одного элемента управления к другому, мы можем получать и изменять их свойства. Для этого достаточно загрузить в переменную типа Variant ссылку на элемент Item(i:Integer), например, как это выполнено в следующем операторе:

```
CommandBarControl:=CommandBarControls.Item[ListBox1.ItemIndex+1];
```

Далее, работая со ссылкой, мы получаем доступ к различным свойствам выбранного элемента управления. В качестве примера сделаем невидимой выбранную кнопку, изменим ее надпись (значение свойства Caption) и запустим на выполнение команду, связанную с данной кнопкой.

Управление свойствами кнопки

```
procedure TOKBottomDlg5.b_visibleClick(Sender: TObject);
begin
  CommandBarControl.Visible:=b_visible.Checked;
end;

procedure TOKBottomDlg5.B_CaptionChange(Sender: TObject);
begin
  CommandBarControl.Caption:=B_Caption.Text;
end;

procedure TOKBottomDlg5.Button1Click(Sender: TObject);
begin
  CommandBarControl.Execute;
end;
```

Используя последнюю процедуру, можно запускать макросы, связанные с кнопками. Для этого удобнее использовать пользовательские, временные панели и элементы управления, создание которых будет рассмотрено далее.

Главное меню

В отличие от панели инструментов, главное меню и его элементы имеют несколько иные свойства, основным отличием является возможность для каждого элемента меню содержать свою коллекцию элементов. Проще говоря, каждый элемент меню может содержать подменю, элементы которого также могут содержать свое подменю. Таким образом, меню имеет древовидную структуру. В этом случае доступ к любым элементам обеспечивается посредством элементов коллекций Controls, принадлежащих индивидуально

каждому элементу меню. На рис. 16.12 представлен пример объектной модели меню.

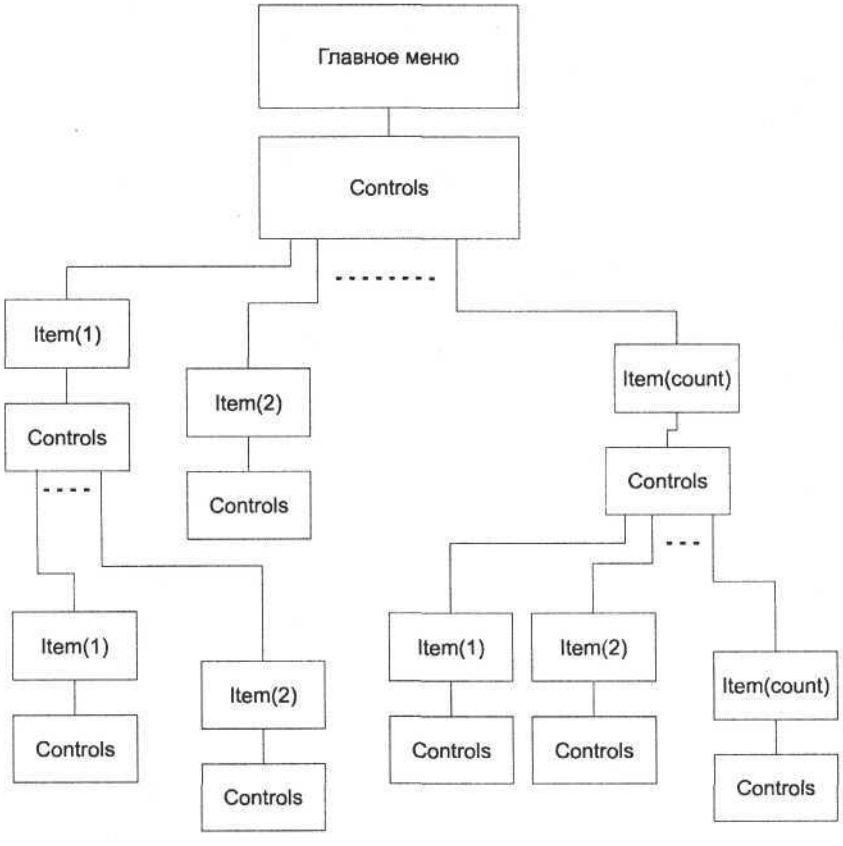


Рис. 16.12. Объектная модель главного меню

Используя коллекции Controls и методы элементов этих коллекций Execute и SetFocus, можно развернуть главное меню примерно так, как показано на рис. 16.13. Исходный текст этого модуля программы и форму для среды разработки приложений Delphi можно посмотреть в приложении на сопроводительном компакт-диске книги.

Из рис. 16.13 понятно, что мы раскрываем подменю (дочерний набор элементов меню), используя свойства и методы родительских объектов, которым принадлежат коллекции Controls (раскрываемое подменю). Метод Execute запускает команду, связанную с данным элементом меню, а если с этим элементом связано подменю, то данная команда раскрывает его. Метод SetFocus устанавливает фокус на данный элемент меню.

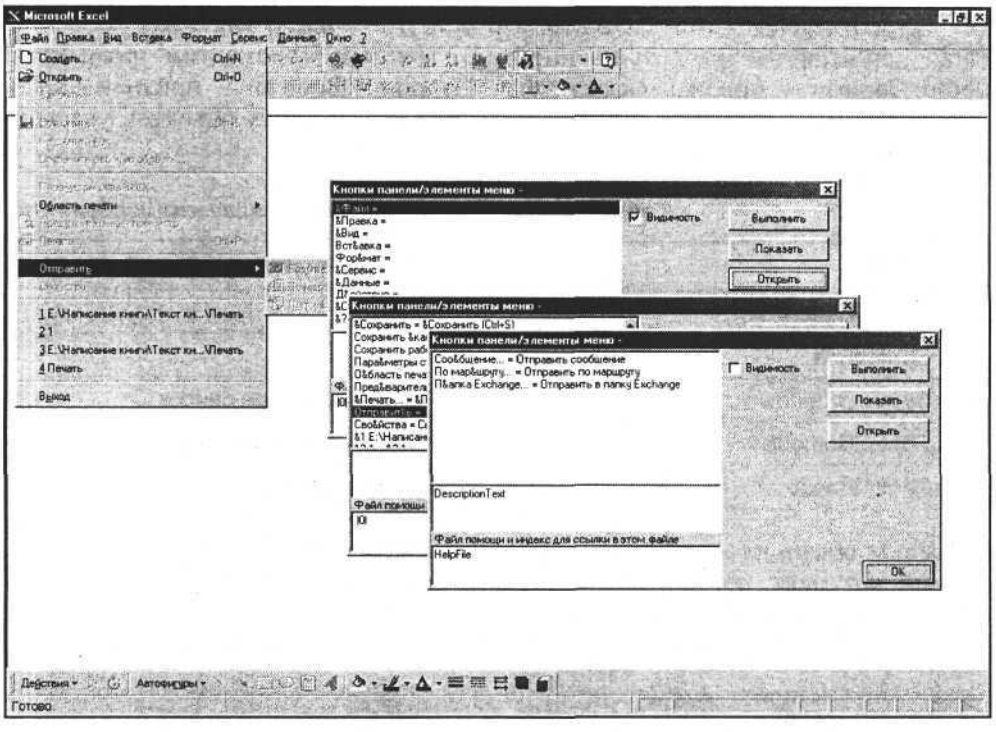


Рис. 16.13. Анализ объекта "главное меню"

От исследования существующих элементов меню и элементов управления на обычных панелях перейдем к созданию собственных элементов управления.

Создание пользовательского элемента управления

Все элементы управления для любой панели собраны в коллекцию Controls. Данная коллекция не только объединяет элементы и их свойства, но и обладает методами, позволяющими создавать или удалять элементы управления. Рассмотрим два метода коллекции. Метод Add создает новый элемент, а метод Delete — удаляет выбранный элемент. Причем метод Delete принадлежит элементу коллекции, а не самой коллекции, как метод Add, и его нельзя применить к стандартным элементам коллекции, а только к вновь созданным.

Синтаксис вызова метода Add:

```
Add(Type, Id, Parameter, Before, Temporary);
```

где Type:Integer — тип объекта, Id:Integer — идентификатор объекта, Parameter:String — имя связанной с данным элементом команды, Before:Boolean — признак разделителя, Temporary:Boolean — признак создания временного элемента. Возможные типы создаваемых объектов указаны в табл. 16.4.

Таблица 16.4. *Возможные типы создаваемых объектов*

Константа	Значение	Тип создаваемого объекта
msoControlButton	1	Кнопка
msoControlEdit	2	Поле ввода
msoControlDropdown	3	Раскрывающийся список
msoControlComboBox	4	Комбинированный список
msoControlPopup	10	Элемент меню

Создадим обычную кнопку на панели **Стандартная**. Используем метод Add, первый аргумент которого (Type) равен msoControlButton, второй (ID) — единице. Установим значение надписи, записав в Caption текст, выберем значок (свойство FaceId) и стиль кнопки (отображение и значка, и текста).

Создание кнопки

```
Var msButton:variant;
procedure TOKBottomDlg8.Button2Click(Sender: TObject);
begin
  msButton:=CommandBars.Item[CheckListBox1.ItemIndex+1].
Controls.Add(Type:=msoControlButton, ID:=1);
  msButton.Caption:='Простая кнопка';
end;

procedure TOKBottomDlg8.FaceIdChange(Sender: TObject);
begin
  msButton.FaceId:=FaceId.Value;
end;

procedure TOKBottomDlg8.ButtonStyleChange(Sender: TObject);
begin
  msButton.Style:=ButtonStyle.ItemIndex;
end;
```

Результат выполнения представленных процедур показан на рис. 16.14. На панели **Стандартная** создана кнопка, задан текст, выбран значок и установлен стиль, в соответствии с которым отображаются текст и значок.

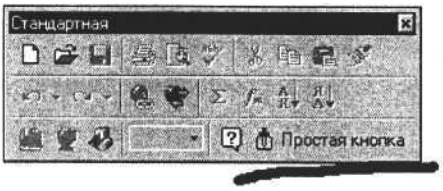


Рис. 16.14. Создаем новую кнопку на панели **Стандартная**

Изменим стиль кнопки, записав в свойство msButton.Style константу msoButtonIcon или msoButtonCaption. Соответственно получаем результат, представленный на рис. 16.15 и 16.16.

Выбор стиля кнопки "только значок"

```
procedure TOKBottomDlg8.ButtonStyleChange(Sender: TObject);
begin
  msButton.Style:= msoButtonIcon;
end;
```

Рис. 16.15. На кнопке только значок

Рис. 16.16. На кнопке только текст

Константа msoButtonIcon представляет собой число и имеет значение 1. Записав в свойство Style кнопки константу msoButtonCaption (значение 2), получим кнопку, на которой расположен только текст.

Выбор стиля кнопки "только текст"

```
procedure TOKBottomDlg8.ButtonStyleChange(Sender: TObject);
begin
msButton.Style:= msoButtonCaption;
end;
```

При создании обычного пользовательского элемента (кнопки) мы присваивали параметру ID значение, равное единице. Присвоив данному параметру другое значение, получим функциональную кнопку со своим значком и командой, выполняемой при обращении к данной кнопке. Для примера можем использовать значения от 2 до 9 или от 1850 до 1859.

Создание кнопок с предопределенными свойствами

```
procedure TOKBottomDlg8.Button1Click(Sender: TObject);
 var msbootom:variant;
   a_:integer;
begin
  for a_:=1850 to 1859 do begin
  msbootom:=CommandBars.Item[CheckListBox1.ItemIndex+1].Controls.
Add(Type:=msoControlButton, ID:=inttostr(a_));
  msbootom.Caption:='ID='+inttostr(a_);
  end;
end;
```

Значение значка для кнопки определяется величиной числовой константы, записанной в свойство FaceId элемента управления. В качестве примера создадим кнопки со значками для значений этого свойства от 0 до 500.

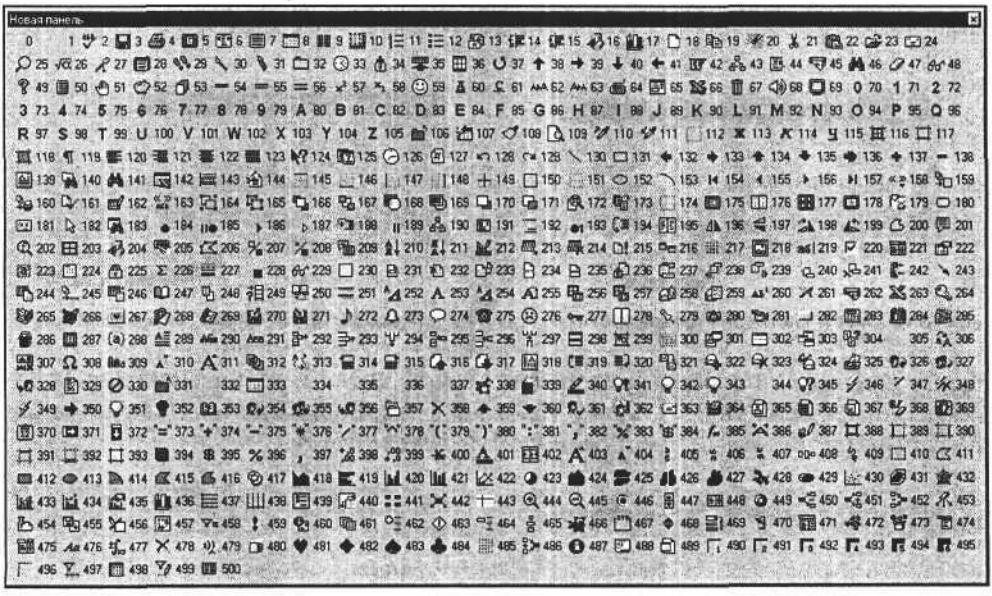


Рис. 16.17. Несколько вариантов кнопок

Создание кнопок с разными значками

```
procedure TForm1.Button13Click(Sender: TObject);
 var panel, buttom:variant;
    a_:integer;
```

```
begin
  panel:=W.CommandBars.Add('Новая панель');
  panel.Visible:=True;
  for a_:= 0 to 500 do begin
    buttom:=panel.Controls.Add(Type:=msoControlButton, ID:=1);
    buttom.Style:=msoButtonIconAndCaption;
    buttom.Caption:=inttostr(a_);
    buttom.FaceId:=a_;
  end;
end;
```

Вид кнопок созданной панели показан на рис. 16.17.

Использование Visual Basic Editor

До этого мы рассматривали доступ к элементам рабочей книги и элементам управления из внешних приложений Delphi. Для полного представления о возможностях управления приложением Excel из внешних программ этого недостаточно. Объектная модель Excel позволяет управлять не только элементами создаваемых документов, но и элементами программ на встроенном языке. Для доступа к элементам программ используется объект VBE. Родительским объектом для VBE является Application (Application.VBE), а свойства и методы самого объекта VBE позволяют полностью управлять приложениями Visual Basic из внешних программ. На рис. 16.18 представлена часть общей объектной модели VBE, позволяющая понять, каким образом можно получить доступ непосредственно к текстам встроенных макросов.

Объект VBE включает в себя коллекции проектов, областей программ, элементов управления и коллекцию окон редактора. Очевидно, что, используя перечисленные ссылки, можно получить доступ к элементам этих коллекций и к их свойствам. Кроме этих коллекций объект VBE включает ссылки на активные элементы этих коллекций и собственные свойства. Собственным свойством является строка, содержащая номер версии редактора. В конечном итоге нам в целях разработки эффективных приложений достаточно будет получить доступ к текстам макросов. Для того чтобы создать новые или изменить уже существующие программные модули VB, используем свойства VBE, представленные в табл. 16.5.

***Таблица 16.5.** Свойства объекта VBE*

Свойство	Тип	Назначение
Version	Строка	Версия
ActiveCodePane	Объект	Ссылка на активный модуль

Таблица 16.5 *(окончание)*

Свойство	Тип	Назначение
ActiveVBProject	Объект	Ссылка на активный проект
ActiveWindow	Объект	Ссылка на активное окно
CodePanes	Объект	Ссылка на коллекцию модулей
CommandBars	Объект	Ссылка на коллекцию элементов управления
Events.CommandBarEvents	Объект	Ссылка на объект-источник события при воздействии на элемент управления
Events.ReferencesEvents	Объект	Ссылка на объект-источник события при добавлении или удалении ссылок
MainWindow	Объект	Ссылка на главное окно
VBE.SelectedVBComponent	Объект	Ссылка на выделенный элемент
VBProjects	Объект	Ссылка на коллекцию проектов
Windows	Объект	Ссылка на коллекцию окон

Когда по условиям логики работы вашего приложения требуется знать версию Visual Basic Editor, можно воспользоваться свойством Version объекта VBE и получить номер версии редактора.

Получение номера версии VBE

```
procedure TForm1.Button17Click(Sender: TObject);
 var Version:string;
begin
  Version:=E.VBE.Version;
  messagebox(handle,pchar(Version),'Номер версии Visual Basic',0);
end;
```

Для программирования приложений знать только номер версии часто бывает недостаточно, поэтому обратим внимание на коллекции VBProjects и CodePanes. Первая представляет собой список, как правило, совпадающий со списком открытых рабочих книг. Каждая рабочая книга может содержать программные модули, модули классов, формы и модули документов. Совокупность всех или части перечисленных элементов представляет собой *проект*. Далее мы будем рассматривать только программные модули, которые являются элементами коллекции VBComponents и могут быть добавлены или удалены с помощью методов этой коллекции. Коллекция CodePanes позволяет только обеспечить доступ к текстам программных модулей. На-

пример, доступ к программному модулю первого элемента CodePanes осуществляется с помощью следующего оператора:

```
CodeModule :=E.VBE.CodePanes.Item(1).CodeModule;
```

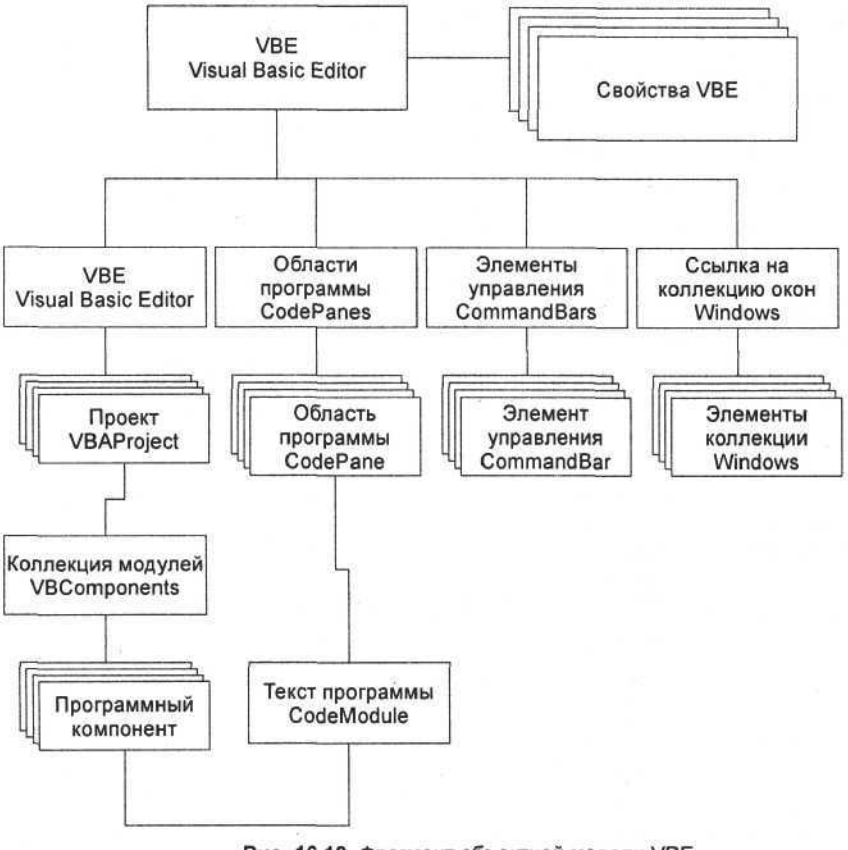


Рис. 16.18. Фрагмент объектной модели VBE

Программный модуль непосредственно содержит тексты макросов. Методы и свойства этого объекта (CodeModule) мы рассмотрим позже. Сейчас вернемся к рассмотрению свойств элемента коллекции VBProjects, доступ к которому обеспечивает метод Item(i:Integer). Его свойства перечислены в табл. 16.6.

Таблица 16.6. *Свойства элемента коллекции VBProjects*

Свойство	Тип	Назначение
References	Объект	Ссылки на внешние объекты
Collection	Объект	Ссылка на родительский объект VBProjects

Таблица 16.6 (окончание)

Свойство	Тип	Назначение
Description	String	Текст, связанный с объектом
HelpContextID	Integer	Ссылка в файле помощи
HelpFile	String	Имя файла помощи
Mode	Integer	Режим, в котором находится проект
Name	String	Имя объекта
Protection	Integer	Признак защиты проекта
Saved	Boolean	True — сохранен, False — не сохранен
VBE	Объект	Ссылка на корневой объект VBE
VBComponents	Объект	Ссылка на коллекцию компонентов

В табл. 16.6 перечислены свойства любого проекта, созданного в среде рабочей книги Excel. Используем только коллекцию VBComponents, которая представляет собой набор элементов, входящих в проект (табл. 16.7).

Таблица 16.7. Свойства и методы коллекции VBComponents

Свойство или метод	Тип	Назначение
Add	Метод	Добавление нового компонента
Count	Integer	Количество элементов коллекции
Import	Метод	Импорт модуля из файла
Remove	Метод	Удаление компонента
Item	Метод	Элемент коллекции

Исследуем некоторые свойства и методы этой коллекции. Свойство Count представляет собой количество элементов коллекции, точнее, суммарное количество модулей классов, форм, стандартных модулей и модулей документа. Чтобы добавить новый элемент коллекции VBComponents, используем метод Add. Единственным аргументом этого метода является числовая константа, определяющая тип создаваемого элемента. Например, создадим форму Microsoft, для этого в качестве аргумента метода Add используем константу vbext_ct_MSForm=3.

Создание формы Microsoft

```
procedure TForm1.Button22Click(Sender: TObject);
begin
  E.VBE.VBProjects.Item(1).VBComponents.Add (vbext_ct_MSForm)
end;
```

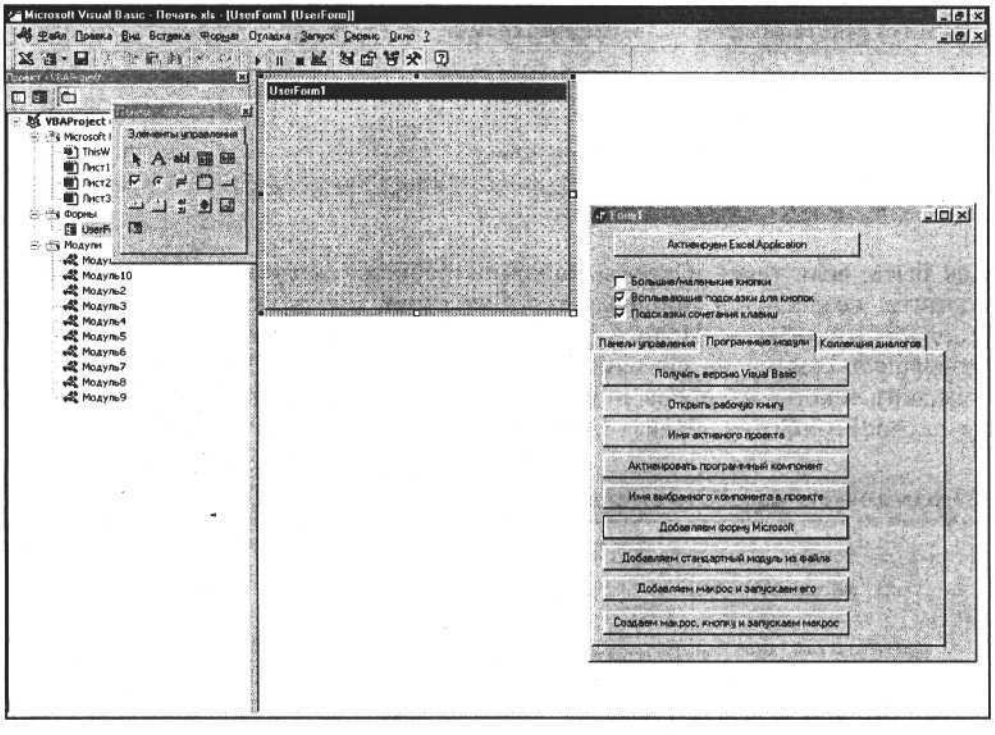


Рис. 16.19. В приложении VB создана новая форма Microsoft

На рис. 16.19 представлен результат выполнения данной процедуры.

Для добавления других типов модулей используем метод Add с другой константой в качестве аргумента. На практике в приложениях Delphi часто требуется создавать стандартные модули, содержащие тексты макросов. С чем это связано? В предыдущих главах для работы с книгами Excel мы использовали методы, аргументы которых имеют совместимые типы данных для Excel и Delphi. При этом не возникало проблем при экспорте и импорте информации между приложением Delphi и рабочей книгой Excel. Совместимыми типами данных являются целые числа и числа с плавающей точкой, строка и дата. Данные, представляющие собой, например, массив точек, не могут передаваться из Delphi в Excel. В таких случаях мы используем возможность создания макроса средствами Delphi и передачи ему команды на выполнение. При этом макросы могут храниться в виде текстовых файлов и загружаться в проект VBE непосредственно перед выполнением и удаляться после окончания выполнения. Рассмотрим процедуру, которая демонстрирует эту возможность.

Загрузка текста модуля из текстового файла

```
procedure TForm1.Button23Click(Sender: TObject);
begin
  if not OpenDialog2.Execute then exit;
  E.VBE.VBProjects.Item(1).VBComponents.Import(OpenDialog2.FileName);
end;
```

Как быть, если текст макроса, который требуется загрузить в проект и выполнить, создается в приложении? Для этого мы должны создать новый стандартный модуль. Используем метод Add(vbext_ct_StdModule), который возвращает ссылку на созданный объект CodeModule. Объект CodeModule содержит тексты макросов и методы работы с этими текстами. Используя метод AddFromString объекта CodeModule, загружаем текст макроса.

Загрузка текста макроса из строки

```
procedure TForm1.Button24Click(Sender: TObject);
 var text_macro:String;
begin
  CodeModule:=E.VBE.VBProjects.Item(1).VBComponents.
Add(vbext_ct_StdModule).CodeModule;
  text_macro:='Sub Новый_макрос()'+Chr(10)+Chr(13)+
        'MsgBox "Новый макрос"'+Chr(10)+Chr(13)+'End Sub ';
  CodeModule.AddFromString(text_macro);
  E.Run('Новый_макрос()');
end;
```

В результате выполнения данной процедуры будет создан, а затем и выполнен новый макрос. В представленном примере мы рассмотрели и использовали только одно свойство элемента коллекции VBComponents — объект CodeModule. Сам объект CodeModule содержит тексты макросов и набор методов и свойств, позволяющих манипулировать ими (табл. 16.8).

Таблица 16.8. *Некоторые свойства и методы объекта CodeModule*

Свойство или метод	Тип	Назначение
AddFromFile	Метод	Загрузка текста из файла
AddFromString	Метод	Загрузка текста из строки
CountOfLines	Integer	Количество строк текста в программном модуле
CountOfDeclarationLines	Integer	Количество строк текста в разделе описаний
InsertLines	Метод	Вставка текста из строки

Таблица 16.8 (окончание)

Свойство или метод	Тип	Назначение
DeleteLines	Метод	Удаление текста из модуля
Lines	Метод	Возвращение текста модуля
ProcBodyLine	Integer	Номер первой строки процедуры
ProcCountLines	Integer	Количество строк в процедуре
ProcOfLine	String	Имя процедуры, содержащей строку с заданным номером
ProcStartLine	Integer	Номер строки, с которой начинается заданная процедура

Используя свойство Lines, получим текст модуля проекта на языке Visual Basic. Для этого воспользуемся следующей процедурой.

Получение текста модуля

```
var CodeModule:variant;
procedure TForm1.Button21Click(Sender: TObject);
 var Text_modula:string;
begin
  CodeModule:=E.VBE.VBProjects.Item(1).VBComponents.Item(1).CodeModule;
  Text_modula:=CodeModule.Lines[1, CodeModule.CountOfLines];
  messagebox(handle,pchar(Text_modula),'',0);
end;
```

Перед тем как воспользоваться представленной процедурой, необходимо открыть рабочую книгу, которая должна содержать макросы. При этом результат выполнения процедуры будет выглядеть так, как показано на рис. 16.20.

В заключение рассмотрим пример с использованием программирования панелей, элементов управления и макросов. В этом примере мы из приложения Delphi создадим временные элементы управления и процедуру на Visual Basic, выполним ее и удалим созданные компоненты.

Создание и использование макроса из приложений Delphi

```
procedure TForm1.Button5Click(Sender: TObject);
 const vbext_ct_StdModule=1;
 var  CodeModule:variant;
      CommandBar:variant;
        msbootom:variant;
            eee_:string;
```

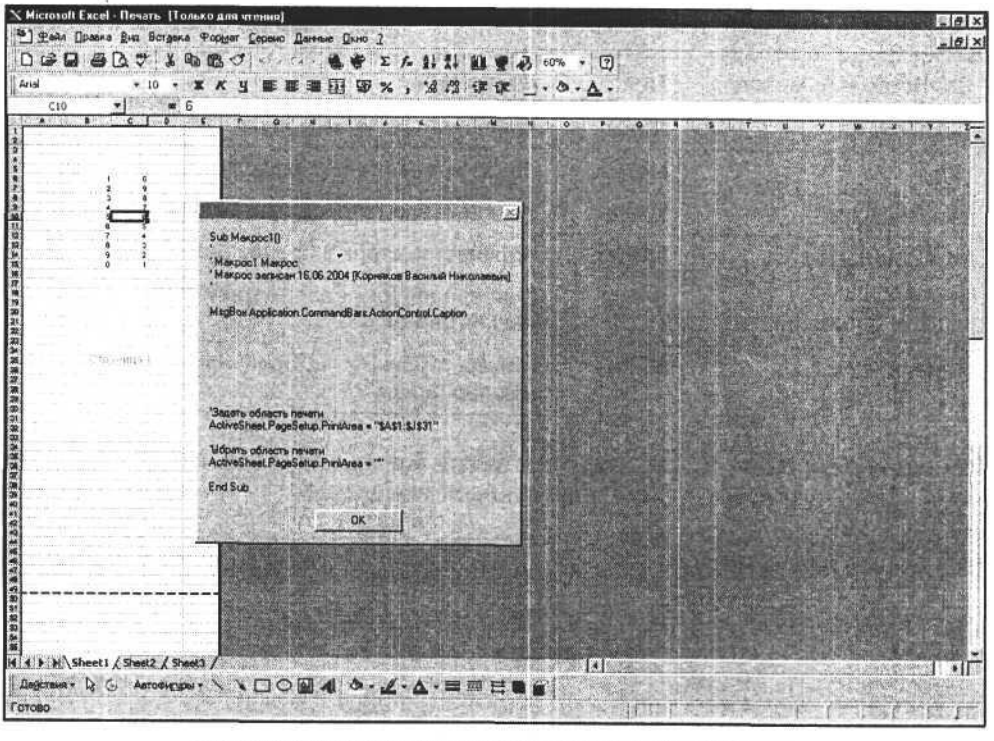


Рис. 16.20. Отображаем содержимое программного модуля

```
begin
  CodeModule:=E.VBE.VBProjects.Item(1).VBComponents.
Add(vbext_ct_StdModule).CodeModule;
  eee_:='Sub Макрос1()'+Chr(10)+Chr(13)+
    'MsgBox '+Chr(34)+'Выполняется процедура нажатия кнопки'+
     Chr(34)+Chr(10)+Chr(13)+
    'set document=Application.WorkBooks.Add '+Chr(10)+Chr(13)+
    'Dim pts(1 To 7, 1 To 2) As Single '+Chr(10)+Chr(13)+
    'pts(1, 1) = 0 '+Chr(10)+Chr(13)+
    'pts(1, 2) = 0 '+Chr(10)+Chr(13)+
    'pts(2, 1) = 72 '+Chr(10)+Chr(13)+
    'pts(2, 2) = 72 '+Chr(10)+Chr(13)+
    'pts(3, 1) = 100 '+Chr(10)+Chr(13)+
    'pts(3, 2) = 40 '+Chr(10)+Chr(13)+
    'pts(4, 1) = 20 '+Chr(10)+Chr(13)+
    'pts(4, 2) = 50 '+Chr(10)+Chr(13)+
    'pts(5, 1) = 90 '+Chr(10)+Chr(13)+
    'pts(5, 2) = 120 '+Chr(10)+Chr(13)+
    'pts(6, 1) = 60 '+Chr(10)+Chr(13)+
```

```
    'pts(6, 2) = 30 '+Chr(10)+Chr(13)+
    'pts(7, 1) = 150 '+Chr(10)+Chr(13)+
    'pts(7, 2) = 90 '+Chr(10)+Chr(13)+
    'ActiveSheet.Shapes.AddPolyline (pts) '+Chr(10)+Chr(13)+
    'End Sub';
  CodeModule.AddFromString(eee_);
  eee_:=CodeModule.name;
  messagebox(handle,pchar('Макрос "Макрос1" создан в модуле "'+eee_+'"'),
    '',0);
  CommandBar:=E.CommandBars.Add(Name:='Временная панель',
    Position:=msoBarFloating);
  CommandBar.Enabled:=true;
  CommandBar.Visible:=true;
  messagebox(handle,pchar('Создана временная панель'),'',0);
  MSBootom:=E.CommandBars.Item['Временная панель'].Controls.
     Add(Type:=msoControlButton, ID:=1);
  MSBootom.Caption:='Временная кнопка';
  MSBootom.OnAction:=CodeModule.name+'.Макрос1';
  messagebox(handle,pchar('Создана временная кнопка'),'',0);
  msbootom.Execute;
  MSBootom.Delete;
  messagebox(handle,pchar('Удалена временная кнопка'),'',0);
  CommandBar.Delete;
  messagebox(handle,pchar('Удалена временная панель'),'',0);
  E.Application.VBE.VBProjects.Item(1).VBComponents.
    Remove(CodeModule.parent);
  messagebox(handle,pchar('Модуле "'+eee_+'" и "Макрос1" удалены'),'',0);
end;
```

В первом операторе данного примера мы создаем программный модуль и получаем доступ к нему, чтобы записать текст создаваемого макроса. Далее в строковую переменную записываем текст создаваемого макроса, учитывая стиль написания операторов Visual Basic. Оператор

```
CodeModule.AddFromString(eee_);
```

где eee_ — строка, содержащая текст макроса, записывает текст макроса в модуль. Далее создаем меню и кнопку, в свойство OnAction которой записываем строку — имя модуля и макроса, разделенные точкой. Метод Execute кнопки запускает созданный макрос на выполнение. Далее удаляем кнопку, панель и компонент, содержащий программный модуль.

Если эту процедуру выполнять по шагам, то сможем наблюдать следующее. На первом этапе мы видим в окне редактора Visual Basic, как создается программный модуль и в него записывается текст макроса (рис. 16.21).

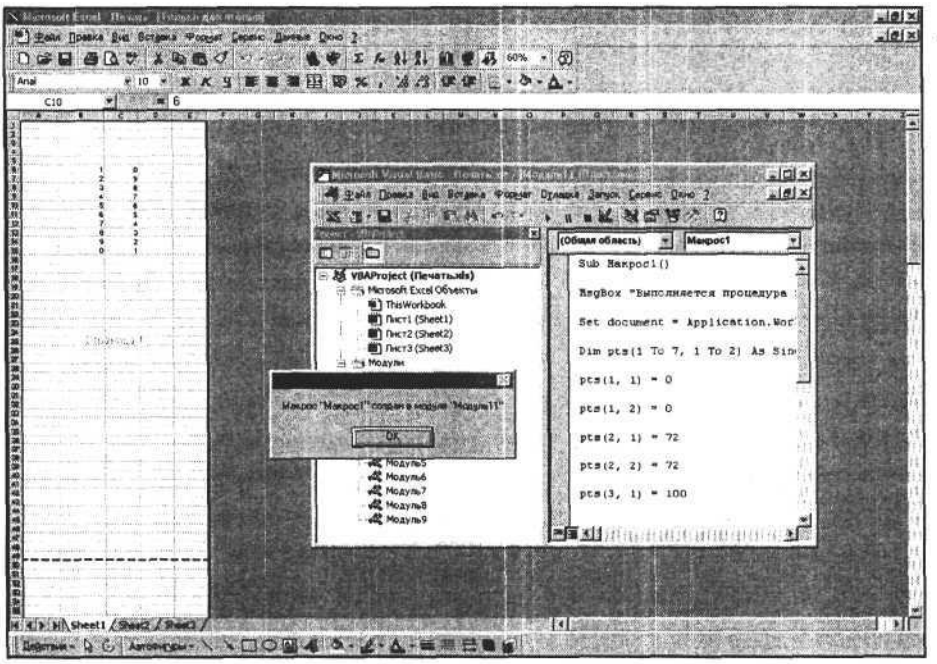


Рис. 16.21. Создан модуль и записан текст макроса

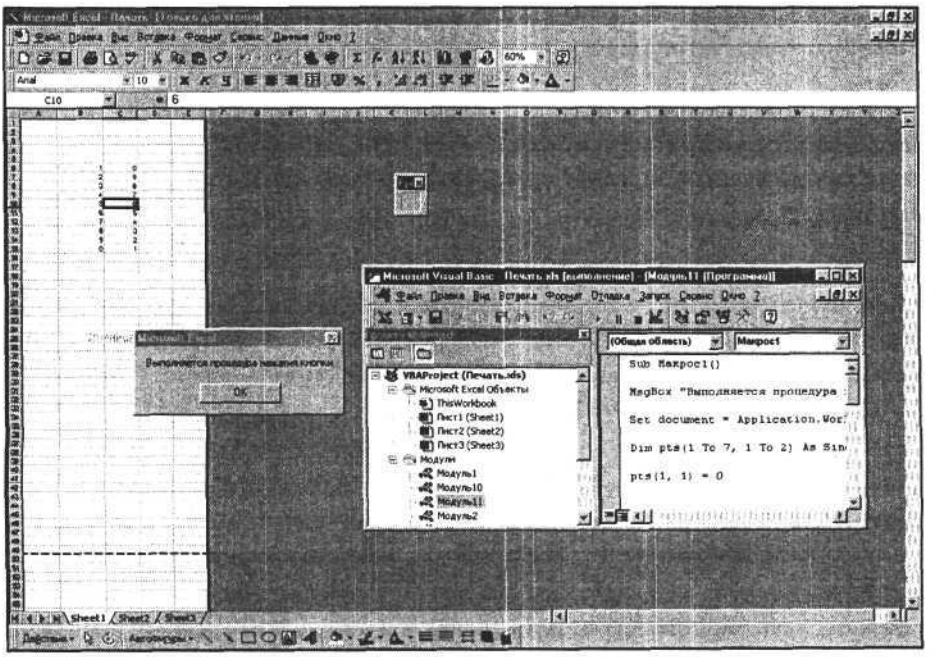


Рис. 16.22. Макрос запущен на выполнение

Далее выполняются метод Execute кнопки, с которой связан макрос, и программа на Visual Basic, записанная в программном модуле (рис. 16.22).

После выполнения удаляются созданные элементы управления и программный модуль с текстом подпрограммы на Visual Basic (рис. 16.23).

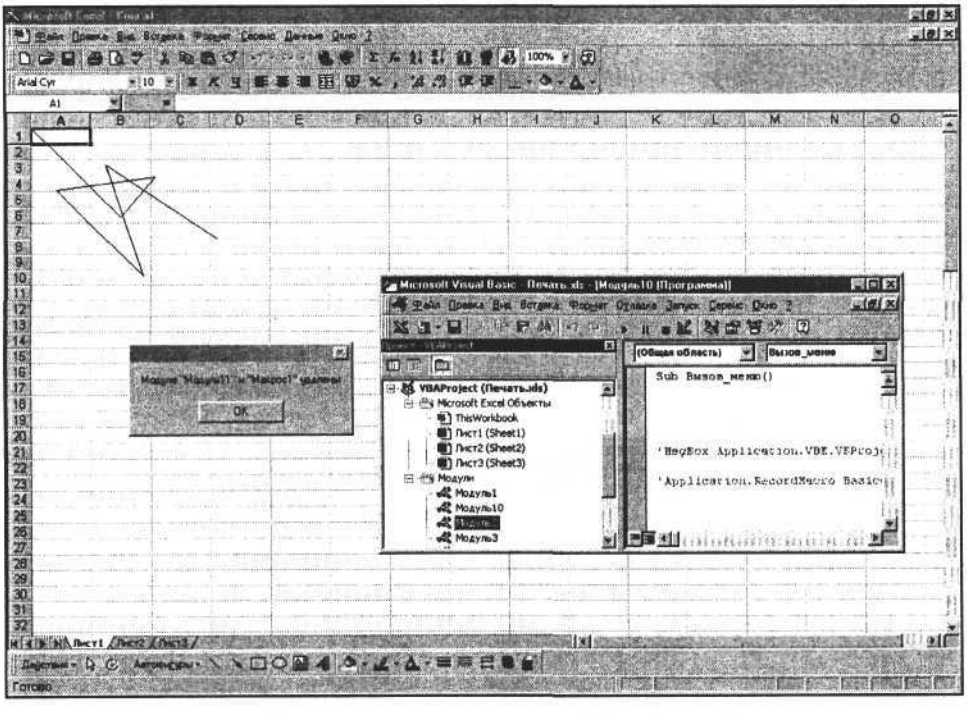


Рис. 16.23. Макрос выполнен, элементы управления и программный модуль удалены

Коллекция диалогов

Рассмотрим еще один тип элементов управления приложения Excel — элементы коллекции Dialogs. Коллекция Dialogs имеет небольшой список свойств, два из которых это количество элементов коллекции (свойство Count) и ссылка на элементы коллекции (свойство Item(i:Integer), где i — индекс диалога в коллекции). Чтобы ознакомиться со всем списком и значениями индексов и аргументов диалогов, достаточно открыть справочную систему MS Excel на нужной странице. Здесь мы рассмотрим только общие принципы вызова диалогов из приложений, разработанных в среде Delphi, и несколько примеров их использования.

Каждый элемент диалога имеет метод и свойства. Свойства элементов коллекции диалогов можно не рассматривать. Для каждого диалога мы распо-

лагаем одним методом — Show. Его можно вызывать с аргументами, которые определяются типом самого диалога. Метод Show возвращает значение True, если диалог завершен выбором (например, нажатием кнопки **OK**), или False, если пользователь отменил выбор и диалог закрыт. Следующая процедура позволяет получить количество различных диалогов, используемых в Excel.

Определение количества диалогов в Excel

```
procedure TForm1.Button7Click(Sender: TObject);
 var count_:integer;
begin
  count_:=E.Dialogs.count;
  messagebox(handle,pchar(inttostr(count_)),
          'Количество элементов коллекции диалогов',0);
end;
```

На рис. 16.24 представлен результат выполнения данной процедуры.

Рис. 16.24. Количество диалогов в Excel составляет несколько сотен

Рассмотрим вызов метода Show элемента коллекции диалогов. Этот метод можно вызывать как без аргументов, так и с аргументами. Если для конкретного диалога при вызове заданы аргументы, то обычно они служат для настройки некоторых его полей и свойств.

Примечание

Список аргументов можно получить в справочной системе MS Excel.

При вызове метода обычно не используются реальные имена аргументов, указанные в справочной системе. Наименование аргумента представляет собой сочетание символьной строки arg и номера аргумента, например:

```
arg1:=2;
```

В табл. 16.9 представлены списки аргументов для двух диалогов. Этот список является частью списка справочной системы.

Воспользуемся этой информацией, чтобы при вызове диалога открытия документа подставить в поле **Имя файла** нужное нам значение.

Таблица 16.9. *Аргументы метода Show для диалогов*

Константа для вызова диалога	Список аргументов
xlDialogOpen	file_text, update_links, read_only, format, prot_pwd, write_res_pwd, ignore_rorec, file_origin, custom_delimit, add_logical, editable, file_access, notify_logical, converter
xlDialogSaveAs	document_text, type_num, prot_pwd, backup, write_res_pwd, read_only_rec

Задание имени файла в диалоге открытия документа

```
procedure TForm1.Button11Click(Sender: TObject);
begin
  E.Dialogs.Item[xlDialogOpen].Show(arg1:='Моя рабочая книга');
end;
```

Результат выполнения данного диалога представлен на рис. 16.25.

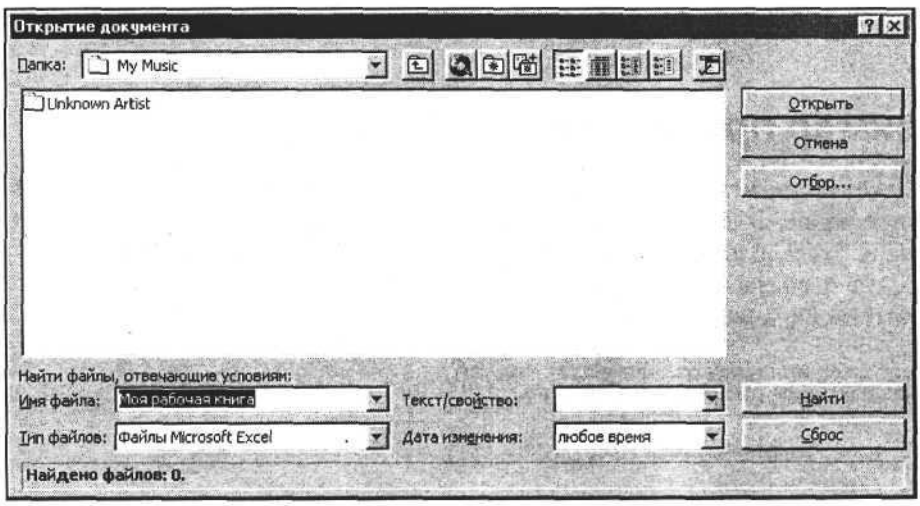


Рис. 16.25. Подставляем имя файла в диалог открытия документа

Далее вызовем диалог сохранения документа в файле, при этом зададим имя файла непосредственно во время вызова диалога.

Задание имени файла в диалоге сохранения рабочей книги

```
procedure TForm1.Button15Click(Sender: TObject);
begin
  E.Dialogs.Item[xlDialogSaveAs].
Show(arg1:='Подставляем собственное имя файла');
end;
```

Вид диалога показан на рис. 16.26.

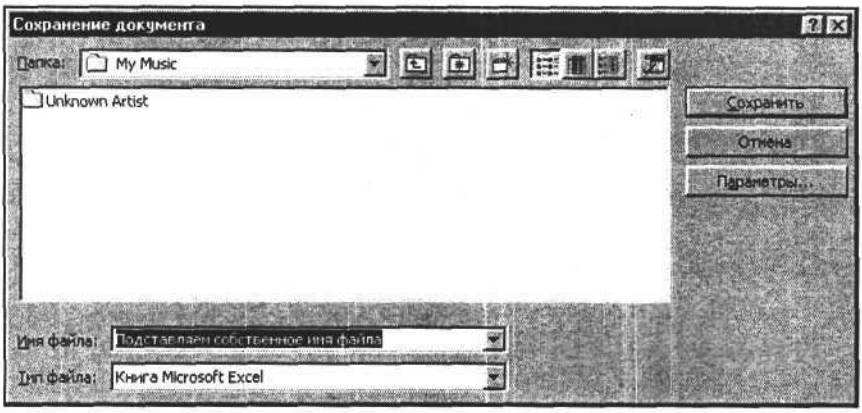


Рис. 16.26. Подставляем имя файла в диалог сохранения документа

Пример программирования панели

В качестве примера возьмем формирование налоговой декларации, рассмотренное ранее. В данном примере дополнительно отключаем все стандартные панели Excel, блокируем доступ пользователя к изменению содержания документа и создаем пользовательскую панель, на которую поместим кнопки печати и сохранения документа.

Процедура формирования документа "Налоговая декларация"

```
function FindAndReplace(find_,rep_:string):boolean;
 var range:variant;

begin
  FindAndReplace:=False;
  if find_<>'' then begin
    try
```

```
range:=Form1.E.Range['A1:EL230'].Replace(What:=find_,Replacement:=rep_);
    FindAndReplace:=True;
    except
    FindAndReplace:=False;
  end;
end;
end;

procedure TForm1.Button14Click(Sender: TObject);
    var a_:integer;
 CommandBar_:variant;
  msbootom_:variant;
begin
  for a_:=1 to E.CommandBars.Count do
    E.CommandBars.Item[a_].Enabled:=False;
    // Создаем новый документ по шаблону
    E.WorkBooks.Add(ExtractFileDir(Application.ExeName)+
                                   '\Декларация НДС.xls');
    messagebox(handle,'Шаблон создан! Переходим к заполнению.',
               'Внимание!',0);
    // Подставляем текст
    FindAndReplace('#ИНСПЕКЦИЯ&','Инспекция №1');
    FindAndReplace('#ОРГАНИЗАЦИЯ&','ЗАО "Сокол"');
    for a_:=1 to 12 do FindAndReplace('И'+inttostr(a_)+'&',INN.Text[a_]);
    for a_:=1 to 9 do FindAndReplace('К'+inttostr(a_)+'&',KPP.Text[a_]);
    for a_:=1 to 15 do FindAndReplace('С'+inttostr(a_)+'&',
                                   (SUMMA.Text+'         ')[a_]);
    for a_:=1 to 2 do FindAndReplace('Д'+inttostr(a_)+'&',DATA.Text[a_]);
    for a_:=1 to 2 do
                  FindAndReplace('М'+inttostr(a_)+'&',DATA.Text[a_+3]);
    for a_:=1 to 4 do
                  FindAndReplace('Г'+inttostr(a_)+'&',DATA.Text[a_+6]);
    E.ActiveSheet.Protect( DrawingObjects:=True, Contents:=True,
                          Scenarios:=True, Password:='');
    CommandBar_:=E.CommandBars.Add('Печать документа',msoBarTop, False,
                                   True);
    CommandBar_.Enabled:=True;
    CommandBar_.Visible:=True;
    CommandBar_.Protection:=msoBarNoMove;
    for a_:=3 to 4 do begin
      try
      msbootom_:=CommandBar_.Controls.Add(Type:=msoControlButton,
                                          ID:=inttostr(a_));
```

```
      msbootom_.Caption:='ID='+inttostr(a_);
      except
      end;
    end;
  Messagebox(handle,'Возвращаем панели в исходное состояние!',
             'Внимание!',0);
  for a_:=1 to E.CommandBars.Count do
           E.CommandBars.Item[a_].Enabled:=True;
end;
```

Результат выполнения процедуры представлен на рис. 16.27.

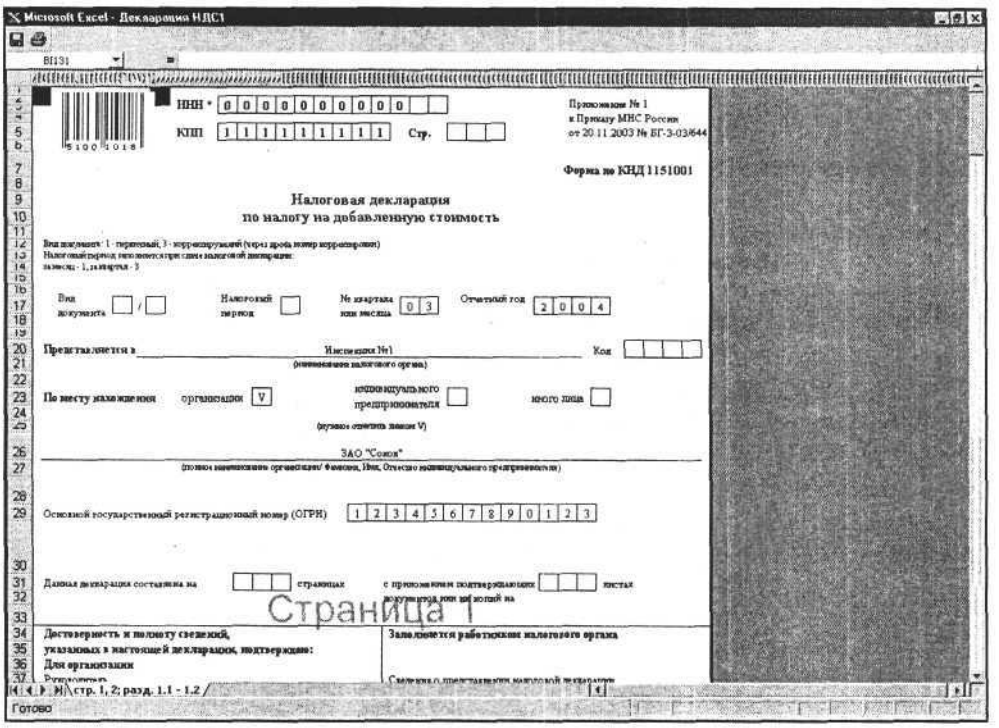
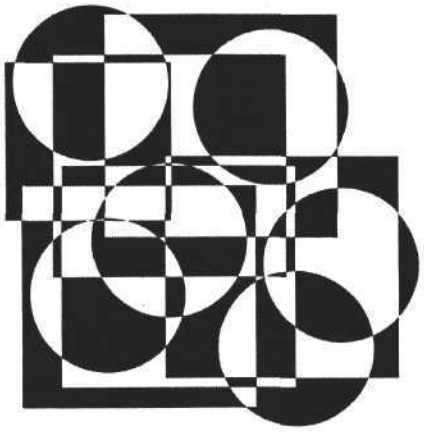


Рис. 16.27. Пользовательская настройка приложения Excel для печати документа

В предыдущих главах мы рассмотрели, как объект Excel.Application используется для создания документов и управления ими, далее рассмотрим создание и использование динамических библиотек для приложений, формирующих документы в приложениях Word и Excel, а также для их применения в проектах Visual Basic.



ЧАСТЬ IV

Разработка в Delphi и использование динамических библиотек для работы с MS Office

ГЛАВА 17

Создание пользовательской библиотеки DLL

В этой главе рассмотрены следующие темы:

- создание пользовательской библиотеки;
- создание пользовательской динамической библиотеки;
- неявная загрузка модуля DLL;
- явная загрузка модуля DLL.

Создание пользовательской библиотеки

В предыдущих главах мы рассмотрели свойства и методы объектов Application приложений Word и Excel. На основе этих методов были рассмотрены фрагменты приложений, позволяющих создавать и работать с документами Word и Excel, но в больших приложениях работать с объектами Application непосредственно в модулях форм не всегда удобно. Как правило, это связано с тем, что в приложении приходится вставлять и использовать код обработки ошибок или использовать несколько операторов обращения к объектам. Все это приводит к увеличению исходного текста модуля формы, что затрудняет написание сложных приложений. Решать эту проблему можно несколькими способами. Одним из таких способов является разработка и использование *пользовательской библиотеки*. Рассмотрим создание и использование такой библиотеки для работы приложений с документами Word.

Для создания модуля используем команду **File > New** главного меню Delphi, в результате выполнения которой откроется диалоговое окно выбора типа создаваемого объекта (рис. 17.1).

В этом окне выбираем тип создаваемого объекта **Unit**, в результате чего будет создан файл с расширением PAS, представляющий собой пользователь-

ский модуль. В этом модуле пока нет ни одной функции и процедуры (рис. 17.2). Модуль состоит из заголовка и секций interface и implementation.

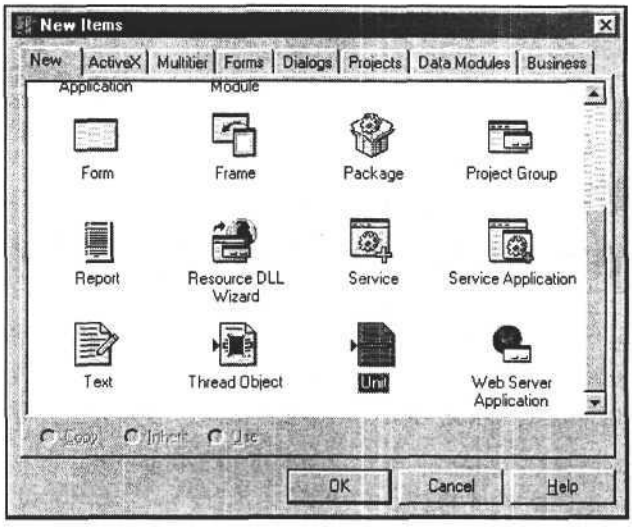


Рис. 17.1. Диалоговое окно выбора типа создаваемого объекта

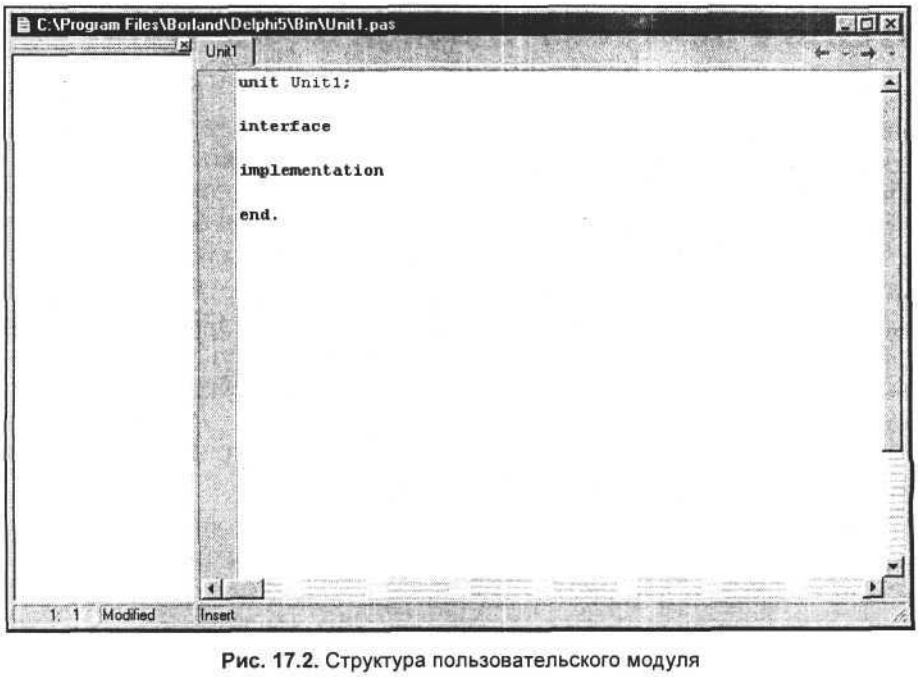


Рис. 17.2. Структура пользовательского модуля

Секция interface должна содержать описания заголовков процедур и функций модуля, а также переменные, которые будут доступны из других модулей. Секция implementation содержит переменные, которые не будут доступны для других модулей, и полные тексты процедур и функций.

Создадим модуль, который будет содержать необходимые функции для работы с документами Word. Заголовок этого модуля представляет собой комбинацию слов "unit MSWORD", где первое является зарезервированным словом, обозначающим заголовок, а второе соответствует имени блока, которое должно совпадать с именем файла. После зарезервированного слова interface перечисляются определения типов, констант и переменных, которые могут быть доступны для других модулей, там же перечисляются и заголовки функций. Следующий исходный текст является фрагментом модуля, в который входят заголовок и секция interface. Создадим в модуле несколько функций для работы с MS Word и опишем заголовки этих функций в секции interface создаваемого модуля.

Заголовок и секция interface создаваемого модуля

```
unit MSWORD;
interface
var CreatedWord:boolean;
Function CreateWord:boolean;
Function VisibleWord(visible:boolean):boolean;
Function AddDoc(file_:string):variant;
Function CloseDoc(document:variant):boolean;
Function CloseDocEx(document:variant;saved_:boolean):boolean;
Function CloseWord:boolean;
Function OpenDoc(file_:string):variant;
Function CopyTextDocToClipboard(doc_:variant):boolean;
Function ImportTextFromDoc(doc_:variant):string;
```

Из имен функций понятно, что они предназначены для создания объекта Word.Application, визуализации окна приложения Word, создания, открытия и закрытия документа, а также для работы с текстовыми модулями.

Рассмотрим секцию implementation и функции, представленные в ней. В начале секции перечисляются библиотеки функций и процедур, используемых в создаваемом модуле, затем следует описание типов, констант и переменных, после которых идет описание процедур и функций модуля.

Секция implementation модуля

```
implementation
uses Windows,Sysutils,ComObj;
  var W:variant;
```

Особо следует отметить ссылку на библиотеку ComObj и переменную W:variant. Мы используем функцию CreateOleObject из этой библиотеки, создающую ссылку на объект Application и сохраняющую его в переменной W. Это реализуется в функции CreateWord, которая в случае удачного выполнения возвращает значение True.

Получение ссылки на объект Application

```
Function CreateWord:boolean;
begin
  CreateWord:=True;
  CreatedWord:=True;
  try
    W:=CreateOleObject('Word.Application');
    except
    CreateWord:=False;
    CreatedWord:=False;
  end;
End;
```

После удачного выполнения функции переменная CreatedWord будет содержать значение True, а переменная W — ссылку на объект Application, после чего можно переходить к выполнению других функций.

Рассмотрим следующие функции для работы с MS Word, например функцию для визуализации окна Word.

Визуализация окна приложения Word

```
Function VisibleWord(Visible:boolean):boolean;
begin
  VisibleWord:=True;
  try
    W.Visible:= Visible;
    except
    VisibleWord:=False;
  end;
End;
```

Данная функция, используя свойство Visible объекта Application, позволяет скрыть или отобразить окно приложения Word. В случае успешного выполнения она возвращает значение True, при возникновении исключительной ситуации ошибка обрабатывается и функция возвращает значение False.

Для создания нового документа или открытия существующего документа, сохраненного в файле, используем функции, возвращающие ссылку на созданный или открытый документ.

Создание документа

```
Function AddDoc(file_:string):variant;
 Var Docs_:variant;
begin
  AddDoc:=null;
  try
    Docs_:=W.Documents;
    AddDoc:=Docs_.Add(file_);
    except
    AddDoc:=null;
  end;
End;
```

В представленной функции переменная Docs_:variant используется для доступа к коллекции документов. После того как в нее была записана ссылка на коллекцию Documents, мы имеем возможность обращаться к свойствам коллекции через эту переменную.

Для открытия документа, созданного ранее и сохраненного в файле, используем следующую функцию.

Открытие существующего документа

```
Function OpenDoc(file_:string):variant;
 Var Docs_:variant;
begin
  OpenDoc:=True;
  try
    Docs_:=W.Documents;
    OpenDoc:=Docs_.Open(file_);
    except
    OpenDoc:=null;
  end;
End;
```

Эта функция возвращает ссылку на элемент коллекции документов, т. е. на документ. Используя эту ссылку, мы получаем доступ к самому документу. Следующая функция, используя ссылку на документ, копирует его содержимое в буфер обмена.

Копирование содержимого документа в буфер обмена

```
Function CopyTextDocToClipboard(doc_:variant):boolean;
begin
  CopyTextDocToClipboard:=True;
  try
    doc_.Range.Select;
    W.Selection.Copy;
    except
    CopyTextDocToClipboard:=False;
  end;
End;
```

Для того чтобы выполнить обратное действие и вставить текст из буфера обмена в документ, нужна другая функция, использующая метод Paste, который вставляет информацию из буфера обмена в документ. Примеры исходного текста, реализующего обмен информации с буфером обмена, можно найти в приложениях на сопроводительном компакт-диске книги.

Обмен информации непосредственно между документом и приложением Delphi можно организовать, воспользовавшись другим способом, реализованным в следующей функции (эта функция возвращает строку текста, импортированную из документа Word).

Импорт текста из документа Word

```
Function ImportTextFromDoc(doc_:variant):string;
begin
  ImportTextFromDoc:='';
  try
    ImportTextFromDoc:=doc_.Range.Text;
    except
    ImportTextFromDoc:='';
  end;
End;
```

После окончания работы с документом необходимо его закрыть, а также закрыть приложение Word. Используем для этого следующие функции. Функцию закрытия документа можно реализовать двумя способами: закрыть документ, если он был до этого сохранен (первый вариант), или сохранить документ в момент его закрытия; закрыть документ без сохранения (второй вариант).

Закрытие документа

```
Function CloseDoc(document:variant):boolean;
begin
  CloseDoc:=True;
  try
    document.Close;
    except
    CloseDoc:=False;
  end;
End;

Function CloseDocEx(document:variant;saved_:boolean):boolean;
begin
  CloseDocEx:=True;
  try
    document.Close(saved_);
    except
    CloseDocEx:=False;
  end;
End;
```

После закрытия документов закрываем приложение Word и освобождаем память компьютера. Для освобождения памяти компьютера используем оператор:

```
W:=UnAssigned;
```

Закрытие приложения Word

```
Function CloseWord:boolean;
begin
  CloseWord:=True;
  try
    W.Quit;
    W:=UnAssigned;
    except
    CloseWord:=False;
  end;
End;
```

Используя перечисленные функции, которые собраны в модуле, сформируем простой документ на основе шаблона — текст на почтовом конверте. Следующая процедура формирует этот документ.

Создание документа на основе шаблона

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  // Создаем новый документ по шаблону
  document:=AddDoc(ExtractFileDir(Application.ExeName)+
                   '\Шаблон конверта.dot');
  messagebox(handle,'Шаблон почтового конверта создан!','Внимание!',0);
  // Подставляем адрес
  FindAllAndPasteTextDoc(document,'###индекс&','350049');
  FindAllAndPasteTextDoc(document,'###адрес&',
                'Краснодар, ул. Севастопольская, д. 3, кв. 123');
  FindAllAndPasteTextDoc(document,'###получатель&',
                'Иванов Иван Иванович');
  // Обратный адрес
  FindAllAndPasteTextDoc(document,'###обратный индекс&','198005');
  FindAllAndPasteTextDoc(document,'###обратный адрес&',
                'Санкт-Петербург, Измайловский пр., д. 29, кв. 111');
  FindAllAndPasteTextDoc(document,'###отправитель&',
                'Петрова Светлана Ивановна');
end;
```

Мы создали, а затем и использовали в своем приложении библиотеку процедур и функций, позволяющую работать с документами Word. Аналогичным образом создается библиотека для работы с рабочими книгами Excel. Исходные тексты этих библиотек представлены в приложении на сопроводительном компакт-диске книги. Их можно легко модифицировать и использовать под свои задачи. Далее рассмотрим создание динамической библиотеки DLL на базе созданного модуля.

Создание пользовательской динамической библиотеки

Для создания динамической библиотеки DLL используем команду **File > New** главного меню Delphi, в результате выполнения которой откроется диалоговое окно выбора типа создаваемого объекта (рис. 17.3).

В этом окне на вкладке **New** выбираем вариант **DLL**, после чего будет создан новый проект. Созданный проект состоит из одного файла, в котором есть заголовок, секция подключаемых и используемых модулей, а также секция, содержащая операторы, выполняемые во время загрузки библиотеки (рис. 17.4).

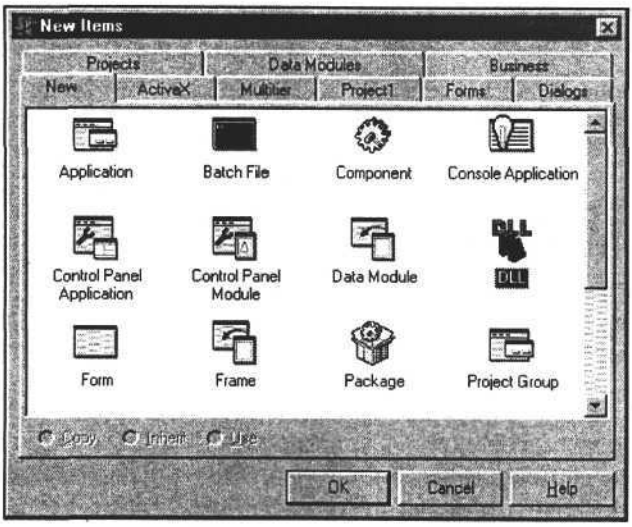


Рис. 17.3. Выбираем тип создаваемого объекта

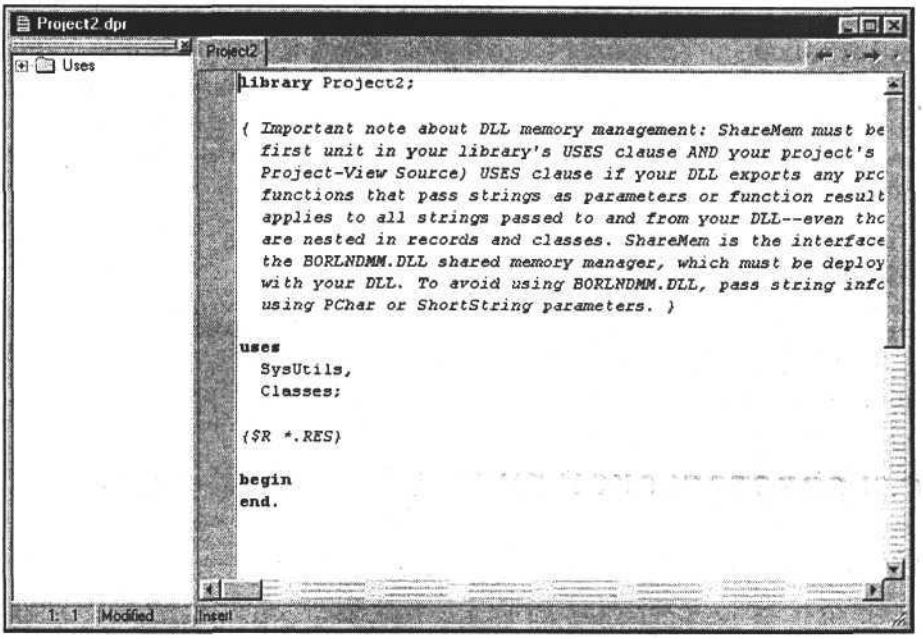


Рис. 17.4. Содержимое файла проекта динамической библиотеки

Очевидно, что созданный проект пока не содержит ни одной функции или процедуры. Преобразуем исходный текст и заголовок этого проекта и добавим в него наши функции для работы с MS Word, созданные ранее.

Содержание файла проекта динамической библиотеки

```
library dserver;
{$D © Корняков В.Н.}
  uses MSWORD;
  exports
CreateWord, VisibleWord, AddDoc,
CloseDoc, CloseDocEx, CloseWord,
OpenDoc, CopyTextDocToClipboard,
FindAllAndPasteTextDoc, ImportTextFromDoc;
begin
end.
```

Как видно из исходного текста, в секции используемых модулей uses указана ссылка на созданную ранее библиотеку MSWORD. В секции exports перечисляются процедуры и функции из этой библиотеки, которые будут доступны для приложений, использующих динамическую библиотеку.

Откомпилируем этот проект, в результате чего получим файл DSERVER.DLL. В данной библиотеке мы используем фрагмент библиотеки, представленной в приложении на сопроводительном компакт-диске книги.

После того как динамическая библиотека создана, переходим к ее использованию. Для использования функций и процедур динамического модуля его сначала нужно загрузить. Есть два способа загрузки модуля DLL — неявный и явный.

Неявная загрузка модуля DLL

Для того чтобы использовать неявную загрузку модуля, достаточно описать процедуры и функции этого модуля, которые вы будете задействовать в своем приложении (лучше сделать это в отдельном модуле unit). Такое описание состоит из имени функции (процедуры), под которым она будет использована в приложении, зарезервированного слова external, строки с именем файла библиотеки DLL, зарезервированного слова name и строки, содержащей имя функции (процедуры) в динамическом модуле. Следующий программный код содержит такое описание.

Описание внешних процедур и функций модуля DLL в приложении

```
function CreateWord_DLL:boolean;
                  external 'dserver.dll' name 'CreateWord';
function VisibleWord_DLL(kod_:boolean):boolean;
                  external 'dserver.dll' name 'VisibleWord';
```

```
function  AddDoc_DLL(filename_:string):variant;
                external 'dserver.dll' name 'AddDoc';
function FindAllAndPasteTextDoc_DLL(document:variant;findtext_,
                                    pastetext_:string):boolean;
          external 'dserver.dll' name 'FindAllAndPasteTextDoc';
function OpenDoc_DLL(file_:string):variant;
                external 'dserver.dll' name 'OpenDoc';
function  CopyTextDocToClipboard_DLL(document:variant):boolean;
          external 'dserver.dll' name 'CopyTextDocToClipboard';
function ImportTextFromDoc_DLL(document:variant):string;
          external 'dserver.dll' name 'ImportTextFromDoc';
function CloseDocEx_DLL(document:variant;saved_:boolean):boolean;
          external 'dserver.dll' name 'CloseDocEx';
function CloseWord_DLL:boolean; external 'dserver.dll' name 'CloseWord';
```

Если модуль DLL откомпилирован, размещен на диске компьютера и доступен приложению, а описание процедур и функций этого модуля в использующем его приложении не содержит ошибок, то приложение запустится без проблем. Тогда во время запуска приложения также будет загружен динамический модуль. Если файл DLL не доступен приложению, которое его использует и загружает неявно, то загрузка приложения вызовет ошибку. Если в модуле отсутствует функция, описываемая в приложении, то это также вызовет ошибку (рис. 17.5 и 17.6).

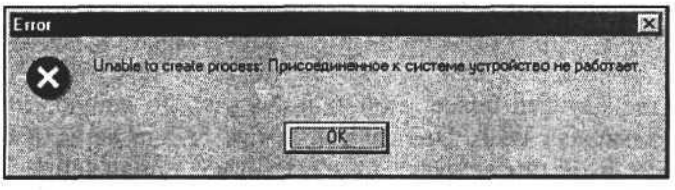


Рис. 17.5. Ошибка загрузки модуля DLL

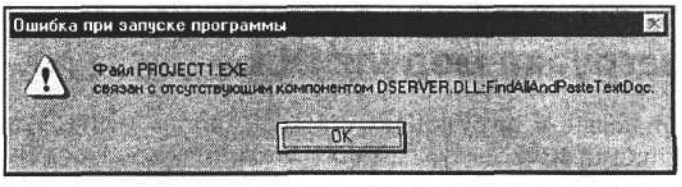


Рис. 17.6. Ошибка при отсутствии в модуле DLL функции, описанной в приложении

Достоинство метода неявной загрузки модуля DLL заключается в простоте его использования. Недостаток метода проявляется во время загрузки приложения на выполнение, если описания функций (процедур) в приложении и модуле не совпадают. Для решения этой проблемы приходится или

корректировать исходный текст приложения или исправлять ошибки в модуле DLL.

Используем созданную библиотеку в приложении Delphi.

Создание документа на основе шаблона (неявная загрузка модуля)

```
procedure TForm1.Button8Click(Sender: TObject);
begin
// Создаем новый документ по шаблону
  document:=AddDoc_DLL(ExtractFileDir(Application.ExeName)+
                       '\Шаблон конверта.dot');
  messagebox(handle,'Шаблон почтового конверта создан!','Внимание!',0);
// Подставляем адрес
  FindAllAndPasteTextDoc_DLL(document,'###индекс&','350049');
  FindAllAndPasteTextDoc_DLL(document,'###адрес&',
                     'Краснодар, ул. Севастопольская, д. 3, кв. 123');
  FindAllAndPasteTextDoc_DLL(document,'###получатель&',
                              'Иванов Иван Иванович');
// Обратный адрес
  FindAllAndPasteTextDoc_DLL(document,'###обратный индекс&','198005');
  FindAllAndPasteTextDoc_DLL(document,'###обратный адрес&',
                  'Санкт-Петербург, Измайловский пр., д. 29, кв. 111');
FindAllAndPasteTextDoc_DLL(document,'###отправитель&',
                           'Петрова Светлана Ивановна');
end;
```

В большинстве случаев более удобно использовать явную, или динамическую загрузку модуля DLL. Это связано с тем, что даже если будут изменены некоторые функции модуля, то на загрузке приложения это никак не отразится. Не будут выполняться только те функции и процедуры, которые были ошибочно изменены.

Явная загрузка модуля DLL

В отличие от неявной загрузки модуля DLL, когда имена функций импортируемых функций и процедур, а также библиотека должны быть определены на этапе формирования исходного текста приложения и его компиляции, использование явной загрузки позволяет задавать имя динамической библиотеки и имя функции непосредственно перед ее выполнением. Для явной загрузки модулей DLL необходимо загрузить модуль DLL в адресное пространство процесса, получить точку входа для нужной функции, выполнить внешнюю функцию и освободить память процесса от модуля. Для это-

го используются следующие процедуры и функции API Win32: LoadLibrary и LoadLibraryEx, GetProcAddress и FreeLibrary.

Для сравнения явной загрузки модуля с неявной загрузкой рассмотрим уже знакомый пример заполнения почтового конверта. Для использования импортируемых функций зададим их типы и на их основании определим переменные, которые будут использованы как точки входа в процессе выполнения приложения.

Задание типов функций, импортируемых из модуля

```
type
TCreateWord        = function:boolean;
TVisibleWord       = function(kod_:boolean):boolean;
TAddDoc            = function(filename_:string):variant;
TFindAllAndPasteTextDoc = function(document:variant;findtext_,
                                   pastetext_:string):boolean;
TOpenDoc = function(file_:string):variant;
TCopyTextDocToClipboard = function(document:variant):boolean;
TImportTextFromDoc = function(document:variant):string;
TCloseDocEx = function(document:variant;saved_:boolean):boolean;
TCloseWord = function:boolean;
```

Задав типы импортируемых из модуля DLL функций, переходим к определению переменных, которые будут служить точками входа. Необходимо также определить переменную, которую будем использовать для обращения к модулю.

Задание переменных для точек входа в импортируемые функции

```
var  hdll:thandle;
 CreateWord_DLL2:TCreateWord;
 VisibleWord_DLL2:TVisibleWord;
 AddDoc_DLL2:TAddDoc;
 FindAllAndPasteTextDoc_DLL2:TFindAllAndPasteTextDoc;
 OpenDoc_DLL2:TOpenDoc;
 CopyTextDocToClipboard_DLL2:TCopyTextDocToClipboard;
 ImportTextFromDoc_DLL2:TImportTextFromDoc;
 CloseDocEx_DLL2:TCloseDocEx;
 CloseWord_DLL2:TCloseWord;
```

Загружаем модуль с помощью функции LoadLibrary, которая в случае успеха возвращает указатель на загруженный модуль.

Загрузка модуля

```
procedure TForm1.Button19Click(Sender: TObject);
begin
  hdll:=LoadLibrary('Dserver.dll');
end;
```

Переходим непосредственно к выполнению функций модуля, которые создадут и заполнят конверт на основе заготовленного шаблона.

Сначала определим точки входа для используемых процедур с помощью функции GetProcAddress, аргументами которой являются указатель на загруженный модуль и строка с именем функции, импортируемой из модуля. После инициализации функций выполняем их (исходный текст в этой части программы не отличается от исходного текста при использовании неявной загрузки модуля).

Создание документа на основе шаблона (явная загрузка модуля)

```
procedure TForm1.Button14Click(Sender: TObject);
begin
// Определяем точки входа функций
  AddDoc_DLL2:=GetProcAddress(hdll,'AddDoc');
  FindAllAndPasteTextDoc_DLL2:=
                    GetProcAddress(hdll,'FindAllAndPasteTextDoc');
// Создаем новый документ по шаблону
  document:=AddDoc_DLL2(ExtractFileDir(Application.ExeName)+
                        '\Шаблон конверта.dot');
  messagebox(handle,'Шаблон почтового конверта создан!','Внимание!',0);
// Подставляем адрес
  FindAllAndPasteTextDoc_DLL2(document,'###индекс&','350049');
  FindAllAndPasteTextDoc_DLL2(document,'###адрес&',
                    'Краснодар, ул. Севастопольская, д. 3, кв. 123');
FindAllAndPasteTextDoc_DLL2(document,'###получатель&',
                          'Иванов Иван Иванович');
// Обратный адрес
  FindAllAndPasteTextDoc_DLL2(document,'###обратный индекс&','198005');
FindAllAndPasteTextDoc_DLL2(document,'###обратный адрес&',
                  'Санкт-Петербург, Измайловский пр., д. 29, кв. 111');
FindAllAndPasteTextDoc_DLL2(document,'###отправитель&',
                           'Петрова Светлана Ивановна');
end;
```

Результат выполнения процедуры представлен на рис. 17.7. Этот результат ничем не отличался бы от результата использования неявной загрузки мо-

дуля или использования обычной библиотеки, подключаемой к создаваемому проекту.

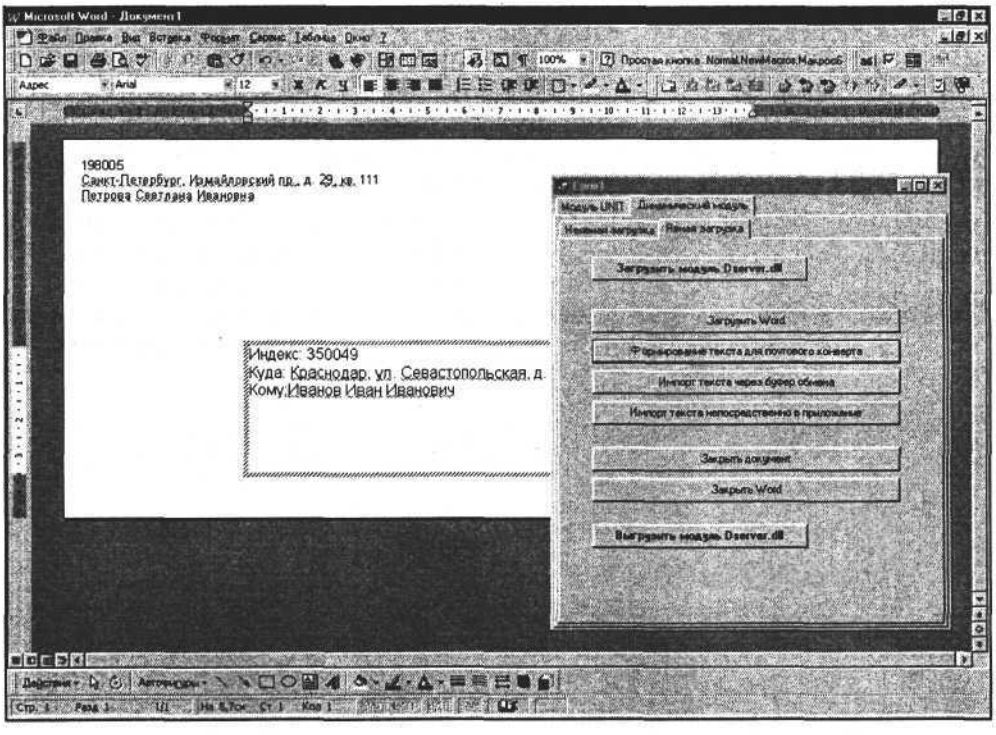


Рис. 17.7. Сформированный документ

После выполнения функций можно освободить память процесса и выгрузить динамический модуль, если в нем больше нет необходимости.

Освобождение памяти

```
procedure TForm1.Button20Click(Sender: TObject);
begin
  FreeLibrary(hdll);
end;
```

В этой главе мы рассмотрели создание и использование динамически загружаемых модулей для работы с документами MS Word в приложениях Delphi. Справедливости ради стоит отметить, что и сами приложения MS Office могут работать с внешними динамическими модулями. Эта возможность очень заманчива и может дать большой положительный эффект при создании и использовании приложений на любом языке, позволяющем создавать динамические библиотеки. Следующая глава будет посвящена этому.

ГЛАВА 18

Использование DLL в макросах MS Office

Эта книга посвящена использованию объектов MS Office для создания документов Word и Excel и последующего изменения этих документов с помощью внешних приложений, разработанных в среде Delphi. Применение объекта Application и его свойств (в том числе объектов и коллекций объектов) позволяет управлять не только документами, но и свойствами самих приложений Word и Excel. Для программиста, решившего создавать документы (например, отчеты) на основе MS Office, это открывает широкие возможности.

Приложения MS Office могут использовать Visual Basic — встроенный язык, обладающий достаточной функциональностью для создания документов любой сложности. Можно спросить: зачем тогда использовать Delphi, может быть лучше сразу писать приложение на Visual Basic? У тех, кто профессионально работает в среде Delphi, такой вопрос не возникнет. Среда Delphi является инструментом создания приложений, которым доступны все возможности операционных систем Windows и поддерживаемых ими баз данных. Visual Basic для MS Office как инструмент больше ориентирован на создание и управление документами. Из этого следует, что наиболее эффективным решением будет не противопоставление, а объединение возможностей этих инструментов.

Рассмотрим еще один способ совместного использования Visual Basic и приложений Delphi. Только теперь управляющей программой будет приложение документа MS Office, а управляемой — приложение, разработанное в среде Delphi. Речь пойдет о частном случае применения созданных динамических библиотек (файлов DLL) в макросах рабочей книги Excel, позволяющих использовать дополнительные функции для формирования документа.

Например, вы разработали и тщательно отладили достаточно сложное приложение, которое работает с распределенными базами данных и имеет набор диалогов ввода и вывода информации. Поставлена задача: для некото-

рых пользователей, работающих и создающих отчеты исключительно в Excel, предоставить интерфейс, используя стандартные для вашей программы диалоговые окна, и организовать загрузку данных непосредственно в документ. Один из способов решения этой задачи — разработка динамической библиотеки, функции которой вызывают необходимые диалоги и обеспечивают обмен информацией между документом и созданным приложением. Средства Visual Basic позволяют использовать внешние функции, но перед использованием их нужно описать.

Описание внешних функций и процедур в модуле документа

Для начала рассмотрим, как можно получить доступ к функциям и процедурам из стандартной динамической библиотеки и какие ограничения накладываются на них и на их аргументы и возвращаемые значения. В таких функциях можно использовать не все типы переменных — только некоторые типы данных в Excel и Delphi имеют общую структуру. Подходят следующие типы данных: String — строка длиной до нескольких гигабайт, ограниченная символом CHR(0); Longint (Integer) — целое число от −2 147 483 648 до 2 147 483 647; Byte — целое число от 0 до 255; Boolean — логическая переменная (True / False); Variant — тип данных, к которому относятся все переменные, не описанные явно. Чтобы подробней ознакомиться с данным вопросом и получить список всех возможных типов данных, используйте справку по Excel, раздел "Справочник по Visual Basic".

Определив, какими типами данных будем оперировать, попробуем описать и вызвать какую-либо функцию или процедуру из стандартной библиотеки Windows. В качестве примера можем поработать с функцией Messagebox из стандартной динамической библиотеки. Функция Messagebox использует в качестве входных и возвращаемых значений типы PChar и Integer. В макросе Excel тип PChar можно заменить типом String. Прежде чем обращаться к внешней процедуре, опишем ее.

Описание внешних процедур и функций, используемых в макросах, имеет следующий синтаксис.

Описание процедуры

```
Declare Sub "имя процедуры" Lib "имя библиотеки" [Alias "псевдоним"]
  [(["список аргументов"])]
```

Описание функции

```
Declare Function "имя функции" Lib "имя библиотеки" [Alias "псевдоним"]
  [(["список аргументов"])] [As "тип"]
```

Ключевые слова: Sub — указывает, что процедура не возвращает значение; Function — указывает, что процедура возвращает значение, которое может быть использовано в выражении; Lib — указывает, что описываемая процедура содержится в динамической библиотеке, *"имя библиотеки"* — имя динамической библиотеки; Alias — указывает, что вызываемая процедура имеет другое имя в библиотеке, *"псевдоним"* — имя процедуры в библиотеке; *"список аргументов"* — список переменных, представляющий аргументы, которые передаются в процедуру при ее вызове; *"тип"* — тип данных значения, возвращаемого процедурой типа Function.

В нашем примере из рабочей книги Excel вызовем функцию Messagebox. Ссылку на эту функцию можно найти в файле WINDOWS.PAS, входящем в состав Delphi. Используем полученную информацию. Из описания, представленного в файле WINDOWS.PAS, видно, что эта функция входит в состав библиотеки USER32.DLL.

Описание ссылки на функцию Messagebox

```
Interface
...
function MessageBoxA(hWnd:HWND;lpText,lpCaption:PAnsiChar;
                     uType:UINT):Integer; stdcall;
...
implementation
...
function MessageBoxA; external user32 name 'MessageBoxA';
```

В макросе в объявлении функции мы напишем строку, определяемую следующим синтаксисом.

Описание функции Messagebox в макросе Excel

```
Declare Function MessageBoxA Lib "user32.dll" (ByVal hwnd As Integer,
                    ByVal lpText As String, ByVal lpCaption As String,
                    ByVal uType As Integer) As Integer
```

Представленное описание указывает на то, что мы используем функцию, возвращающую целое значение. Функция импортируется из библиотеки USER32.DLL. Имена функции в библиотеке и в макросе совпадают, но для макроса можно было бы задать и другое имя. Аргументы импортируемой функции не возвращают значения, а только передают. Для аргументов, возвращающих значения, необходимо опустить зарезервированное слово ByVal. В теле макроса напишем следующую процедуру для вызова функции MessageBox динамической библиотеки user32.dll.

Вызов функции Messagebox из макроса

```
Public Sub Message()
Call MessageBoxA(0, "Тип объекта = " + TypeName(Selection),
          "Вызов функции MessageBox из библиотеки user32.dll", 0)
Call MessageBoxA(0, "Значение ячейки = " + Range("A1").Text,
          "Вызов функции MessageBox из библиотеки user32.dll", 0)
End Sub
```

Данная процедура дважды вызывает функцию MessageBox. Первый вызов процедуры отображает значение типа выделенного объекта. Второй вызов выводит в диалоговом окне значение, записанное в ячейке A1. Выполним этот макрос и получим результат в виде диалогового окна (рис. 18.1).

Рис. 18.1. Открытие диалогового окна из макроса Excel

Соглашение о вызовах

Разрабатывая динамическую библиотеку процедур и функций, необходимо помнить, что к созданной библиотеке могут обращаться приложения, написанные на разных языках и скомпилированные в разных средах программирования. Поэтому при обращении к функциям и процедурам используются различные способы передачи их параметров. Передача параметров при вызове динамических функций производится через стек или регистры процессора. Помещаемые в стек или регистр параметры могут следовать в различном порядке. После передачи параметров стек должен очищаться вызывающим процессом или вызываемой процедурой. Поэтому соглашение о вызовах это способ передачи параметров в вызываемую функцию или процедуру.

В табл. 18.1 приведены директивы соглашения о вызовах.

Таблица 18.1. *Директивы соглашения о вызовах*

Директива	Порядок следования параметров	Очищает стек, регистры	Использование регистров для передачи параметров
Register	Слева направо	Вызываемая процедура	Да

Таблица 18.1 (окончание)

Директива	Порядок следования параметров	Очищает стек, регистры	Использование регистров для передачи параметров
Pascal	Слева направо	Вызываемая процедура	Нет
Cdecl	Справа налево	Вызывающая процедура	Нет
Stdcall	Справа налево	Вызываемая процедура	Нет
Safecall	Справа налево	Вызываемая процедура	Нет

Директива, определяющая тот или иной способ передачи параметров, приводится в заголовке процедуры или функции и указывает компилятору соглашение о вызовах, применяемое для данной функции. В языке Pascal по умолчанию всегда используется директива Register. Она определяет, что параметры будут передаваться через регистры процессора, в порядке слева направо, и регистры будет очищать вызываемая процедура. Директива Cdecl указывает компилятору, что вызов функции должен быть осуществлен в стиле языка C. Параметры загружаются в стек справа налево, затем стек очищается вызывающим процессом. Директива Pascal извещает компилятор, что необходимо сгенерировать код в стиле языка Pascal. Параметры процедуры помещаются в стек слева направо, стек очищает вызываемая процедура. Директивы Stdcall и Safecall, как и Cdecl, указывают компилятору помещать в стек параметры слева направо, но стек очищает вызываемая процедура, как в случае применения директивы Pascal. Если мы будем использовать создаваемую динамическую библиотеку для вызова процедур и функций из макросов Excel, то лучше в их описании использовать директиву Stdcall:

```
Function GetDateDialog:variant; Stdcall;
```

Рассмотрим программный код динамической библиотеки, которую мы будем использовать в макросах документов MS Excel.

Создание в среде Delphi динамической библиотеки для ее использования в макросах Excel

В качестве примера создадим динамическую библиотеку, состоящую из двух функций, одна из которых содержит диалоговое окно выбора даты (она бу-

дет возвращать дату). Другая функция будет просто преобразовывать значение аргумента, заданного в формате денежной единицы, и возвращать его в виде текстовой строки. Весь проект будет состоять из файла проекта Delphi LIBFOROF.DPR, модуля MYLIB.PAS, формы диалогового окна задания даты и программного модуля к этой форме (полный текст приложения представлен на сопроводительном компакт-диске книги).

Проект динамической библиотеки экспортируемых функций

```
library LibForOf;
uses
 SysUtils,
 Classes,
 MyLib;
{$R *.RES}
 exports
 GetDateDialog,
 GetSumInWords;
begin
end.
```

Текст файла проекта содержит имя библиотеки, список используемых модулей и список экспортируемых функций. Модуль MYLIB.PAS содержит исходные тексты экспортируемых функций. Обратим внимание на то, что при описании функций используется соглашение о вызове Stdcall, позволяющее компилировать библиотеку так, чтобы она была совместима для использования в программных модулях документов Excel.

Исходный текст модуля MYLIB.PAS

```
unit MyLib;

interface
Function GetDateDialog:variant;  Stdcall;
Function GetSumInWords(cur_:currency; var stroka_:string):boolean;
  Stdcall;

implementation
  uses windows,Sysutils,Forms,Controls,DateDlg;
  var j_:longint;

Function GetDateDialog:variant;
begin
Application.CreateForm(TDateDlg_,DateDlg_);
```

```
if DateDlg_.ShowModal=mrOk
then GetDateDialog:=DateDlg_.MonthCalendar1.Date
else GetDateDialog:=null;
DateDlg_.Free;
End;

Function currtostrSS(str_:shortstring;zpz_:word):shortstring;
begin
if pos(DecimalSeparator,str_)>0
then str_:=copy(str_+'00000',1,pos(DecimalSeparator,str_)+zpz_)
else str_:=str_+DecimalSeparator+copy('00000',1,zpz_);
currtostrSS:=str_;
End;

Function Del_sstr(ss_:shortstring;var str_:shortstring):boolean;
  var a_:word;
begin
Del_sstr:=false; a_:=pos(ss_,str_);
if pos(ss_,str_)=0 then exit;
delete(str_,a_,length(ss_)); Del_sstr:=True;
End;

Function Get_strok_num(var ss1,ss2:shortstring;
           razd_:char;nn_raz_:byte) :byte;
const
src_1:array[0..20]of string[15]=('','один','два',
                    'три','четыре','пять','шесть','семь',
                    'восемь','девять','десять','одиннадцать',
                    'двенадцать','тринадцать','четырнадцать',
                    'пятнадцать','шестнадцать','семнадцать',
                    'восемнадцать','девятнадцать','двадцать');
src_1_:array[0..20]of string[15]=('','одна','две',
                    'три','четыре','пять','шесть','семь',
                    'восемь','девять','десять','одиннадцать',
                    'двенадцать','тринадцать','четырнадцать',
                    'пятнадцать','шестнадцать','семнадцать',
                    'восемнадцать','девятнадцать','двадцать');
src_2:array[0..9]of string[15]=('','десять','двадцать',
                    'тридцать','сорок','пятьдесят','шестьдесят',
                    'семьдесят','восемьдесят','девяносто');
src_3:array[0..9]of string[15]=('','сто','двести',
                    'триста','четыреста','пятьсот','шестьсот',
                    'семьсот','восемьсот','девятьсот');
```

```
src_4:array[0..4]of string[15]=('','тысяча','миллион',
                                'миллиард','триллион');
src_5:array[0..4]of string[15]=('','тысячи','миллиона',
                               'миллиарда','триллиона');
src_6:array[0..4]of string[15]=('','тысяч','миллионов',
                              'миллиардов','триллионов');
rub_:array[0..2]of string[10]=('рубль','рубля','рублей');
kop_:array[0..2]of string[10]=('копейка','копейки','копеек');
   var cc1_,cc2_:string[20];
          a_,b_:byte;
         razrad:array[1..5,1..3]of byte;

Function preob(ccc:shortstring;kod_:byte):shortstring;
  type mema=array[0..100]of byte;
  var eee_:shortstring;
    a_,b_:byte;
    mem_:^mema;
begin
for a_:=1 to length(ccc) do
if ((ord(ccc[a_])<48)or(ord(ccc[a_])>57))
then begin preob:='0'; exit; end;
fillchar(razrad,sizeof(razrad),48);
mem_:=@razrad;
for a_:=1 to length(ccc) do
mem_[a_-1+sizeof(razrad)-length(ccc)]:=ord(ccc[a_]);
eee_:='';
for b_:=1 to 5 do begin
for a_:=1 to 3 do begin
if (a_=1)and(razrad[b_,a_]>48) then eee_:=eee_+' '+
                src_3[razrad[b_,a_]-48];
if razrad[b_,2]>49 then begin
if (a_=2)and(razrad[b_,a_]>48) then eee_:=eee_+' '+
                  src_2[razrad[b_,a_]-48];
if b_<>4 then
  if (a_=3)and(razrad[b_,a_]>48)and(kod_=1) then eee_:=eee_+
                              ' '+src_1[razrad[b_,a_]-48];
if (a_=3)and(razrad[b_,a_]>48)and(kod_=2) then eee_:=eee_+' '+
                                   src_1_[razrad[b_,a_]-48];
if b_=4 then
if (a_=3)and(razrad[b_,a_]>48) then eee_:=eee_+' '+
                                 src_1_[razrad[b_,a_]-48];
end;
```

```
if razrad[b_,2]=48 then begin
if (a_=2)and(razrad[b_,a_]>48) then eee_:=eee_+' '+
                          src_2[razrad[b_,a_]-48];
if b_<>4 then
if (a_=3)and(razrad[b_,a_]>48)and(kod_=1) then eee_:=eee_+' '+
                                  src_1[razrad[b_,a_]-48];
if (a_=3)and(razrad[b_,a_]>48)and(kod_=2) then eee_:=eee_+' '+
                                src_1_[razrad[b_,a_]-48];
if b_=4 then
if (a_=3)and(razrad[b_,a_]>48) then eee_:=eee_+' '+
                            src_1_[razrad[b_,a_]-48];
end;
if (razrad[b_,2]=49)and(a_=2) then begin
eee_:=eee_+' '+src_1[(razrad[b_,2]-48)*10+(razrad[b_,3]-48)];
end;
end;
if razrad[b_,2]=48 then begin
if (razrad[b_,3]=49)then eee_:=eee_+' '+src_4[5-b_];
if ((razrad[b_,3]>49)and(razrad[b_,3]<53))
  then eee_:=eee_+' '+src_5[5-b_];
if (razrad[b_,3]>52)then eee_:=eee_+' '+src_6[5-b_];
end;
if razrad[b_,2]=49 then begin
    if (razrad[b_,3]>=48)then eee_:=eee_+' '+src_6[5-b_];
end;
if (razrad[b_,2]=48)and(razrad[b_,3]=48) then begin
    if razrad[b_,1]>48 then eee_:=eee_+' '+src_6[5-b_];
end;
if razrad[b_,2]>49 then begin
if (razrad[b_,3]=49)then eee_:=eee_+' '+src_4[5-b_];
if ((razrad[b_,3]>49)and(razrad[b_,3]<53))
then eee_:=eee_+' '+src_5[5-b_];
if (razrad[b_,3]>52)then eee_:=eee_+' '+src_6[5-b_];
if (razrad[b_,3]=48)then eee_:=eee_+' '+src_6[5-b_];
end;
end;
preob:=eee_;
end;

begin
Get_strok_num:=0;
if (pos(razd_,ss1)>0)and(length(ss1)-pos(razd_,ss1)>nn_raz_)
  then ss1:=copy(ss1,1,pos(razd_,ss1)+nn_raz_);
```

```
if pos(razd_,ss1)>0
 then ss1:=ss1+copy('000000',1,nn_raz_-(length(ss1)-pos(razd_,ss1)));
if pos(razd_,ss1)>0 then begin
 cc1_:=copy(ss1,1,pos(razd_,ss1)-1);
 cc2_:=copy(ss1,pos(razd_,ss1)+1,length(ss1)-pos(razd_,ss1));
end
else begin
 cc1_:=ss1; cc2_:='';
end;
if length(cc1_)>15 then begin get_strok_num:=2; exit; end;
if length(cc2_)>15 then begin get_strok_num:=3; exit; end;
ss1:=preob(cc1_,1); ss2:=preob(cc2_,2);
End;

Function get_rub_kop(var rub,kop:shortstring):byte;
  label 0,1,2;
  const
  rub_:array[0..2]of string[10]=('рубль','рубля','рублей');
  kop_:array[0..2]of string[10]=('копейка','копейки','копеек');
  begin
  get_rub_kop:=0;
  try
  if rub='' then begin rub:=''; goto 1; end;
  if ((strtoint(rub) mod 100 >4)and(strtoint(rub)mod 100 <=20))
    then begin rub:=rub_[2]; goto 1; end;
  if (strtoint(rub) mod 10>1)and(strtoint(rub) mod 10<5)
    then begin rub:=rub_[1]; goto 1; end;
  if (strtoint(rub) mod 10=1) then begin rub:=rub_[0]; goto 1; end;
  if ((strtoint(rub) mod 10>4)and(strtoint(rub) mod
  0<=9))or(strtoint(rub) mod 10=0) then begin rub:=rub_[2]; goto 1; end;
1:
  except
  get_rub_kop:=1;
  end;
  try
  if kop='' then begin kop:=''; goto 2; end;
  if ((strtoint(kop) >4)and(strtoint(kop) <=20))
    then begin kop:=kop_[2]; goto 2; end;
  if (strtoint(kop) mod 10>1)and(strtoint(kop) mod 10<5)
    then begin kop:=kop_[1]; goto 2; end;
  if (strtoint(kop) mod 10=1) then begin kop:=kop_[0]; goto 2; end;
  if ((strtoint(kop) mod 10>4)and(strtoint(kop) mod
  10<=9))or(strtoint(kop) mod 10=0) then begin kop:=kop_[2]; goto 2; end;
```

```
2:
  except
  get_rub_kop:=2;
  end;
  End;

Function GetSumInWords(cur_:currency; var stroka_:string):boolean;
 label 1;
 var ss_r,ss_k,rub_,ckop_,kop_:shortstring;
   str_,eee_,vux_:string;
   a_,b_:word;
begin
1:
str_:=currtostr(cur_);
if str_<>'' then
 begin ss_r:=currtostrSS(str_,2); while Del_sstr(' ',ss_r) do;
    rub_:=ss_r; kop_:='';
    if pos(DecimalSeparator,ss_r)>0 then
begin rub_:=copy(ss_r,1,pos(DecimalSeparator,ss_r)-1);
if length(ss_r)>pos(DecimalSeparator,ss_r)
then kop_:=copy(ss_r,pos(DecimalSeparator,ss_r)+1,
              length(ss_r)-pos(DecimalSeparator,ss_r))
else kop_:=''; ss_r[pos(DecimalSeparator,ss_r)]:=','; end;
ckop_:=kop_; get_strok_num(ss_r,ss_k,',',2); get_rub_kop(rub_,kop_);
if ckop_<>'' then vux_:=ss_r+' '+rub_+' '+ckop_+' '+kop_
else vux_:=ss_r+' '+rub_;
vux_:=copy(vux_,2,length(vux_)-1);
eee_:=vux_; eee_[1]:=chr(ord(eee_[1])-32); stroka_:=eee_;
end; end;
End.
```

После компиляции проекта мы получим файл LIBFOROF.DLL и можем использовать доступные процедуры и функции, которые содержатся в этой библиотеке. Для этого расположим файл библиотеки таким образом, чтобы он был доступен приложению Excel. Это можно сделать, поместив файл LIBFOROF.DLL в одной папке с Excel или в папке, в которой он будет доступен для приложения Excel, например, в папке Windows\System32.

Использование созданной динамической библиотеки

Переходим к использованию в Excel функций динамической библиотеки, созданной в Delphi. Для начала используем функцию GetDateDialog, кото-

рая будет по нажатию кнопки открывать диалоговое окно и вставлять дату в выделенную ячейку таблицы.

Вот исходный текст объявления функции и макроса.

Использование функции ввода даты в ячейку листа рабочей книги

```
Declare Function GetDateDialog Lib "LibForOf.dll" () As Variant
Public Sub DateDialog()
  Selection = GetDateDialog
End Sub
```

На рис. 18.2 представлен результат выполнения процедуры GetDateDialog.

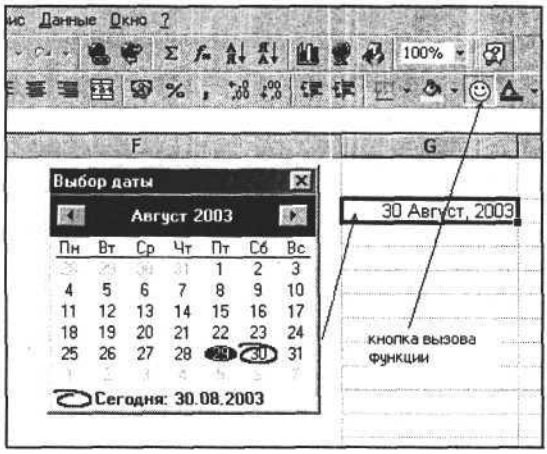


Рис. 18.2. Открытие диалогового окна выбора даты

Очевидно, что такой способ выбора даты будет неудобен пользователю. Предложим второй вариант, когда диалог выбора даты будет открываться по двойному щелчку левой кнопки мыши в ячейке таблицы. Здесь мы используем стандартную процедуру отклика на двойной щелчок. В эту процедуру передается ссылка на объект Range, в который необходимо записать дату. Вот исходный текст этой процедуры.

Использование процедуры отклика на двойной щелчок

```
Private Sub Worksheet_BeforeDoubleClick(ByVal Target As Excel.Range,
    Cancel As Boolean)
Target = GetDateDialog
Target.NumberFormat = "d mmmm, yyyy"
End Sub
```

Использование данной процедуры позволит отказаться от кнопки и сократит количество действий пользователя. Дополнительно в этой процедуре выполняется преобразование формата ячейки для обеспечения правильного отображения даты.

Рассмотрим использование процедуры GetSumInWords из состава динамической библиотеки LIBFOROF.DLL, которую мы создали для преобразования числового значения типа Currency в строковый эквивалент. В объявлении этой процедуры обратите внимание на описание ее параметров. Первый параметр объявляется с ключевым словом ByVal, указывающим на то, что этот параметр только передает и не возвращает данные. Второй параметр описан без ключевых слов, он возвращает строку — эквивалент значения первого параметра. В качестве второго параметра мы используем переменную str_, которую декларируем непосредственно в теле функции SumInWords. В данном случае в модуле рабочей книги нам удобней использовать функцию, потому что ее можно будет применять в формулах Excel. Исходные тексты, представленные далее, дополняют сказанное.

Использование функции преобразования числа в строковый эквивалент

```
Declare Function GetSumInWords Lib "LibForOf.dll" (ByVal cur As Currency,
  stroka_ As String) As Boolean
Public Function SumInWords(cur_ As Currency) As String
Dim str_ As String
Call GetSumInWords(cur_, str_)
SumInWords = str_
End Function
```

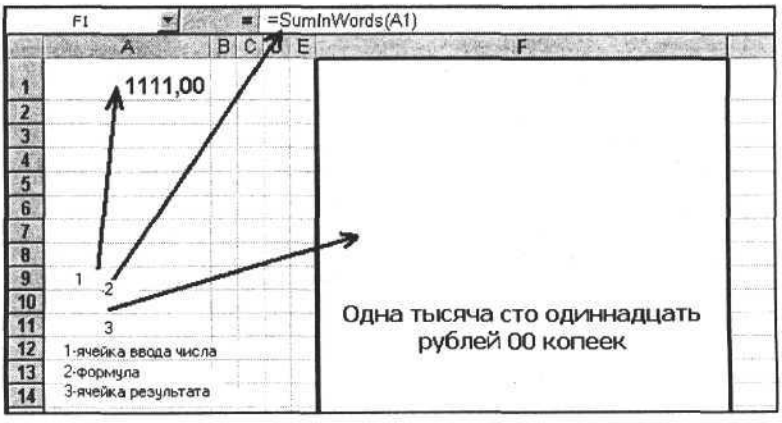
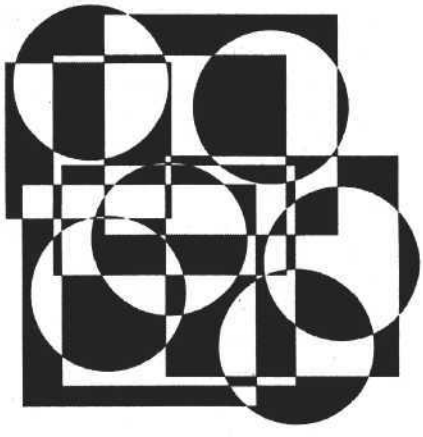


Рис. 18.3. Преобразование числового значения в строковый эквивалент

После того как макрос записан и протестирован, переходим к его использованию. Для этого в ячейку F1 запишем формулу =SumInWords(A1). Эта формула позволит отображать в ячейке F1 строковый эквивалент числового значения, введенного в ячейку A1 (рис. 18.3).

В данной главе мы рассмотрели, как создавать и использовать динамические библиотеки в модулях документов MS Office. Были рассмотрены простые примеры, которые при желании можно выполнить и средствами языка Visual Basic. Но для экспорта и импорта информации в приложениях, созданных на базе Delphi, представленная информация весьма полезна.



ПРИЛОЖЕНИЯ

Приложение 1. Объекты, свойства и методы

Приложение 2. Ответы на вопросы

Приложение 3. Описание компакт-диска

ПРИЛОЖЕНИЕ 1

Объекты, свойства и методы

Приложение MS Word

Таблица П1.1. *Свойства объекта Application*

Свойство	Тип	Описание
ActiveDocument	Объект	Активный документ
ActivePrinter	String	Строка, содержащая имя принтера по умолчанию
ActiveWindow	Объект	Ссылка на активное окно, т. е. окно, которое получило фокус ввода
AddIns	Коллекция	Библиотека шаблонов стилей документов
Assistant	Объект	Ссылка на Помощник (анимированный элемент интерфейса)
AutoCaptions	Коллекция	Заголовки объектов, автоматически добавляемые, когда в документ вставляются объекты
AutoCorrect	Объект	Содержит информацию для автозамены слов в тексте
BackgroundPrintingStatus	Integer	Положение в очереди печати
BackgroundSavingStatus	Integer	Количество документов, сохраняемых в файлах
Build	String	Информация о версии MS Word
CapsLock	Boolean	True — включен режим CAPS LOCK (нажата клавиша <Caps Lock>)

Таблица П1.1 *(продолжение)*

Свойство	Тип	Описание
Caption	String	Строка, которая содержит заголовок главного окна приложения
CommandBars	Коллекция	Панели элементов управления приложения Word
CustomDictionaries	Коллекция	Словари
DefaultSaveFormat	String	Тип формата сохраняемого документа по умолчанию (текстовая строка)
DefaultTableSeparator	String	Символ-разделитель, используемый для преобразования текста в таблицу
Dialogs	Коллекция	Диалоги
DisplayRecentFiles	Boolean	True — меню **Файл** содержит список недавно открывавшихся файлов
DisplayScreenTips	Boolean	True — комментарии, сноски, ссылки и т. п. выделяются в документе
DisplayScrollBars	Boolean	True — полосы прокрутки отображаются, False — полосы прокрутки скрыты
DisplayStatusBar	Boolean	True — отображается строка состояния
Documents	Коллекция	Открытые документы
FileConverters	Коллекция	Файлы-конверторы, применяемые для преобразования документов во время их открытия, если файлы документов имеют формат, отличный от стандартных форматов
FileSearch	Объект	Используется для поиска файлов
FindKey	Объект	Ссылка на объект, ассоциированный с комбинацией клавиш
FontNames	Коллекция	Список имен всех доступных шрифтов
Height	Integer	Значение высоты главного окна приложения Word
International	Коллекция	Значения некоторых национальных стандартов для текущей версии ОС
IsObjectValid (object)	Boolean	Имеет значение True, если объект object существует, иначе имеет значение False
KeyBindings	Коллекция	Комбинации клавиш и команды, которые с ними связаны
KeysBoundTo	Объект	Все комбинации клавиш, связанные с определенной командой

Таблица П1.1 *(продолжение)*

Свойство	Тип	Описание
Languages	Коллекция	Языки, которые могут быть использованы в документе
Left	Integer	Значение положения левой границы главного окна Word на рабочем столе
MacroContainer	Объект	Ссылка на объект, ассоциированный с документом или шаблоном, в модуле которого содержится процедура, запущенная на выполнение
MailSystem	Integer	Установленная система обмена почтовыми сообщениями
MAPIAvailable	Boolean	True, если установлен MAPI
MathCoprocessorAvailable	Boolean	True, если установлен математический сопроцессор
MouseAvailable	Boolean	True, если манипулятор "мышь" установлен в системе
Name	String	Обычно содержит текст "Microsoft Word"
NormalTemplate	String	Имя файла, содержащего шаблон Normal
NumLock	Boolean	True — на клавиатуре нажата клавиша <Num Lock>
Options	Объект	Опции приложения MS Word, большинство из которых представляет собой значения различных параметров по умолчанию
Path	String	Рабочий каталог приложения MS Word
PrintPreview	Boolean	True — включен режим просмотра печатной формы документа
RecentFiles	Коллекция	Имена недавно открытых файлов
ScreenUpdating	Boolean	True — окно и элементы окна были обновлены
Selection	Объект	Выделенный объект или диапазон текста в документе
ShowVisualBasicEditor	Boolean	True — окно редактора Visual Basic отображено
StatusBar	String	Строка состояния
System	Объект	Системная информация
Tasks	Коллекция	Объекты, связанные с запущенными на выполнение процессами

Таблица П1.1 *(окончание)*

Свойство	Тип	Описание
Templates	Коллекция	Доступные шаблоны
Top	Integer	Верхняя координата главного окна приложения Word
UsableHeight	Integer	Высота рабочей области главного окна приложения Word
UsableWidth	Integer	Ширина рабочей области главного окна приложения Word
UserAddress	String	Почтовый адрес пользователя
UserControl	Boolean	True — приложение Word или документ были созданы или открыты пользователем, False — приложение Word или документ были созданы или открыты программным путем
UserInitials	String	Инициалы пользователя
UserName	String	Имя пользователя
VBE	Объект	Visual Basic Editor
Version	String	Информация о версии приложения MS Word
Visible	Boolean	True — окно приложения MS Word отображено. Позволяет отобразить/скрыть окно MS Word
Width	Integer	Ширина главного окна приложения Word
Windows	Коллекция	Окна, связанные с приложением MS Word
WindowState	Integer	Состояние главного окна приложения MS Word
WordBasic	Объект	Объект Word Basic

Таблица П1.2. *Методы объекта Application*

Метод	Описание
Activate	Активизирует приложение MS Word
AddAddress	Добавляет запись в адресную книгу
BuildKeyCode	Возвращает код для комбинации клавиш
CentimetersToPoints	Преобразует значение в сантиметрах в количество пикселов

***Таблица П1.2** (продолжение)*

Метод	Описание
ChangeFileOpenDirectory	Устанавливает каталог, в котором MS Word по умолчанию ищет и открывает документы
CheckGrammar (String="Строка текста")	Проверяет грамматику в строке текста
CleanString (String)	Заменяет в строке символы, которые не могут быть напечатаны, символами пробела (код символа пробела — 32)
DDEExecute, DDEInitiate, DDEPoke, DDERequest, DDETerminate, DDETerminateAll	Методы, позволяющие реализовать DDE-интерфейс
GetAddress	Возвращает адрес из адресной книги
GoBack	Перемещает курсор в предыдущее местоположение (где курсор располагался до последнего перемещения)
GoForward	Перемещает курсор вперед (действие, обратное действию, выполняемому методом GoBack)
Help	Открывает справочное окно. Единственный параметр метода определяет индекс блока информации из файла помощи, которую необходимо отобразить в окне
InchesToPoints	Преобразует значение в дюймах в количество пикселов
KeyString (KeyCode, KeyCode2)	Возвращает строку — текстовый эквивалент комбинации клавиш для заданных кодов клавиш
MillimetersToPoints	Преобразует значение в миллиметрах в количество пикселов
Move	Перемещает главное окно приложения MS Word в новые координаты
NewWindow	Создает новое окно для активного документа
OnTime	Запускает на выполнение макрос в заданное время. Параметры метода — значение времени и имя макроса
PointsToCentimeters	Преобразует количество пикселов в значение в сантиметрах
PointsToInches	Преобразует количество пикселов в значение в дюймах

Таблица П1.2 (окончание)

Метод	Описание
PointsToMillimeters	Преобразует количество пикселов в значение в миллиметрах
PrintOut	Печать документа. Параметры метода позволяют задать режим печати
Quit	Закрывает приложение MS Word
Repeat	Повторяет последнее действие редактирования документа
Resize	Изменяет размеры главного окна приложения MS Word
Run	Запускает на выполнение макрос Visual Basic
ScreenRefresh	Обновляет главное окно MS Word в соответствии с новой информацией
SendFax	Посылает факсимильное сообщение

Документы Word

Таблица П1.3. Свойства коллекции Documents

Свойство	Тип	Описание
Count	Integer	Количество открытых документов
Parent	Объект	Ссылка на объект, которому принадлежит коллекция Documents

Таблица П1.4. Методы коллекции Documents

Метод	Описание
Add(Template, NewTemplate)	Создает новый документ. Необязательные параметры: Template — путь и имя используемого шаблона, NewTemplate:=True — создание нового шаблона
Close(SaveChanges, OriginalFormat, RouteDocument)	Закрывает все открытые документы. Необязательные параметры: SaveChanges — порядок сохранения закрываемых документов, OriginalFormat — формат закрываемых документов, используется, если документ имеет вложения

Таблица П1.4 *(окончание)*

Метод	Описание
Item(Index)	Возвращает ссылку на открытый документ. Параметром метода служит номер документа в коллекции или его имя
Open(FileName, ConfirmConversions, ReadOnly, AddToRecentFiles, PasswordDocument, PasswordTemplate, Revert, WritePasswordDocument, WritePasswordTemplate, Format)	Открывает ранее созданный и сохраненный документ. Первый параметр является обязательным (путь и имя открываемого файла). Остальные параметры необязательны и определяют режимы открытия документа
Save(NoPrompt, OriginalFormat)	Сохраняет все открытые документы. Необязательные параметры: NoPrompt:=True — позволяет автоматически сохранить все открытые документы, OriginalFormat — задание формата сохраняемых документов

Таблица П1.5. *Свойства объекта Document*

Свойство	Тип	Описание
ActiveWindow	Объект	Активное окно документа
ActiveWritingStyle	Объект	Активный стиль документа
AttachedTemplate	Объект	Информация о шаблоне, используемом при создании документа
Background	Объект	Свойства фона области документа
Bookmarks	Коллекция	Закладки в документе
Characters	Коллекция	Символы, составляющие текст документа
CommandBars	Коллекция	Панели элементов управления
Comments	Коллекция	Комментарии в документе
Content	Объект	Область, которая представляет главную часть документа
CustomDocumentProperties	Коллекция	Свойства документа (имя, значение и тип свойств), задаваемые пользователем
Fields	Коллекция	Поля. Например, коллекция может содержать дату, которая размещается в тексте и обновляется автоматически
Footnotes	Коллекция	Сноски в документе

***Таблица П1.5** (продолжение)*

Свойство	Тип	Описание
FormFields	Коллекция	Формы в тексте документа
Frames	Коллекция	Рамки в документе
FullName	String	Полный путь и имя файла документа на диске
HasPassword	Boolean	True — требуется пароль, чтобы открыть данный документ
Hyperlinks	Коллекция	Размещенные в документе гиперссылки
InlineShapes	Коллекция	Объекты-формы, которые могут содержать только рисунки или OLE-объекты
IsMasterDocument	Boolean	True — признак главного документа, имеющего один или более подчиненных документов
IsSubdocument	Boolean	True — признак подчиненного документа
Name	String	Имя файла документа
PageSetup	Объект	Параметры страницы
Paragraphs	Коллекция	Абзацы документа
Parent	Объект	Ссылка на объект-хозяин данного документа
Password	String	Пароль для открытия документа
Path	String	Полное путевое имя файла документа
PrintFormsData	Boolean	True — печатать только данные для форм
ProtectionType	Integer	Тип защиты документа
ReadOnly	Boolean	True — документ открыт в режиме "только для чтения" и не может быть сохранен под старым именем
ReadOnlyRecommended	Boolean	True — при открытии документа будет выведен диалог, рекомендующий пользователя открывать данный документ в режиме "только для чтения"
Saved	Boolean	True — с момента последнего сохранения документа в файле изменения в документ не вносились
SaveFormat	Integer	Формат документа
Shapes	Коллекция	Объекты в документе — надписи, рисунки, геометрические фигуры и др.

Таблица П1.5 *(окончание)*

Свойство	Тип	Описание
ShowGrammaticalErrors	Boolean	True — отображать грамматические ошибки в документе
ShowRevisions	Boolean	True — изменения в документе отображаются на экране
TrackRevisions	Boolean	True — изменения в документе отслеживаются
ShowSpellingErrors	Boolean	True — слова с ошибками в документе выделяются подчеркиванием
StoryRanges	Коллекция	Различные области документа
Styles	Коллекция	Текстовые стили, используемые для форматирования документа
Subdocuments	Коллекция	Документы, вложенные в главный документ
Tables	Коллекция	Ссылки на таблицы
Type	Integer	Тип документа: WdTypeDocument(=0) — документ, wdTypeTemplate(=1) — шаблон
UpdateStylesOnOpen	Boolean	True — коллекция стилей обновляется при открытии документа и соответствует стилям шаблона, с помощью которого создан документ
UserControl	Boolean	True — документ был создан или открыт пользователем, False — документ создан или открыт программно, с использованием методов CreateOleObject
Variables	Коллекция	Заданные переменные для использования в вычислениях в модуле документа
VBProject	Коллекция	Проекты, включенные в состав шаблона или документа
Windows	Коллекция	Ссылки на активные окна открытого документа
Words	Коллекция	Ссылки на слова в документе (объекты имеют тип Range)
WritePassword	String	Пароль, используемый для защиты документа при записи его на диск
WriteReserved	Boolean	True — документ был защищен паролем для записи на диск

Таблица П1.6. *Методы объекта Document*

Метод	Описание
Activate	Активизирует документ
AddToFavorites	Добавляет документ в список избранных документов
AutoFormat	Форматирует документ или его часть
CheckGrammar	Проверяет грамматику. Если есть ошибки, то открывается окно для их исправления
CheckSpelling	Проверяет орфографию. Отображает окно для контроля и исправления ошибок. Если заданы параметры, то они определяют дополнительные условия: используемые словари, регистр символов, режим окна
Close	Закрывает документ. Если метод вызывается с параметрами, то они определяют, как будут или не будут сохранены изменения в главном и в подчиненных документах, а также формат документа
ClosePrintPreview	Закрывает окно предварительного просмотра документа
Compare(Name:String)	Сравнивает документ с документом, путь которого содержится в строке Name
ComputeStatistics(Statistic, IncludeFootnotesAndEndnotes)	Возвращает статистические данные для документа: количество символов, линий, страниц, абзацев и т. д.
CopyStylesFromTemplate(Name: String)	Копирует стили из шаблона в документ, Name — путь к файлу шаблона
FollowHyperlink(Address:String)	Загружает HTML-документ в отдельное окно
GoTo(What, Which, Count, Name)	Возвращает ссылку на область, на которую был осуществлен переход. What определяет тип объекта, Which — направление перехода, Count — количество объектов, на которое производится переход, Name — имя объекта, в область которого производится переход (для некоторых типов объектов)
Merge(FileName:String)	Объединяет текущий документ с документом, путь к которому содержится в параметре FileName
Post	Отправляет документ по электронной почте
PrintOut	Отправляет документ на печать. Если используются параметры, то они позволяют задать режимы и области печати

Таблица П1.6 *(окончание)*

Метод	Описание
PrintPreview	Предварительный просмотр печати
Protect(Type, NoReset, Password)	Устанавливает защиту на документ. Аргументы позволяют задать режим защиты и пароль
Range	Возвращает область документа. Если используются аргументы, то они устанавливают ограничение этой области
Redo	Возвращает изменения в документе, которые были отменены
RejectAllRevisions	Отменяет все последние изменения в документе
Reload	Отменяет все последние действия и загружает последнюю версию документа
RunAutoMacro	Выполняет макрокоманды, которые могут быть использованы при открытии, закрытии, сохранении документа и т. д.
Save	Сохраняет документ в файл
SaveAs(FileName:String)	Сохраняет документ в файл с новым именем (имя файла — обязательный аргумент). Могут использоваться дополнительные аргументы, определяющие формат сохраняемого документа, режимы доступа и другие свойства
Select	Выделяет содержимое документа
SendFax(Address, Subject)	Передает документ по факсу, параметры определяют номер факса и тему факсимильного сообщения
SendMail	Создает письмо электронной почты с вложением в это письмо данного документа
Undo	Отменяет последнее изменение документа
UndoClear	Очищает лист последних изменений документа
UnProtect(Password:String)	Отменяет защиту, которая была установлена на документ. В качестве параметра используется строка пароля, заданного при установке защиты на документ
UpdateStyles	Копирует все стили из шаблона в документ

Область Range

Таблица П1.7. *Свойства объекта Range*

Свойство	Тип	Описание
Bold	Boolean	True — в заданной области документа текст имеет полужирное начертание
BookmarkID	Integer	Количество закладок в данной области документа
Bookmarks	Коллекция	Закладки в документе
Borders	Коллекция	Свойства границы области
Case	Integer	Регистр символов для выделенного текста
Characters	Коллекция	Символы, из которых состоит область текста
Comments	Коллекция	Комментарии, входящие в область
End	Integer	Конечная граница области текста
Fields	Коллекция	Поля, входящие в заданную область
Find	Объект	Объект, свойства которого определяют критерии и методы для поиска фрагмента текста в документе
Font	Объект	Свойства шрифта заданной области документа
Footnotes	Коллекция	Сноски в документе
FormattedText	Объект	Объект, который дополнительно к тексту содержит информацию и о формате этой области
FormFields	Коллекция	Формы в заданной области
Frames	Коллекция	Рамки в заданной области
GrammarChecked	Boolean	True — для заданной области была проверена проверка грамматики
GrammaticalErrors	Коллекция	Слова заданной области, которые написаны с грамматическими ошибками
Hyperlinks	Коллекция	Гиперссылки, размещенные в данной области документа
Information	Объект	Различного рода информация о данной области в документе
InlineShapes	Коллекция	Объекты-формы, которые могут содержать только рисунки или OLE-объекты. Все объекты коллекции принадлежат заданной области документа
Italic	Boolean	True — для данной области шрифт имеет курсивное начертание

Таблица П1.7 (окончание)

Свойство	Тип	Описание
LanguageID	Integer	Позволяет определить или установить язык для заданного объекта
ListParagraphs	Коллекция	Список абзацев области
Orientation	Integer	Позволяет определить или задать направление текста в заданной области
PageSetup	Объект	Параметры страницы
ParagraphFormat	Объект	Свойства абзацев, включенных в заданную область
Paragraphs	Коллекция	Абзацы заданной области
Parent	Объект	Ссылка на объект-хозяин объекта Range
PreviousBookmarkID	Integer	Номер закладки, которая находится до заданной области
Start	Integer	Позиция первого символа заданной области
Text	String	Текст данной области
Underline	Integer	Стиль подчеркивания текста
Words	Коллекция	Слова, входящие в данную область

Таблица П1.8. Методы объекта Range

Метод	Описание
AutoFormat	Форматирует заданную область документа
Calculate	Вычисляет результат математического выражения, которое содержится в заданной области
CheckGrammar	Проверяет грамматику в тексте заданной области. Если есть ошибки, то открывается окно для их исправления
CheckSpelling	Проверяет орфографию в тексте заданной области. Если есть ошибки, то открывается окно для их исправления
CheckSynonyms	Показывает диалоговое окно со словами-синонимами для замены слова, которое находится в заданной области
Collapse	Изменяет значения начальной и конечной границ заданной области (после выполнения этого метода эти значения становятся равными друг другу)

Таблица П1.8 *(продолжение)*

Метод	Описание
ComputeStatistics(Statistic, IncludeFootnotesAndEndnotes)	Возвращает статистические данные для документа: количество символов, линий, страниц, абзацов и т. д.
ConvertToTable	Преобразует данную область текста в таблицу
Copy	Копирует данную область в буфер обмена
CopyAsPicture	Копирует данную область в буфер обмена как рисунок
Cut	Вырезает данную область и помещает ее в буфер обмена
Delete	Удаляет заданную область из документа
EndOf	Изменяет конечную границу области, приводя ее к положению, совпадающему с окончанием символа, слова и т. п.
Expand	Расширяет границы заданной области; возвращает величину, на которую произведено расширение
GetSpellingSuggestions	Возвращает коллекцию слов, подходящих для замены первого слова в заданном диапазоне (орфография)
GoTo(What, Which, Count, Name)	Возвращает ссылку на область, на которую был осуществлен переход: What — тип объекта, Which — направление перехода, Count — количество объектов, на которое производится переход, Name — имя объекта, в область которого производится переход (для некоторых типов объектов)
GoToNext	Осуществляет переход на следующий объект и возвращает ссылку на него
GoToPrevious	Осуществляет переход на предыдущий объект и возвращает ссылку на него
InRange(Range)	Проверяет вхождение заданной области в область Range и возвращает значение True, если это условие выполняется
InsertAfter	Вставляет текст после границы заданной области
InsertAutoText	Вставляет текст в режиме "автозамена", если есть подходящий вариант
InsertBefore	Вставляет текст перед границей заданной области
InsertBreak	Вставляет разрыв страницы в заданную область
InsertCaption	Вставляет текст

Таблица П1.8 *(продолжение)*

Метод	Описание
InsertCrossReference	Вставляет ссылки к заголовкам, страницам, сноске и т. п.
InsertDatabase	Вставляет данные из ресурса, который представляет собой рабочую книгу Excel, документ Word, базу данных Access или текстовый файл
InsertDateTime	Вставляет дату и время как текст в заданном формате
InsertFile	Вставляет содержимое файла в заданную область
InsertParagraph	Вставляет новый абзац в область Range документа
InsertParagraphAfter	Вставляет новый абзац после границы заданной области
InsertParagraphBefore	Вставляет новый абзац перед границей заданной области
InsertSymbol	Вставляет символ в заданную область, при этом можно выбрать режим и шрифт вставляемого символа
InStory	Возвращает True, если область находится в том же диапазоне, в котором находится область, указанная в качестве параметра этого метода
IsEqual	Возвращает True, если область, к которой применен этот метод, равна области, указанной в качестве параметра этого метода
LookupNameProperties	Ищет имя в адресной книге и отображает диалог свойств адресата
Move	Изменяет границы области и перемещает указатель в новую позицию
MoveEnd	Перемещает конечную позицию указателя объекта Range или курсора в конец или начало документа на указанное количество объектов (символов, слов, ячеек и т. п.)
MoveEndUntil	Перемещает конечную позицию указателя объекта Range или выделенной области в конец или начало документа до тех пор, пока не встретится заданный объект (символ)
MoveEndWhile	Перемещает конечную позицию указателя объекта Range или выделенной области в конец или начало документа до тех пор, пока встречается заданный символ или объект

Таблица П1.8 *(окончание)*

Метод	Описание
MoveStart	Перемещает начальную позицию указателя объекта Range или курсора в конец или начало документа на указанное количество объектов (символов, слов, ячеек и т. п.)
MoveStartUntil	Перемещает начальную позицию указателя объекта Range или выделенной области в конец или начало документа до тех пор, пока не встретится заданный объект (символ)
MoveStartWhile	Перемещает начальную позицию указателя объекта Range или выделенной области в конец или начало документа до тех пор, пока встречается заданный символ или объект
MoveUntil	Перемещает позицию объекта Range или выделения в области до тех пор, пока заданная комбинация символов не будет найдена. Возвращает фактическое количество символов, на которое было произведено перемещение
MoveWhile	Перемещает позицию объекта Range или выделения на заданное количество символов, слов и т. п. в области до тех пор, пока встречается заданная комбинация символов. Возвращает фактическое количество символов, на которое было произведено перемещение
Next	Перемещает позицию объекта Range или выделения к следующему символу, слову и т. п.
Paste	Вставляет текст из буфера обмена в заданную область
PasteSpecial	Вставляет текст из буфера обмена в заданную область. В отличие от метода Paste, данный метод позволяет управлять форматом вставляемого текста
Previous	Возвращает область, связанную с символом, словом и т. п., предшествующим заданной области
Select	Выделяет заданную область
SetRange	Устанавливает начальную и конечную позиции заданной области
Sort	Сортирует текст в заданной области
StartOf	Перемещает указатели объекта Range или выделения в начало заданной области
WholeStory	Расширяет границы заданной области

Область Selection

Таблица П1.9. *Свойства объекта Selection*

Свойство	Тип	Описание
Cells	Коллекция	Ячейки таблицы, попадающие в выделенную область
Columns	Коллекция	Столбцы таблицы, попадающие в выделенную область
ColumnSelectMode	Boolean	True — режим выделения столбца
Document	Объект	Ссылка на документ, которому принадлежит выделенная область
ExtendMode	Boolean	True — режим, позволяющий расширять границы выделенной области
Flags	Integer	Свойства выделенной области (объекта), выражаемые суммой определенных констант
Rows	Коллекция	Строки таблицы, попадающие в выделенную область
Shading	Объект	Свойства штриховки выделенной области
Start	Integer	Позиция первого символа заданной области
StartIsActive	Boolean	True — активно начало выделенной области, например, если правая часть области текста зафиксирована и область изменяет свои границы за счет изменения левой границы
Tables	Коллекция	Таблицы, попадающие в выделенную область
Text	String	Текст выделенной области
Type	Integer	Тип выделенного объекта — таблица, окно и т. п.
Words	Коллекция	Слова текста, попадающие в выделенную область

Таблица П1.10. *Методы объекта Selection*

Метод	Описание
Copy	Копирует выделенную область в буфер обмена
CopyAsPicture	Копирует выделенную область в буфер обмена как рисунок
CopyFormat	Копирует выделенную область и формат, в котором представлена информация в этой области, в буфер обмена

Таблица П1.10 *(продолжение)*

Метод	Описание
CreateTextbox	Создает надпись в месте выделенной области, перенося ее в область объекта
Cut	Вырезает текст (объект) и помещает его в буфер обмена
Delete	Удаляет выделенный объект или область
EndKey	Расширяет выделенную область к концу указанной единицы. Возвращает величину фактического перемещения (в символах)
EscapeKey	Отменяет установленный режим задания границ выделенной области
Extend	Расширяет выделенную область до положения символа, указанного в качестве аргумента метода
MoveDown	Перемещает конечную позицию указателя объекта Selection вниз и возвращает величину, на которую реально было выполнено перемещение
MoveEnd	Перемещает конечную позицию указателя объекта Selection в конец или начало документа на указанное количество объектов (символов, слов, ячеек и т. п.)
MoveLeft	Перемещает положение указателя объекта Selection влево и возвращает величину, на которую реально было выполнено перемещение
MoveRight	Перемещает положение указателя объекта Selection вправо и возвращает величину, на которую реально было выполнено перемещение
MoveUp	Перемещает начальную позицию указателя объекта Selection вверх и возвращает величину, на которую реально было выполнено перемещение
Paste	Вставляет текст из буфера обмена в выделенную область
PasteFormat	Вставляет текст из буфера обмена в выделенную область с учетом формата
SelectColumn	Выделяет столбец таблицы, в котором находится курсор
SelectCurrentAlignment	Расширяет выделенную область до тех пор, пока не встретится текст с другим выравниванием
SelectCurrentColor	Расширяет выделенную область до тех пор, пока не встретится текст с другим цветом
SelectCurrentFont	Расширяет выделенную область до тех пор, пока не встретится текст с другим шрифтом

Таблица П1.10 (окончание)

Метод	Описание
SelectCurrentSpacing	Расширяет выделенную область до тех пор, пока не встретится текст с другим межбуквенным интервалом
SelectCurrentTabs	Расширяет выделенную область до тех пор, пока не встретится абзац с другим отступом
SelectRow	Выделяет строку таблицы, в которой находится курсор
TypeBackspace	Удаляет выделенную область или символ слева от курсора (как по нажатию клавиши <Backspace>)

Шрифт, свойства и методы

Таблица П1.11. Свойства объекта Font

Свойство	Тип	Описание
AllCaps	Boolean	True — все буквы прописные
Animation	Integer	Вид анимации символов
Bold	Boolean	True — полужирное начертание символов
ColorIndex	Integer	Индекс цвета из цветовой палитры
DoubleStrikeThrough	Boolean	True — к символам применяется эффект двойного перечеркивания
Duplicate	Объект	Свойства шрифта, используемые для их дублирования применительно к набору символов
Emboss	Boolean	True — приподнятые символы
Engrave	Boolean	True — утопленные символы
Hidden	Boolean	True — символы шрифта скрыты
Italic	Boolean	True — курсивное начертание символов
Kerning	Extended	Величина кернинга
Name	String	Имя шрифта
NameAscii	String	Имя шрифта, используемое для символов с кодами от 0 до 127
NameOther	String	Имя шрифта, используемое для символов с кодами больше 127
Outline	Boolean	True — отображение символов в виде контура

Таблица П1.11 *(окончание)*

Свойство	Тип	Описание
Parent	Объект	Объект-хозяин, свойством которого является объект Font
Position	Extended	Величина вертикального смещения
Scaling	Extended	Величина масштаба отображения символов
Shading	Объект	Свойства фона символов
Shadow	Boolean	True — символы отображаются с тенью
Size	Integer	Размер шрифта
SmallCaps	Boolean	True — малые прописные буквы
Spacing	Extended	Расстояние между символами
StrikeThrough	Boolean	True — зачеркнутые символы
Subscript	Boolean	True — символы в виде нижнего индекса
Superscript	Boolean	True — символы в виде верхнего индекса
Underline	Integer	Стиль подчеркивания

Таблица П1.12. *Методы объекта Font*

Метод	Описание
Grow	Увеличивает размер шрифта на минимальную доступную величину
Reset	Отменяет пользовательские настройки шрифта
SetAsTemplateDefault	Устанавливает данный шрифт как шрифт по умолчанию для этого документа и для всех новых документов
Shrink	Уменьшает размер шрифта на минимальную доступную величину

Коллекция таблиц, свойства и методы

Таблица П1.13. *Свойства коллекции Tables*

Свойство	Тип	Описание
Count	Integer	Количество таблиц коллекции
Parent	Объект	Ссылка на объект, которому принадлежит коллекция Tables

Таблица П1.14. *Методы коллекции Tables*

Метод	Описание
Add	Добавляет новую таблицу в коллекцию
Item	Возвращает ссылку на таблицу коллекции

Таблица, свойства и методы

Таблица П1.15. *Свойства объекта Table*

Свойство	Тип	Описание
AutoFormatType	Integer	Формат таблицы
Borders	Коллекция	Свойства линий сетки таблицы
Columns	Коллекция	Столбцы таблицы
Parent	Объект	Ссылка на объект, которому принадлежит таблица
Range	Объект	Область таблицы
Rows	Коллекция	Строки таблицы
Shading	Объект	Свойства штриховки области таблицы
Uniform	Boolean	True — все строки таблицы имеют одинаковое количество столбцов

Таблица П1.16. *Методы объекта Table*

Метод	Описание
AutoFormat	Форматирует таблицу. Параметры метода позволяют задать формат, шрифт, цвет, стиль линий сетки и другие свойства таблицы и текста
Cell	Возвращает объект, представляющий собой ячейку таблицы
ConvertToText	Преобразует таблицу в текст и возвращает объект Range как ссылку на область, которая содержит этот текст
Delete	Удаляет таблицу
Select	Выделяет таблицу
Sort	Сортирует информацию в таблице. Параметры метода позволяют задать способы и условия сортировки
SortAscending	Сортирует информацию в таблице по возрастанию

Таблица П1.16 *(окончание)*

Метод	Описание
SortDescending	Сортирует информацию в таблице по убыванию
Split	Разрывает таблицу (вставляет пустую строку) в месте таблицы, задаваемом номером строки
UpdateAutoFormat	Восстанавливает формат таблицы, приводя его к стандартному формату, который был использован при создании таблицы

Коллекция объектов Shapes, свойства и методы

Таблица П1.17. *Свойства коллекции Shapes*

Свойство	Тип	Описание
Count	Integer	Количество элементов коллекции
Parent	Объект	Ссылка на объект, которому принадлежит коллекция Shapes

Таблица П1.18. *Методы коллекции Shapes*

Метод	Описание
AddCallout	Создает объект-выноску. Параметры метода позволяют определить тип объекта, его координаты и размеры. Возвращает ссылку на созданный объект
AddCurve	Создает кривую линию. Параметры метода несут информацию о массиве координат точек создаваемой линии. Возвращает ссылку на созданный объект
AddLabel	Создает метку. Параметры метода определяют ориентацию текста надписи, координаты ее местонахождения и размеры. Возвращает ссылку на созданный объект
AddLine	Создает линию. Параметры метода определяют начальные и конечные координаты линии. Возвращает ссылку на созданный объект
AddOLEControl	Создает элемент управления ActiveX. Параметры метода определяют класс объекта, его координаты и размеры
AddOLEObject	Создает OLE-объект. Параметры метода определяют класс объекта, файл, откуда он загружается, координаты объекта, размеры и способ отображения
AddPicture	Создает рисунок, загружаемый из графического файла. Параметры метода определяют координаты, размеры и способ отображения и сохранения изменений рисунка

Таблица П1.18 (окончание)

Метод	Описание
AddPolyline	Создает полилинию (линию, состоящую из прямых и кривых участков). Параметры метода несут информацию о массиве координат точек создаваемой полилинии. Возвращает ссылку на созданный объект
AddShape	Создает автофигуру. Параметры метода определяют вид создаваемой фигуры, ее координаты и размеры
AddTextbox	Создает надпись. Параметры метода определяют ориентацию текста надписи, координаты ее местонахождения и размеры. Возвращает ссылку на созданный объект
AddTextEffect	Создает объект WordArt. Параметры метода определяют стиль создаваемого объекта, текст, параметры шрифта и координаты
BuildFreeform	Создает объект, позволяющий описать и создать в документе произвольную фигуру, состоящую из прямых и кривых линий
Item	Предоставляет доступ к элементам коллекции Shapes
Range	Возвращает ссылку на область одного из элементов коллекции. Параметром служит порядковый номер этого элемента в коллекции Shapes
SelectAll	Выделяет все элементы коллекции

Объект Shape, свойства и методы

Таблица П1.19. Свойства объекта Shape

Свойство	Тип	Описание
Adjustments	Коллекция	Точки, позволяющие настраивать вид графической фигуры. Элементы коллекции представляют собой значения типа Extended
Anchor	Объект типа Range	Определяет связь объекта с областью текста страницы документа (якорь)
AutoShapeType	Integer	Тип объекта Shape (автофигуры)
Callout	Объект	Обеспечивает связь графического элемента с элементом — выноской
Fill	Объект	Свойства заливки графического объекта
GroupItems	Коллекция	Свойства сгруппированных объектов
Height	Integer	Высота объекта

Таблица П1.19 *(продолжение)*

Свойство	Тип	Описание
HorizontalFlip	Boolean	True (–1) — объект был зеркально отражен в горизонтальной плоскости
Hyperlink	Объект	Гиперссылка
Left	Extended	Левая координата объекта
Line	Объект	Свойства линий границы объекта или самой линии для объекта Line
LinkFormat	Объект	Свойства ресурсов, используемых при создании этого объекта, и действия над ними
LockAnchor	Boolean	True — объект связан с областью Range и заблокирован. Якорь не может быть перемещен и указывает на то место документа, где будет отображен объект
LockAspectRatio	Boolean	True — объект автоматически пропорционально изменяет все свои размеры при изменении хотя бы одной характеристики размера
Name	String	Имя объекта
Nodes	Коллекция	Геометрическое описание объекта
OLEFormat	Объект	OLE-объект, заключенный в объект Shape, и его свойства
Parent	Объект	Ссылка на объект, которому принадлежит объект Shape
PictureFormat	Объект	Рисунок, заключенный в объект Shape, и его свойства
RelativeHorizontalPosition	Integer	Определяет, относительно чего будет задано горизонтальное положение (выравнивание) объекта (например, относительно страницы, колонки, края текста)
RelativeVerticalPosition	Integer	Определяет, относительно чего будет задано вертикальное положение (выравнивание) объекта (например, относительно края текста, страницы, абзаца)
Rotation	Integer	Угол поворота фигуры
Shadow	Boolean	True — объект отображен с тенью
TextEffect	Объект	Объект WordArt, заключенный в объекте Shape, и его свойства
TextFrame	Объект	Текст, заключенный в объект Shape, и его свойства

Таблица П1.19 (окончание)

Свойство	Тип	Описание
ThreeD	Объект	3D-объект, заключенный в объект Shape, и его свойства
Top	Integer	Верхняя координата расположения объекта Shape
Type	Integer	Тип объекта Shape
VerticalFlip	Boolean	True (–1) — объект зеркально отражен в вертикальной плоскости
Vertices	Коллекция	Координаты точек кривых и ломаных линий
Visible	Boolean	True — объект отображен, иначе False
Width	Integer	Ширина объекта
WrapFormat	Объект	Способ и параметры обтекания текста вокруг объекта Shape
ZOrderPosition	Integer	Порядок размещения данного объекта Shape относительно других объектов Shape (на переднем или заднем плане относительно друг друга)

Таблица П1.20. Методы объекта Shape

Метод	Описание
Activate	Активизирует объект
Apply	Применяет к объекту формат, скопированный у другого подобного объекта с помощью метода PickUp
ConvertToFrame	Преобразует объект в рамку. Возвращает ссылку на новый объект
ConvertToInlineShape	Конвертирует рисунок в объект InlineShape — объект, который при редактировании документа ведет себя как символ текста
Delete	Удаляет объект из документа
Duplicate	Создает копию объекта
Flip	Зеркально отражает объект
IncrementLeft	Перемещает объект по горизонтали на заданное количество точек
IncrementRotation	Изменяет угол поворота объекта на заданное количество градусов

Таблица П1.20 *(окончание)*

Метод	Описание
IncrementTop	Перемещает объект по вертикали на заданное количество точек
PickUp	Копирует формат объекта
ScaleHeight	Задает масштаб изменения высоты объекта. Параметры метода задают величину и условия применения масштаба
ScaleWidth	Задает масштаб изменения ширины объекта. Параметры метода задают величину и условия применения масштаба
Select	Выделяет объект
SetShapesDefaultProperties	Сохраняет и использует свойства объекта как свойства по умолчанию, которые будут использоваться при создании новых объектов
Ungroup	Разгруппировывает сгруппированные объекты Shape
ZOrder	Изменяет порядок размещения объекта относительно других объектов (на переднем или заднем плане относительно друг друга)

Приложение MS Excel

Таблица П1.21. *Свойства объекта Application*

Свойство	Тип	Описание
ActiveCell	Объект	Ссылка на активную ячейку (имеющую фокус ввода)
ActiveChart	Объект	Ссылка на активную диаграмму
ActivePrinter	Объект	Свойства активного принтера
ActiveSheet	Объект	Ссылка на активный лист
ActiveWindow	Объект	Ссылка на активное окно
ActiveWorkbook	Объект	Ссылка на активную рабочую книгу
AddIns	Коллекция	Библиотека шаблонов стилей документов
Assistant	Объект	Ссылка на Помощник (анимированный элемент интерфейса)
AutoCorrect	Объект	Информация для автозамены слов в тексте

Таблица П1.21 *(продолжение)*

Свойство	Тип	Описание
Build	String	Информация о версии MS Excel
CalculateBeforeSave	Boolean	True — информация в ячейках рабочих книг рассчитывается перед тем, как они будут сохранены в файле
Calculation	Integer	Режим вычисления формул на листе — автоматическое или неавтоматическое (ручной режим)
Caption	String	Заголовок окна
CellDragAndDrop	Boolean	True — включен режим перетаскивания
Cells	Объект	Ссылка на область ячеек активной рабочей книги
Charts	Коллекция	Диаграммы
ClipboardFormats	Integer	Формат буфера обмена
Columns	Коллекция	Столбцы листа рабочей книги
CommandBars	Коллекция	Панели элементов управления приложения Excel
Cursor	Integer	Вид курсора
CutCopyMode	Integer	Статус команд **Вырезать** и **Копировать**
DefaultFilePath	String	Путь по умолчанию, используемый для открытия файлов
DefaultSaveFormat	Integer	Формат рабочих книг, по умолчанию используемый для сохранения файлов
Dialogs	Коллекция	Диалоги
DisplayAlerts	Boolean	False — отключить сообщения, генерируемые во время выполнения макросов
DisplayCommentIndicator	Integer	Режим отображения индикаторов и (или) примечаний для ячеек таблицы
DisplayFormulaBar	Boolean	False — строка формул не отображается
DisplayFullScreen	Boolean	True — полноэкранный режим отображения таблицы рабочей книги
DisplayNoteIndicator	Boolean	True — отображение комментариев
DisplayRecentFiles	Boolean	True — в меню **Файл** отображается список недавно открытых файлов

Таблица П1.21 *(продолжение)*

Свойство	Тип	Описание
DisplayScrollBars	Boolean	True — полосы прокрутки отображаются, False — скрыты
DisplayStatusBar	Boolean	True — отображается строка состояния
EditDirectlyInCell	Boolean	True — допускается правка информации прямо в ячейке
EnableEvents	Boolean	True — допускается обработка откликов объекта на различные события
EnableSound	Boolean	Включение/отключение звуковых эффектов в Excel
FileConverters	Коллекция	Файлы-конверторы
FileSearch	Объект	Используется для поиска файлов
FixedDecimal	Boolean	True — все вводимые числовые данные будут автоматически приводиться к формату с фиксированной точкой. Количество десятичных знаков определяется свойством FixedDecimalPlaces
FixedDecimalPlaces	Integer	Определяет количество знаков для формата с фиксированной точкой (см. свойство FixedDecimal)
Height	Integer	Высота главного окна приложения Excel
IgnoreRemoteRequests	Boolean	True — запросы DDE внешних программ игнорируются
Interactive	Boolean	False — блокируется ввод с клавиатуры и от мыши
International	Коллекция	Значения некоторых национальных стандартов для текущей версии ОС
Left	Integer	Положение левой границы главного окна Excel на рабочем столе
LibraryPath	String	Путь к файлам библиотек
MailSystem	Integer	Информация об установленной системе обмена почтовыми сообщениями
MathCoprocessorAvailable	Boolean	True — установлен математический сопроцессор
MemoryFree	Integer	Объем свободной доступной приложению Excel оперативной памяти (в байтах)
MemoryTotal	Integer	Объем всей доступной приложению Excel оперативной памяти (в байтах)

Таблица П1.21 (продолжение)

Свойство	Тип	Описание
MemoryUsed	Integer	Объем используемой приложением Excel оперативной памяти (в байтах)
MouseAvailable	Boolean	True — манипулятор "мышь" установлен в системе
MoveAfterReturn	Boolean	Управление перемещением фокуса ввода к новой ячейке по нажатию клавиши <Enter>
MoveAfterReturnDirection	Integer	Направление перемещения фокуса ввода к новой ячейке (см. свойство MoveAfterReturn)
Name	String	Обычно содержит текст "Microsoft Excel"
Names	Коллекция	Имена объектов всех рабочих книг приложения
NetworkTemplatesPath	String	Сетевой путь к папке, содержащей шаблоны документов
OnWindow	String	Имя процедуры, выполняемой при активации окна
OperatingSystem	String	Наименование и номер установленной операционной системы
OrganizationName	String	Наименование организации, зарегистрировавшей данное приложение MS Excel
Path	String	Путь к рабочей папке приложения MS Excel
PathSeparator	Char	Символ-разделитель для строки, содержащей путь к рабочей папке приложения MS Excel (см свойство Path)
PromptForSummaryInfo	Boolean	True — во время сохранения рабочей книги в файле выводится окно для редактирования ее свойств
Range	Объект	Ссылка на область ячеек рабочей книги
RecentFiles	Коллекция	Имена недавно открытых файлов
ReferenceStyle	Integer	Стиль заголовков столбцов таблицы
RollZoom	Boolean	True — включить режим управления масштабом отображения рабочей книги с помощью колеса мыши
Rows	Коллекция	Строки таблицы рабочей книги
ScreenUpdating	Boolean	True — окно и элементы управления окна были обновлены

Таблица П1.21 *(окончание)*

Свойство	Тип	Описание
Selection	Объект	Выделенный объект или диапазон ячеек в рабочей книге
Sheets	Коллекция	Листы активной рабочей книги
SheetsInNewWorkbook	Integer	Количество листов рабочей книги, создаваемых при создании новой рабочей книги
StandardFont	String	Наименование стандартного шрифта
StandardFontSize	Integer	Размер стандартного шрифта
StartupPath	String	Полный путь к папке запуска Excel
StatusBar	String	Строка состояния
TemplatesPath	String	Папка с шаблонами документов
ThisWorkbook	Объект	Ссылка на рабочую книгу, макрос из программного модуля которой выполняется в настоящее время
Top	Integer	Верхняя координата главного окна приложения Excel
UsableHeight	Integer	Высота рабочей области главного окна приложения Excel
UsableWidth	Integer	Ширина рабочей области главного окна приложения Excel
UserControl	Boolean	True — приложение Excel или рабочая книга были созданы или открыты пользователем, False — приложение Excel или рабочая книга были созданы или открыты программным путем
UserName	String	Имя пользователя
Value	String	Всегда содержит строку "Microsoft Excel"
VBE	Объект	Объект Visual Basic Editor
Version	String	Информация о версии приложения Excel
Visible	Boolean	Позволяет отобразить/скрыть главное окно приложения Excel (True — окно отображено)
Width	Integer	Ширина главного окна приложения Excel
Windows	Коллекция	Окна, связанные с приложением Excel
WindowState	Integer	Состояние главного окна приложения Excel
Workbooks	Коллекция	Открытые рабочие книги
Worksheets	Коллекция	Листы открытых рабочих книг

Таблица П1.22. *Методы объекта Application*

Метод	Описание
ActivateMicrosoftApp	Активирует или запускает, если приложение не было активно, одно из приложений MS Office. Параметр метода — числовая константа, ассоциированная с одним из этих приложений
AddChartAutoFormat	Добавляет в коллекцию форматов диаграмм новый формат, используя для этого готовую диаграмму
Calculate	Рассчитывает все формулы во всех открытых рабочих книгах
CentimetersToPoints	Преобразует значение в сантиметрах в количество точек
CheckSpelling	Проверяет орфографию. Отображает окно для контроля и исправления ошибок. Если заданы параметры, то они определяют дополнительные условия: используемые словари, регистр символов, режим окна
ConvertFormula	Конвертирует формулы, которые содержат адреса ячеек в стиле R1C1, в идентичные формулы, содержащие адреса в стиле A1
DDEExecute, DDEInitiate, DDEPoke, DDERequest, DDETerminate	Позволяют реализовать DDE-интерфейс
DeleteChartAutoFormat	Удаляет формат диаграммы, заданный названием, из списка форматов диаграмм, используемых при создании новой диаграммы
DoubleClick	Действие метода эквивалентно двойному щелчку левой кнопки мыши
Evaluate	Преобразует имя объекта в ссылку на этот объект. Параметром метода является строка, содержащая имя объекта
FindFile	Открывает диалог **Открытие документа**
GetOpenFilename	Открывает диалог **Открытие документа**, затем возвращает результат выполнения диалога: строку (путь и имя открываемого файла) или False, если пользователь отменил открытие документа
GetSaveAsFilename	Открывает диалог **Сохранение документа**, затем возвращает результат выполнения диалога: строку (путь и имя сохраняемого файла) или False, если пользователь отменил сохранение документа
Goto	Выполняет переход к объекту, заданному параметром метода

Таблица П1.22 *(окончание)*

Метод	Описание
Help	Открывает справочное окно. Первый параметр метода указывает на файл помощи, второй определяет индекс блока информации из файла помощи, который требуется отобразить в окне
InchesToPoints	Преобразует значение в дюймах в количество пикселов
InputBox	Открывает диалог для ввода информации. Возвращает значение, введенное пользователем. Параметры метода задают свойства диалога
Intersect	Возвращает объект, представляющий собой пересечение областей, адреса которых заданы аргументами метода
MacroOptions	Позволяет задать для макроса параметры, например, сочетание клавиш и его описание. См. диалог **Параметры макроса**
OnKey	Позволяет задать ключевую комбинацию для выполнения заданных процедур. Параметрами метода являются описание ключевой комбинации и наименование выполняемой процедуры
OnTime	Предписывает запустить на выполнение макрос в заданное время. Параметрами метода являются значение времени и имя макроса
Quit	Закрывает приложение MS Excel
Repeat	Повторяет последнее действие редактирования документа
Run	Запускает на выполнение макрос Visual Basic
SaveWorkspace	Сохраняет рабочую область документа
SendKeys	Действие метода эквивалентно нажатию клавиши или комбинации клавиш, коды которых заданы первыми параметрами метода, последний параметр метода — время задержки выполнения
SetDefaultChart	Определяет наименование шаблона диаграммы Excel, который будет использоваться при создании новых диаграмм
Undo	Отменяет последнее изменение документа
Union	Возвращает объект, представляющий собой объединение двух и более областей Range
Wait	Позволяет задержать выполнение макросов Excel

Рабочая книга Excel

Таблица П1.23. *Свойства коллекции Workbooks*

Свойство	Тип	Описание
Count	Integer	Количество открытых рабочих книг
Item	Объект	Ссылка на открытую рабочую книгу. Параметром метода служит номер документа в коллекции или его имя
Parent	Объект	Ссылка на объект, которому принадлежит коллекция Workbooks

Таблица П1.24. *Методы коллекции Workbooks*

Метод	Описание
Add(Template)	Создает новый документ. Необязательный параметр Template — путь и имя используемого шаблона
Close	Закрывает все открытые документы
Open(FileName, UpdateLinks, ReadOnly, Format, Password, WriteResPassword, IgnoreReadOnlyRecommended, Origin, Delimiter, Editable, Notify, Converter, AddToMRU)	Открывает созданную ранее и сохраненную рабочую книгу. Первый параметр является обязательным и указывает путь и имя открываемого файла. Остальные параметры необязательны и определяют режимы открытия документа
OpenText (Filename, Origin, StartRow, DataType, TextQualifier, ConsecutiveDelimiter, Tab, Semicolon, Comma, Space, Other, OtherChar, FieldInfo)	Открывает текстовый файл. Первый параметр является обязательным и указывает путь и имя открываемого файла. Остальные параметры необязательны и определяют режимы открытия документа

Таблица П1.25. *Свойства объекта Workbook*

Свойство	Тип	Описание
ActiveChart	Объект	Ссылка на активную диаграмму
ActiveSheet	Объект	Ссылка на активный лист рабочей книги
AutoUpdateFrequency	Integer	Период обновления документа в минутах (для документов с общим доступом)
AutoUpdateSaveChanges	Boolean	True — обновлять документ, если он был сохранен (для документов с общим доступом)

Таблица П1.25 *(продолжение)*

Свойство	Тип	Описание
BuiltinDocumentProperties	Объект	Свойства документа (см. диалог, открываемый командой **Свойства** меню **Файл**
ChangeHistoryDuration	Integer	Количество дней, в течение которых необходимо хранить журнал изменений документа (для документов с общим доступом)
Charts	Коллекция	Диаграммы рабочей книги
CodeName	String	Кодовое название (обозначение) рабочей книги. Обозначение может подставляться в макросах вместо описателя объекта
Colors	Коллекция	Цветовая палитра для данной рабочей книги
CommandBars	Коллекция	Панели управления
ConflictResolution	Integer	Способ решения конфликтов при совместном доступе к документу
CreateBackup	Boolean	True — создается и используется резервная копия рабочей книги
CustomDocumentProperties	Объект	Свойства документа (имя, значение и тип свойств), которые задает пользователь
Date1904	Boolean	True — используется система дат, применяемая до 1904 года
DisplayDrawingObjects	Integer	Режим отображения графических объектов в документе
FileFormat	Integer	Формат файла загруженной рабочей книги
FullName	String	Полный путь и имя файла документа
HasPassword	Boolean	True — рабочая книга защищена паролем
KeepChangeHistory	Boolean	True — приложение Excel позволяет сохранять журнал изменений для рабочей книги с раздельным доступом
Mailer	Объект	Электронное письмо, его свойства и методы (текст, адрес и др.)
MultiUserEditing	Boolean	True — рабочая книга открыта в режиме раздельного доступа для нескольких пользователей
Name	String	Имя файла, в котором хранится документ
Names	Коллекция	Имена объектов активной рабочей книги
Path	String	Полный путь файла документа

Таблица П1.25 *(окончание)*

Свойство	Тип	Описание
PrecisionAsDisplayed	Boolean	True — точность числовых значений такая, как отображена в ячейках рабочей книги
ProtectStructure	Boolean	True — структура книги защищена от изменений
ProtectWindows	Boolean	True — окна рабочей книги защищены от изменений
ReadOnly	Boolean	True — документ открыт в режиме "только для чтения" и не может быть сохранен
ReadOnlyRecommended	Boolean	True — при открытии документа выводится диалог, рекомендующий пользователю открывать документ в режиме "только для чтения"
Saved	Boolean	True — после последнего сохранения документа в файле документ не был изменен
SaveLinkValues	Boolean	True — сохранение внешних связей в активном документе
Sheets	Коллекция	Листы рабочей книги
Styles	Коллекция	Стили, используемые для формирования рабочей книги
TemplateRemoveExtData	Boolean	True — удаление внешних связей при сохранении шаблонов
UpdateRemoteReferences	Boolean	True — обновление внешних связей в рабочей книге
UserStatus	Массив	Информация о пользователях, открывших рабочую книгу
VBProject	Коллекция	Проекты, включенные в состав шаблона или документа
Windows	Коллекция	Ссылки на активные окна открытой рабочей книги
Worksheets	Коллекция	Обеспечивает доступ к листам рабочей книги
WriteReserved	Boolean	True — документ защищен паролем для записи на диск
WriteReservedBy	String	Имя пользователя, который в настоящее время редактирует рабочую книгу

Таблица П1.26. *Методы объекта Workbook*

Метод	Описание
Activate	Активирует первое окно, связанное с документом
AddToFavorites	Добавляет документ в список избранных документов
Close (SaveChanges, FileName, RouteWorkbook)	Закрывает рабочую книгу. Необязательные параметры: SaveChanges — определяет, сохранять документ или нет, FileName — имя файла, в котором сохраняется документ, RouteWorkbook — позволяет отправить документ
DeleteNumberFormat	Удаляет числовой формат из рабочей книги. Параметром метода является строка, соответствующая удаляемому формату
ExclusiveAccess	Позволяет дать текущему пользователю исключительный доступ
NewWindow	Создает новое окно для заданной рабочей книги
Post	Отправляет по почте документ
PrintOut	Отправляет документ на печать. Параметры метода позволяют задать режим печати
PrintPreview	Открывает окно предварительного просмотра печати
Protect (Password, Structure, Windows)	Устанавливает защиту на документ. Параметры позволяют задать пароль и степень защиты
PurgeChangeHistoryNow	Удаляет все изменения в документе, сделанные ранее
RejectAllChanges	Отклоняет все изменения в разделенном документе
RemoveUser	Отключает удаленного пользователя от рабочей книги
ResetColors	Устанавливает цветовую палитру в состояние по умолчанию
RunAutoMacros	Запускает макрокоманды, выполняющиеся при открытии, закрытии, активации и деактивации рабочей книги
Save	Сохраняет документ в файле
SaveAs (FileName:String)	Сохраняет документ в файле с новым именем (имя файла — обязательный аргумент). Могут использоваться дополнительные аргументы, определяющие формат сохраняемого документа, режимы доступа и другие свойства
SaveCopyAs	Записывает в файл копию рабочей книги, не изменяя ее
SendMail	Создает письмо электронной почты с вложением в это письмо данного документа
Unprotect (Password:String)	Отменяет защиту, установленную для документа. В качестве параметра используется строка пароля, заданного при установке защиты на документ

Таблица П1.26 (окончание)

Метод	Описание
UnprotectSharing (Password:String)	Отменяет защиту для разделяемого документа и сохраняет его на диске
UpdateFromFile	Обновляет документ, читая его из файла на диске
UpdateLink	Обновляет связи документа MS Excel с внешними объектами

Лист рабочей книги Excel

Таблица П1.27. Свойства коллекций Worksheets, Sheets

Свойство	Тип	Описание
Count	Integer	Количество элементов коллекции Sheets рабочей книги
HpageBreaks	Коллекция	Горизонтальные разрывы листа на страницы
Item	Объект	Ссылка на лист рабочей книги. Параметром метода служит номер элемента в коллекции или имя листа рабочей книги
Parent	Объект	Ссылка на объект, которому принадлежит коллекция Workbooks
VPageBreaks	Коллекция	Вертикальные разрывы листа на страницы

Таблица П1.28. Методы коллекций Worksheets, Sheets

Метод	Описание
Add	Создает новый лист. Необязательные параметры определяют положение создаваемого листа, количество и тип листов
Copy	Создает копии листов рабочей книги. Параметры метода определяют местоположение созданных копий
FillAcrossSheets	Копирует область, заданную аргументом метода, на такие же области других рабочих листов книги
Move	Перемещает выбранные листы в новое место, указанное в качестве аргумента метода
PrintOut	Осуществляет вывод на печать
PrintPreview	Открывает окно предварительного просмотра печати
Select	Выделяет листы рабочей книги

Таблица П1.29. *Свойства объекта Worksheet*

Свойство	Тип	Описание
Cells	Коллекция	Ячейки листа рабочей книги
CodeName	String	Кодовое название (обозначение) листа рабочей книги. Обозначение может подставляться в макросах вместо объекта-описателя
Columns	Коллекция	Столбцы листа рабочей книги
Comments	Коллекция	Список комментариев листа
ConsolidationFunction	Integer	Информация о функции, используемой для консолидации
ConsolidationOptions	Boolean	Информация об опциях консолидации
ConsolidationSources	String	Адрес исходных данных для консолидации
DisplayPageBreaks	Boolean	True — отображаются линии разделения листа на печатные страницы
EnableCalculation	Boolean	True — приложение Excel автоматически пересчитывает лист тогда, когда это требуется
HPageBreaks	Коллекция	Горизонтальные разрывы листа на страницы
Hyperlinks	Коллекция	Гиперссылки
Index	Integer	Номер (индекс) листа в коллекции Worksheets
Name	String	Имя листа рабочей книги
Names	Коллекция	Имена объектов листа
Next	Объект	Ссылка на следующий лист рабочей книги
PageSetup	Объект	Параметры страницы
Previous	Объект	Ссылка на предыдущий лист рабочей книги
ProtectContents	Boolean	True — содержимое листа защищено
ProtectDrawingObjects	Boolean	True — объекты листа защищены
ProtectScenarios	Boolean	True — сценарии защищены
Range	Объект	Ссылка на область ячеек листа рабочей книги
Rows	Коллекция	Строки таблицы листа рабочей книги
ScrollArea	String	Диапазон адресов, доступный пользователю для просмотра, ввода/вывода информации ячеек
Shapes	Коллекция	Надписи, рисунки, геометрические фигуры и другие фигуры на листе рабочей книги Excel

Таблица П1.29 *(окончание)*

Свойство	Тип	Описание
StandardHeight	Extended	Стандартная высота строк таблицы, используемая по умолчанию
StandardWidth	Extended	Стандартная ширина столбцов таблицы, используемая по умолчанию
Type	Integer	Тип листа рабочей книги
UsedRange	Объект	Ссылка на заполненную область листа рабочей книги
Visible	Boolean	Позволяет отобразить/скрыть лист рабочей книги (True — лист отображен)
VPageBreaks	Коллекция	Вертикальные разрывы листа на страницы

Таблица П1.30. *Методы объекта Worksheet*

Метод	Описание
Activate	Активизирует лист рабочей книги MS Excel
Calculate	Рассчитывает все формулы в ячейках листа рабочей книги
ChartObjects	Возвращает объект, представляющий собой диаграмму. Аргумент метода — номер или имя диаграммы
CheckSpelling	Проверяет орфографию. Отображает окно для контроля и исправления ошибок. Если заданы параметры, то они определяют дополнительные условия: используемые словари, регистр символов, режим окна
Copy	Создает копию листа. Параметры метода определяют место, где будет создана копия
Delete	Удаляет лист
Evaluate	Преобразует имя объекта (ячейки), принадлежащего листу, в ссылку на этот объект. Параметром метода является строка, содержащая имя объекта (ячейки)
Move	Перемещает лист. Параметры метода определяют место, куда будет перемещен лист
OLEObjects	Возвращает OLE-объект. Параметр метода представляет собой номер или имя объекта
Paste	Вставляет содержимое буфера обмена в область листа рабочей книги
PasteSpecial	Вставляет текст из буфера обмена в заданную область. В отличие от метода Paste, данный метод позволяет управлять форматом вставляемого текста

Таблица П1.30 *(окончание)*

Метод	Описание
PrintOut	Выполняет печать листа. Если используются параметры, то они позволяют задать режимы и области печати
PrintPreview	Осуществляет предварительный просмотр листа рабочей книги
Protect (Password, DrawingObjects, Contents, Scenarios, UserInterfaceOnly)	Устанавливает защиту на документ. Аргументы позволяют задать режим защиты и пароль
ResetAllPageBreaks	Выполняет начальную установку вертикальных и горизонтальных линий разрывов листа на страницы
SaveAs(FileName:String)	Сохраняет рабочую книгу в файле с новым именем (имя файла — обязательный аргумент). Могут использоваться дополнительные аргументы, определяющие формат сохраняемого документа, режимы доступа и другие свойства
Scenarios	Возвращает сценарий, номер которого используется В качестве аргумента метода
Select	Выделяет содержимое листа рабочей книги
SetBackgroundPicture	Устанавливает рисунок в качестве фона листа. Аргумент метода указывает путь и имя графического файла
UnProtect (Password:String)	Отменяет защиту, установленную для листа рабочей книги. В качестве параметра используется строка пароля, заданного при установке защиты

Таблица П1.31. *Свойства объекта Range*

Свойство	Тип	Описание
Address	String	Адрес области ячеек. Аргументы позволяют задать стиль адреса области
AddressLocal	String	Адрес области ячеек. Аргументы позволяют задать стиль адреса области
Borders	Коллекция	Свойства линий границы области
Cells	Объект	Ячейки области Range
Characters	Коллекция	Символы, составляющие текст области
Column	Integer	Номер первого столбца области Range
Columns	Коллекция	Столбцы области Range листа рабочей книги

Таблица П1.31 *(продолжение)*

Свойство	Тип	Описание
ColumnWidth	Extended	Ширина столбцов области Range
Comment	Объект	Комментарии для верхней левой ячейки области
Count	Integer	Количество ячеек области Range
End	Объект	Область (ячейка), которая находится в крайнем левом, правом, верхнем или нижнем положении на листе рабочей книги в этой области Range
Font	Объект	Шрифт текста области Range
Formula	String	Формула для расчета значений ячеек области Range
FormulaArray	String	Формула для массива ячеек области Range
FormulaHidden	Boolean	True — формула скрыта в защищенной рабочей книге или на защищенном листе
FormulaR1C1	String	Формула в формате R1C1 для расчета значений ячеек области Range
HasArray	Boolean	True — область Range содержит массив формул
HasFormula	Boolean	True — область Range содержит формулу
Height	Extended	Высота области Range
HorizontalAlignment	Integer	Стиль горизонтального выравнивания текста в ячейках области Range
IndentLevel	Integer	Отступ для текста в ячейках области
Interior	Объект	Свойства заливки области ячеек
Item	Объект	Ссылка на элемент коллекции
Left	Extended	Левая координата области Range
Locked	Boolean	True — позволяет заблокировать ячейку или область ячеек при условии, что защищен лист или книга
MergeArea	Объект	Область ячеек, объединенных в "одну ячейку". Возвращает объект Range, ассоциированный с областью ячеек *до* объединения
MergeCells	Boolean	True — область содержит объединенные ячейки
Next	Объект	Указывает на следующий объект коллекции. Если Range представляет собой ячейку, то свойство Next содержит ссылку на следующую ячейку листа

Таблица П1.31 (продолжение)

Свойство	Тип	Описание
NumberFormat, NumberFormatLocal	String	Эти свойства устанавливают формат отображения числовых данных и данных типа "дата" в ячейках области (первое — на языке программы или макроса и едино для всех, второе — на национальном языке пользователя)
Offset	Объект	Область (объект типа Range), смещенную относительно текущей области. Величина смещения задается параметрами метода
Orientation	Integer	Угол наклона текста, отображаемого в ячейках. Величина этого угла может изменяться от –90 до 90 градусов
OutlineLevel	Integer	Уровень группировки для строк или столбцов таблицы листа рабочей книги
PageBreak	Integer	Вертикальный или горизонтальный разрыв листа; используется как свойство для строк или столбцов
Parent	Объект	Ссылка на объект, которому принадлежит область Range
Previous	Объект	Ссылка на предыдущий объект коллекции. Если Range представляет собой ячейку, то Previous содержит ссылку на предыдущую ячейку листа
Resize	Объект	Область, местоположение которой определяется положением области Range, а размер определяется параметрами объекта Resize
Row	Integer	Номер первой строки, принадлежащей области Range
RowHeight	Extended	Высота строк, принадлежащих области Range
Rows	Коллекция	Строки области Range рабочей книги
ShrinkToFit	Boolean	True — текст автоматически сжимается до такой степени, чтобы разместиться в заданных размерах ячейки
Style	String	Стиль текста, используемый для области ячеек рабочей книги
Text	String	Текст, отображаемый в ячейках области рабочей книги
Top	Extended	Верхняя координата области Range
UseStandardHeight	Boolean	True — стандартная высота строк
UseStandardWidth	Boolean	True — стандартная ширина столбцов

Таблица П1.31 *(окончание)*

Свойство	Тип	Описание
Value	Variant	Значение ячейки или массив значений области ячеек
Value2	Variant	Значение ячейки
VerticalAlignment	Integer	Стиль вертикального выравнивания текста в ячейках области Range
Width	Extended	Ширина области Range
Worksheet	Объект	Ссылка на лист рабочей книги, которому принадлежит область ячеек Range
WrapText	Boolean	True — текст в ячейках области будет записан с переносом слов на новую строку, если ширина текста превышает ширину ячейки

Таблица П1.32. *Методы объекта Range*

Метод	Описание
Activate	Активизирует объект. Если применяется к области ячеек Range, то выделяет область
AddComment	Добавляет текст примечания к ячейке. Аргументом служит строка текста
AdvancedFilter	Устанавливает расширенный фильтр
ApplyNames	Устанавливает имена ячеек
ApplyOutlineStyles	Устанавливает типы линий границ
AutoFill	Производит автоматическое заполнение ячеек диапазона
AutoFilter	Устанавливает автоматический фильтр
AutoFit	Автоматически устанавливает ширину столбцов таблицы в зависимости от содержимого ячеек
AutoFormat	Задает формат ячеек. Аргументы метода определяют формат и шрифт текста
BorderAround	Задает свойства линии границы области. В параметрах метода передаются тип, цвет и толщина линии границы
Calculate	Вычисляет результат математического выражения, содержащегося в ячейках заданной области
CheckSpelling	Проверяет орфографию для данной области. Отображает окно для контроля и исправления ошибок. Если заданы параметры, то они определяют дополнительные условия: используемые словари, регистр символов, режим окна

Таблица П1.32 *(продолжение)*

Метод	Описание
Clear	Очищает содержимое и формат ячеек области
ClearComments	Очищает комментарии для ячеек области
ClearContents	Очищает только содержимое, оставляя формат ячеек без изменения
ClearFormats	Очищает формат ячеек области
ClearNotes	Очищает примечания
ColumnDifferences	Возвращает область в столбце таблицы, ячейки которой содержат данные, отличные от значения ячейки того же столбца (эту ячейку задает параметр метода)
Consolidate	Объединяет данные
Copy	Копирует данную область (объект Range со всеми свойствами) в буфер обмена
CopyPicture	Копирует область Range в буфер обмена в виде рисунка
CreateNames	Создает имена для ячеек на основе текста — содержимого соседних ячеек
Cut	Помещает данную область в буфер обмена, затем очищает область при выполнении команды Paste
Delete	Удаляет заданную область из документа
FillDown	Заполняет ячейки области сверху вниз, копируя содержимое верхних ячеек в следующие ячейки диапазона
FillLeft	Заполняет ячейки области справа налево, копируя содержимое правых ячеек в следующие ячейки диапазона
FillRight	Заполняет ячейки области слева направо, копируя содержимое левых ячеек в следующие ячейки диапазона
FillUp	Заполняет ячейки области снизу вверх, копируя содержимое нижних ячеек в следующие ячейки диапазона
Find	Осуществляет поиск данных в заданной области. Аргументы метода задают шаблон и режим поиска. Метод возвращает ссылку на область, содержащую искомые данные
FindNext	Повторяет поиск, возвращает ссылку на следующую область
FindPrevious	Повторяет поиск, возвращает ссылку на предыдущую область
FunctionWizard	Открывает диалог мастера функций
Insert	Вставляет ячейки. Параметры метода определяют направление смещения ячеек

Таблица П1.32 *(окончание)*

Метод	Описание
Justify	Перестраивает текст таким образом, чтобы он равномерно заполнил видимую часть ячеек
Merge	Объединяет ячейки заданной области, если аргумент метода имеет значение True, иначе — разъединяет объединенные ячейки
NoteText	Вставляет или изменяет примечание для ячейки
Parse	На основе содержимого столбца по заданным правилам формирует данные и распределяет их в разные столбцы. Аргументы метода задают правила и местоположение формируемых данных
PasteSpecial	Вставляет текст из буфера обмена в заданную область. В отличие от метода Paste данный метод позволяет управлять форматом вставляемого текста
PrintOut	Отправляет содержимое области на печать. Параметры метода позволяют задать режим печати
PrintPreview	Осуществляет предварительный просмотр печати области
Replace	Осуществляет поиск и замену данных в области Range
RowDifferences	Возвращает область в строке таблицы, ячейки которой содержат данные, отличные от значения ячейки той же строки (параметр метода задает эту ячейку)
Select	Выделяет содержимое области
Sort, SortSpecial	Сортируют содержимое ячеек области
SpecialCells	Возвращает область ячеек, соответствующих определенному типу (текст, формулы, примечание и др.)
UnMerge	Разъединяет объединенные ячейки

Таблица П1.33. *Свойства объекта Font*

Свойство	Тип	Описание
Background	Integer	Характеристика прозрачности фона текста
Bold	Boolean	True — символы шрифта имеют полужирное начертание
Color	Integer (tcolor)	Значение цвета в формате RGB
ColorIndex	Integer	Индекс цвета в палитре цветов
FontStyle	String	Стиль шрифта
Italic	Boolean	True — курсивное начертание символов

Таблица П1.33 *(окончание)*

Свойство	Тип	Описание
Name	String	Наименование шрифта
OutlineFont	Boolean	True — шрифт не изменяется при изменении стиля документа
Parent	Объект	Ссылка на объект, для которого выбирается шрифт
Shadow	Boolean	True — символы текста отображаются с тенью
Size	Integer	Размер шрифта
Strikethrough	Boolean	True — символы текста зачеркнуты
Subscript	Boolean	True — символы текста отображаются в виде нижнего индекса
Superscript	Boolean	True — символы текста отображаются в виде верхнего индекса
Underline	Integer	Стиль подчеркивания текста

Таблица П1.34. *Свойства объекта FillFormat*

Свойство	Тип	Описание
BackColor	Integer (tcolor)	Цвет фона, или второй цвет для градиентной заливки, или цвет фона узора (штриховки) поверхности
ForeColor	Integer (tcolor)	Цвет штриховки или первый цвет для градиентной заливки поверхности
GradientColorType	Integer	Тип цветового градиента
GradientDegree	Extended	Определяет, в какой тон цвета переходит цвет одноцветной градиентной заливки; может иметь значение в пределах от 0 до 1 (плавное изменение от черного цвета до белого)
GradientStyle	Integer	Направление градиента
GradientVariant	Integer	Один из четырех возможных вариантов направления градиента
Parent	Объект	Ссылка на объект, которому принадлежит заливка
Pattern	Integer	Узор заливки
PresetGradientType	Integer	Одна из возможных заготовок градиента
PresetTexture	Integer	Один из возможных образцов текстур

Таблица П1.34 *(окончание)*

Свойство	Тип	Описание
TextureName	String	Имя текстуры, если она выбрана в списке образцов текстур
TextureType	Integer	Тип текстуры (одноцветная, двухцветная или образец)
Transparency	Extended	Может иметь значение в пределах от 0 до 1 и определяет прозрачность заливки
Type	Integer	Тип заливки (градиентная, текстура, узор и др.)
Visible	Boolean	False — заливка становится полностью прозрачной

Таблица П1.35. *Методы объекта FillFormat*

Метод	Описание
OneColorGradient	Устанавливает одноцветную градиентную заливку
Patterned	Устанавливает узор заливки поверхности
PresetGradient	Устанавливает один из возможных заготовленных вариантов градиента
PresetTextured	Устанавливает один из возможных образцов текстуры
Solid	Устанавливает одноцветную заливку (без градиента и текстуры)
TwoColorGradient	Устанавливает двухцветную градиентную заливку
UserPicture	Использует в качестве заливки рисунок из графического файла. Аргумент метода — путь и имя графического файла
UserTextured	Использует в качестве текстуры рисунок из графического файла. Аргумент метода — путь и имя графического файла

Таблица П1.36. *Свойства объекта LineFormat*

Свойство	Тип	Описание
BackColor	Integer (tcolor)	Цвет фона для линии, тип которой — узор (штриховка)
BeginArrowheadLength	Integer	Длина стрелки в начале линии
BeginArrowheadStyle	Integer	Стиль стрелки в начале линии
BeginArrowheadWidth	Integer	Толщина стрелки в начале линии

Таблица П1.36 *(окончание)*

Свойство	Тип	Описание
DashStyle	Integer	Шаблон линии (сплошная линия, варианты штрихпунктирных линий)
EndArrowheadLength	Integer	Длина стрелки в конце линии
EndArrowheadStyle	Integer	Стиль стрелки в конце линии
EndArrowheadWidth	Integer	Толщина стрелки в конце линии
ForeColor	Integer (tcolor)	Цвет узора (штриховки) или цвет линии
Parent	Объект	Ссылка на объект, которому принадлежит линия
Pattern	Integer	Шаблон линии
Style	Integer	Стиль (тип) линии (обычная, двойная толстая + тонкая, двойная тонкая + толстая и т. п.)
Transparency	Extended	Прозрачность линии (может иметь значение в пределах от 0 до 1)
Visible	Boolean	False — невидимая линия
Weight	Integer	Толщина линии

ПРИЛОЖЕНИЕ 2

Ответы на вопросы

В этом приложении даны ответы на типичные вопросы, возникающие при работе с документами MS Office в среде Delphi.

Как подключиться к выполняющемуся приложению Excel?

Во всех предыдущих примерах мы создавали объект Application с помощью функции CreateOleObject. Это влекло за собой запуск соответствующего приложения MS Office. Но когда приложение запущено и требуется создать объект, который будет ссылаться на это приложение, нужно использовать функцию GetActiveOleObject, возвращающую ссылку на объект, класс которого описан в строке — единственном аргументе этой функции. Следующий пример демонстрирует использование данной функции.

Подключение к выполняющемуся приложению MS Excel

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  E:=GetActiveOleObject('Excel.Application');
  E.Visible:=True;
end;
```

В приложениях Delphi, предназначенных для работы с документами MS Office, лучше использовать не одну функцию GetActiveOleObject, а пару функций GetActiveOleObject и CreateOleObject. Это позволит при отсутствии открытых приложений обработать ошибку и открыть приложение для работы с документами. Следующая процедура позволяет запустить приложение MS Office, если оно до этого не было активным.

Запуск неактивного приложения MS Office

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  try
    E:=GetActiveOleObject('Excel.Application');
    except
    E:=CreateOleObject('Excel.Application');
  end;
  E.Visible:=True;
end;
```

Описанный способ использования приложений MS Office в проектах Delphi является более экономичным и гибким и избавит пользователя от лишних манипуляций.

Как освободить память после окончания работы в Excel?

Если я открываю Excel из моей программы, а потом закрываю его и даже делаю Quit, то все равно, пока совсем не закрою мою программу, не могу открыть ни один файл Excel. Как решить эту проблему?

Для освобождения памяти после закрытия Excel с помощью метода Quit используйте такой оператор:

```
E:=UnAssigned;
```

Он освобождает ссылку и память от объекта Application. Используйте следующий фрагмент исходного текста в своих приложениях.

Освобождение памяти по окончании работы с Excel

```
E.Quit;
E:=UnAssigned;
```

Как вставить в документ Word рисунок, не перемещая текст?

Мне потребовалось вставить рисунок в документ Word, не раздвигая текст (перед текстом или за текстом). По логике должно быть какое-то свойство, но найти его не смогла.

Эту проблему можно решить путем использования объектов коллекции Shapes. Объекты коллекции Shapes могут использовать в качестве заливки рисунки, загружаемые из файла. Другим свойством этих объектов является то, что их можно разместить как перед текстом, так и за текстом документа Word. Следующий фрагмент программы с помощью этих свойств помещает рисунок за текст документа Word (в примере сначала вставляется текст, а затем рисунок).

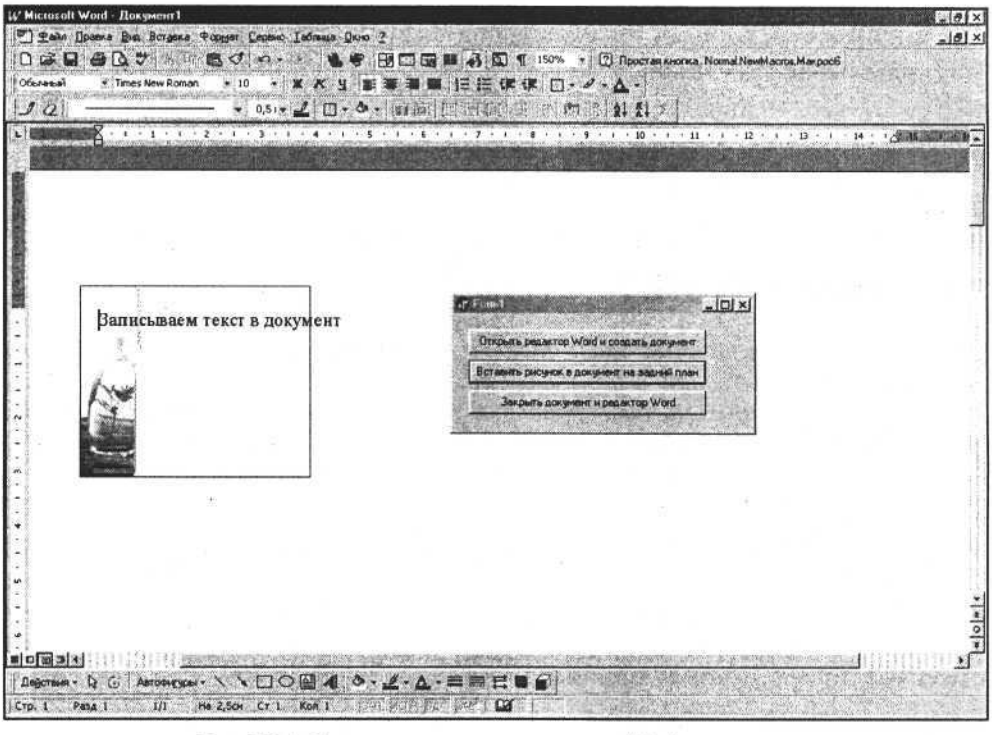


Рис. П2.1. Рисунок размещен в документе Word за текстом

Вставка рисунка в документ Word без перемещения текста

```
const
 msoTextOrientationHorizontal=1;
 msoSendBehindText=5;
 var W:variant;
procedure TForm1.Button1Click(Sender: TObject);
begin
  W:=CreateOleObject('Word.Application');
  W.Visible:=True;
```

```
  W.Documents.Add;
  W.Selection.range:='Записываем текст в документ';
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  W.ActiveDocument.Shapes.AddShape(msoTextOrientationHorizontal,
    80, 60, 120, 100);
  W.ActiveDocument.Shapes.Item(1).ZOrder(msoSendBehindText);
  W.ActiveDocument.Shapes.Item(1).Fill.UserPicture('C:\Мои
 документы\PARFUMERIYA.JPG')
end;
```

Результат представлен на рис. П2.1.

Как выбрать масштаб отображения документа Word?

Одной из решаемых мной задач в разрабатываемом приложении было создание выходной формы в Word. Пользователю потребовалось добавить функцию масштабирования отображения документа (чтобы он мог задавать любой масштаб, например, 100 % или 75 % или 50 %). Как выбрать масштаб?

Масштаб отображения окна документа является свойством объекта View, определяющего режимы отображения области окна. Программно можно задать масштаб, определив его в процентах от нормальной величины. Для этого используется свойство Percentage объекта Zoom. Данное свойство имеет тип Integer и может принимать значения от 10 до 500.

Рассмотрим пример процедуры.

Изменение масштаба отображения документа

```
procedure TForm1.Button3Click(Sender: TObject);
begin
  W.ActiveWindow.ActivePane.View.Zoom.Percentage:=SpinEdit1.Value;
end;
```

На рис. П2.2 представлен результат использования свойства Percentage.

Для того чтобы задать один из наиболее удобных и часто используемых масштабов, можно воспользоваться свойством PageFit объекта Zoom. В этом случае масштаб автоматически рассчитывается и устанавливается таким образом, чтобы одна страница могла разместиться в активном окне целиком.

Этот режим задается путем записи значения константы wdPageFitFullPage в свойство PageFit. Чтобы автоматически задать другой масштаб отображения, когда ширина отображаемой страницы приравнивается ширине окна, в свойство PageFit необходимо записать значение константы wdPageFitBestFit. Для отключения режима автоматического масштабирования в свойство PageFit записывается константа wdPageFitNone. Реализация способа автоматического масштабирования представлена в виде следующей процедуры.

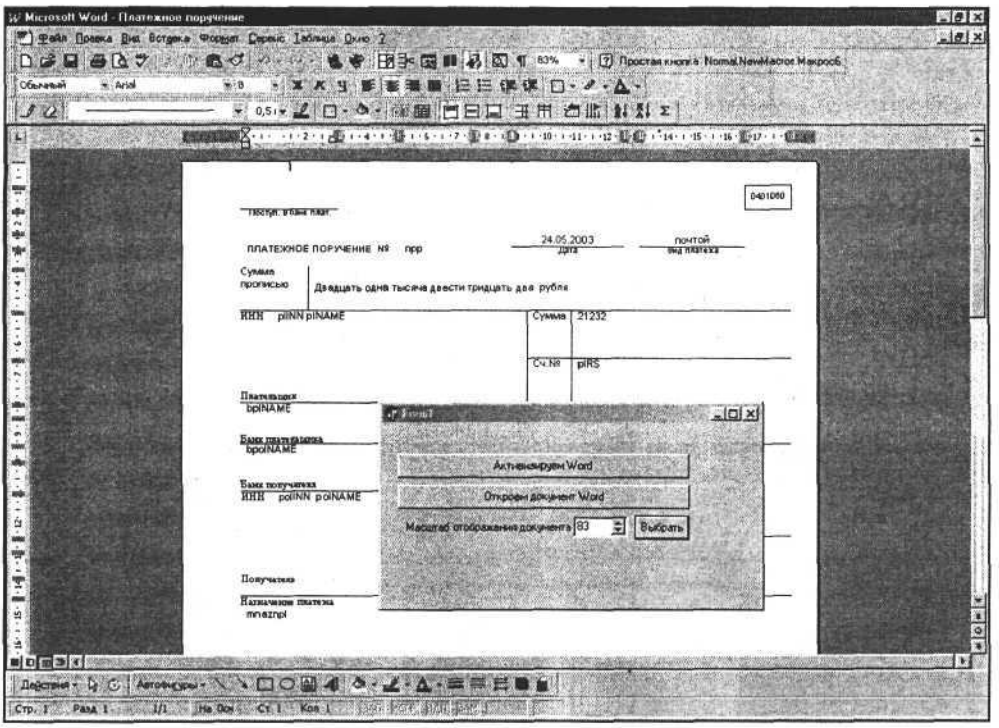


Рис. П2.2. Задаем масштаб отображения документа

Задание режима автоматического масштабирования

```
procedure TForm1.PageFitChange(Sender: TObject);
begin
  W.ActiveWindow.ActivePane.View.Zoom.PageFit:=PageFit.ItemIndex;
end;
```

Есть еще один способ, позволяющий косвенно задавать масштаб отображения документа. Этот способ основан на использовании свойств PageColumns и PageRows объекта Zoom, позволяющих задать количество одновременно отображаемых страниц документа в одном окне. Значение свойства

PageColumns определяет количество страниц, размещенных по ширине активного окна, а значение свойства PageRows — количество страниц, размещенных по высоте активного окна. Таким образом, общее количество страниц будет определяться величиной PageColumns × PageRows. Очевидно, что количество одновременно отображаемых страниц влияет на масштаб отображения страниц документа.

Задание количества одновременно отображаемых страниц по ширине окна

```
procedure TForm1.SpinEdit2Change(Sender: TObject);
begin
  if SpinEdit2.Value<1 then exit;
  W.ActiveWindow.ActivePane.View.Zoom.PageColumns:=SpinEdit2.Value;
end;

procedure TForm1.SpinEdit3Change(Sender: TObject);
begin
  if SpinEdit3.Value<1 then exit;
  W.ActiveWindow.ActivePane.View.Zoom.PageRows:=SpinEdit3.Value;
end;
```

Результат выполнения описанных процедур представлен на рис. П2.3.

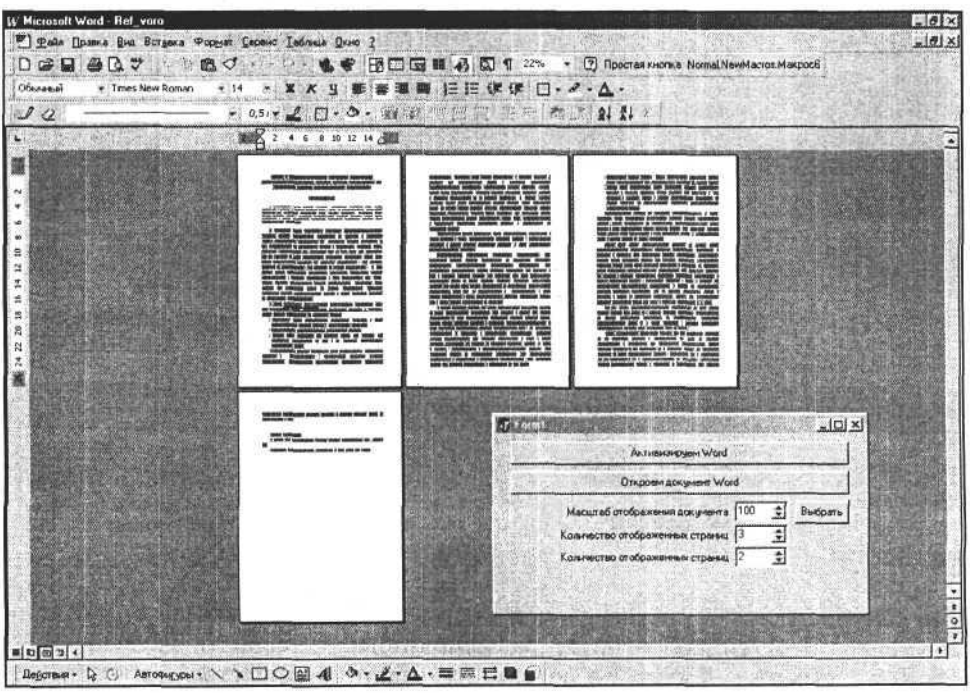


Рис. П2.3. Задаем масштаб отображения путем задания количества страниц

Как добавить новую страницу в документ Word?

Для того чтобы вставить новую страницу в документ Word, используйте метод InsertBreak объекта Range с параметром Type:=wdPageBreak. Новая страница вставляется в область, заданную объектом Range, поэтому от положения этого объекта зависит то, будет новая страница добавлена в конец документа или сплошной текст будет разорван и размещен на разных страницах. Для того чтобы добавить страницу в конец документа, можно использовать метод InsertBreak объекта Range, указывающий на последний символ документа.

Добавление страницы в конец документа

```
const wdPageBreak=7;
 var W:variant;
procedure TForm1.Button3Click(Sender: TObject);
begin
  W.ActiveDocument.Range(W.ActiveDocument.Range.end-1).↵
          InsertBreak(Type:=wdPageBreak);
end;
```

Другой способ более универсален. Перемещаем курсор в конец документа и вызываем метод InsertBreak объекта Selection. Этот способ применим и для того, чтобы вставить страницу, разорвав текст в середине документа. Перед использованием этого способа необходимо разместить курсор в том месте, где требуется вставить страницу. Для этого можно выбрать любой способ, например поиск фрагмента текста в документе. Следующие две процедуры демонстрируют то, как работает этот способ. Вторая процедура отличается от первой тем, что сначала выделяется искомый фрагмент, а затем вызывается метод InsertBreak объекта Selection. Это является избыточным, но наглядным как пример.

Добавление страницы в разрыв текста документа

```
procedure TForm1.Button4Click(Sender: TObject);
 var myRange:variant;
begin
  myRange:=W.ActiveDocument.Range;
  myRange.Find.Execute(FindText:='my', Forward:=True);
  if myRange.Find.Found Then myRange.InsertBreak(Type:=wdPageBreak);
end;
```

```
procedure TForm1.Button4Click(Sender: TObject);
 var myRange:variant;
begin
  myRange:=W.ActiveDocument.Range;
  myRange.Find.Execute(FindText:='my', Forward:=True);
  if myRange.Find.Found then begin
    myRange.select;
    W.Selection.InsertBreak(Type:=wdPageBreak);
  end;
end;
```

Как пронумеровать страницы в документе Word?

Номера страниц представляют собой элементы коллекции PageNumbers. Сама коллекция PageNumbers является свойством элемента коллекции Headers, представляющего собой верхний колонтитул страницы, или элемента коллекции Footers (нижний колонтитул страницы). Если документ имеет более одного раздела, то нумерацию можно создать отдельно для каждого раздела. При создании нумерации страниц можно выбрать вариант пространственного размещения номера страницы и порядок нумерации первой страницы.

Новая нумерация страниц создается с помощью метода Add(PageNumberAlignment:integer, FirstPage:Boolean) коллекции PageNumbers. Первый аргумент этого метода определяет выравнивание номера страницы по горизонтали, второй аргумент — наличие номера на первой странице документа (True) или его отсутствие (False).

Для создания нумерации страниц в верхнем колонтитуле используем следующую процедуру.

Размещение номеров страниц в верхнем колонтитуле

```
procedure TForm1.Button3Click(Sender: TObject);
 var mySection:variant;
   myPageNumber:variant;
   myPageNumbers:variant;
begin
  mySection:=W.ActiveDocument.Range.Sections.item(1);
  MyPageNumbers:=mySection.Headers.item(1).PageNumbers;
  MyPageNumber:=mySection.Headers.item(1).PageNumbers.
  Add(PageNumberAlignment:=wdAlignPageNumberRight, FirstPage:=True)
end;
```

Для создания нумерации в нижнем колонтитуле страницы используем элемент коллекции Footers и его свойство — PageNumbers.

Размещение номеров страниц в нижнем колонтитуле

```
procedure TForm1.Button4Click(Sender: TObject);
 var mySection:variant;
begin
  mySection:=W.ActiveDocument.Range.Sections.item(1);
  MyPageNumbers:=mySection.Footers.item(1).PageNumbers;
  MyPageNumber:=mySection.Footers.item(1).PageNumbers.Add ↵
(PageNumberAlignment:=wdAlignPageNumberRight, FirstPage:=True);
end;
```

В представленных процедурах ссылка mySection указывает на первый раздел документа, для которого будет создана нумерация (документ не может иметь меньше одного раздела). Второй оператор предоставляет доступ к коллекции PageNumbers, средствами которой мы создаем новую нумерацию (третий оператор).

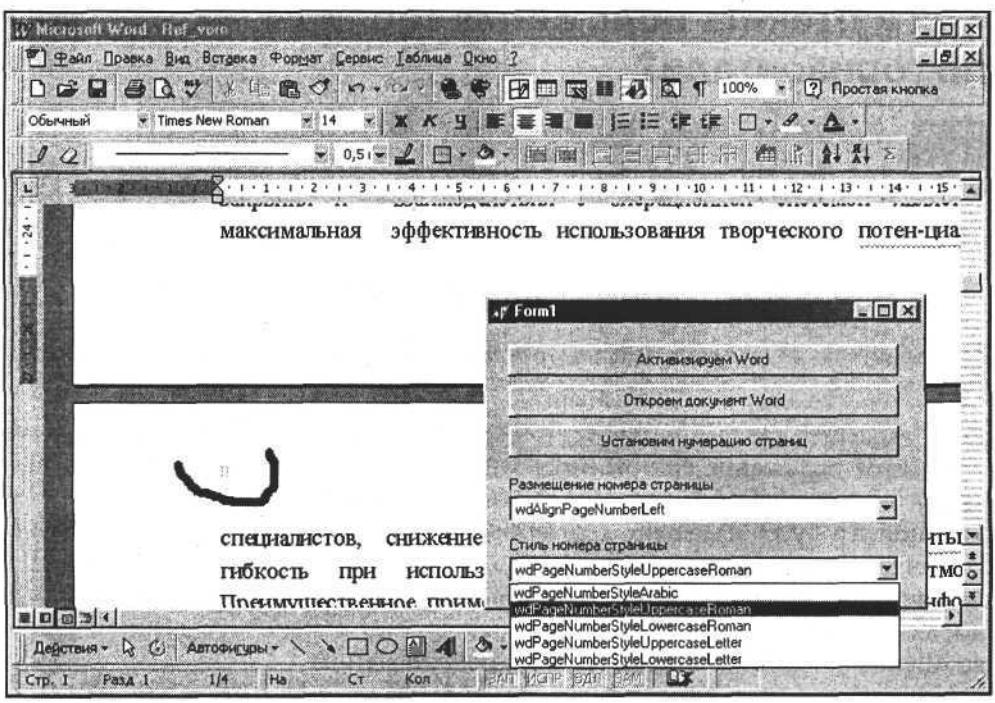


Рис. П2.4. Изменяем стиль и горизонтальное выравнивание номеров страниц

Стиль созданной нумерации страницы можно изменить. Для этого воспользуемся объектами, ссылки на которые были получены во время ее создания. С их помощью изменим горизонтальное выравнивание номеров страниц и их стиль.

Задание горизонтального выравнивания и стиля номеров страниц

```
procedure TForm1.AlignmentChange(Sender: TObject);
begin
  MyPageNumber.Alignment:=Alignment.ItemIndex;
end;

procedure TForm1.PageNumberStyleChange(Sender: TObject);
begin
  MyPageNumbers.NumberStyle:=PageNumberStyle.ItemIndex;
end;
```

На рис. П2.4 представлен результат выполнения процедур создания и изменения выравнивания и стиля номеров страниц.

Как изменить положение таблицы по горизонтали?

Я пробовал задать Alignment для таблицы разными способами:

1. `W.ActiveDocument.Tables.Item(Table).Select;`
   `sel_:=W.selection;`
   `sel_.Range.ParagraphFormat.Alignment := align;`
2. `W.ActiveDocument.Tables.Item(Table).Range.ParagraphFormat.Alignment;`

Но почему-то получается задать Alignment для ячеек, а не для самой таблицы.

Если выделить таблицу и изменить свойство Alignment объекта Selection, то изменится положение содержимого ячеек. Также если выделить не всю таблицу, а несколько ее ячеек, и изменить свойство Alignment объекта Selection, то будет изменено выравнивание содержимого выделенных ячеек. Изменить выравнивание самой таблицы можно с помощью свойства Alignment коллекции Rows выбранной таблицы. Когда требуется изменить выравнивание только выбранной строки, нужно изменить значение свойства Alignment для этой строки. С помощью оператора

```
W.ActiveDocument.Tables(Table).Rows.Alignment:=align;
```

мы добиваемся изменения положения таблицы с помощью выравнивания по горизонтали.

Изменение выравнивания таблицы и первой строки таблицы

```
procedure TForm1.alignChange(Sender: TObject);
begin
W.ActiveDocument.Tables.Item(1).Rows.Alignment:=align.ItemIndex;
end;

procedure TForm1.alignrChange(Sender: TObject);
begin
W.ActiveDocument.Tables.Item(1).Rows.Item(1).Alignment:=alignr.ItemIndex;
end;
```

Результат использования представленных процедур показан на рис. П2.5. Первая строка таблицы выровнена по центру, оставшаяся часть таблицы — по правому краю.

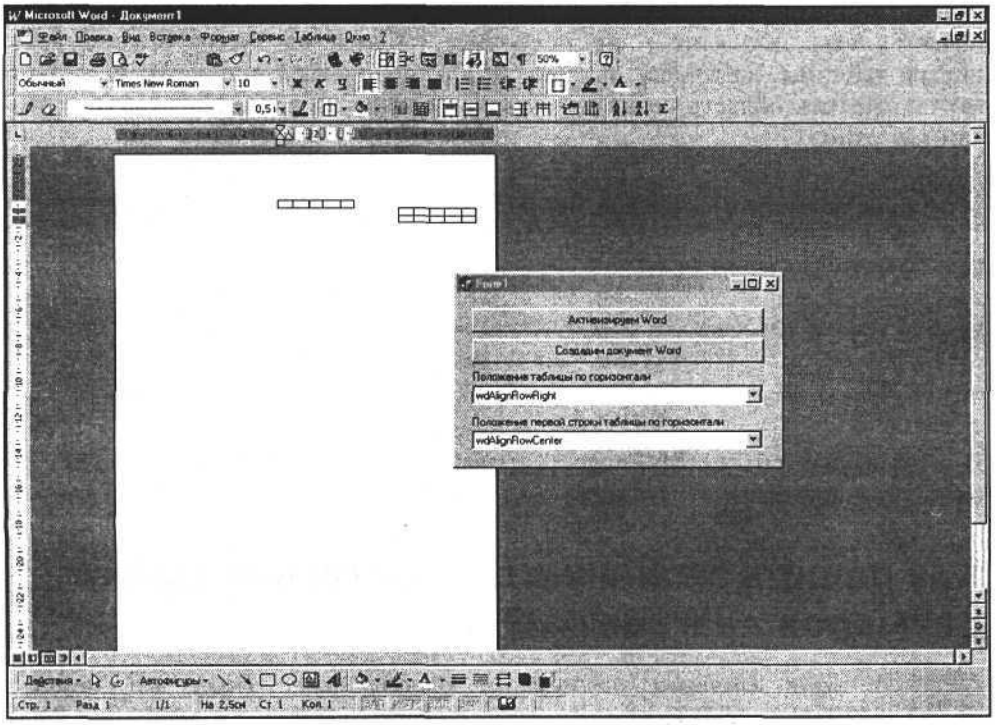


Рис. П2.5. Изменение выравнивания таблицы и ее первой строки

Как решить проблему с добавлением новой таблицы в документ Word?

Есть рабочая книга Excel, в ней — "Лист 1" с выполненным шаблоном. Этот лист нужно перенести/скопировать в "Лист 2", в этой же книге. В Excel все идет нормально, а из Delphi не получается.

Таблицу в новом документе можно создать с помощью метода Add коллекции Tables.

Создание таблицы в новом документе

```
MyRange:=W.ActiveDocument.Range;
W.ActiveDocument.Tables.Add(Range:=MyRange, NumRows:=2,NumColumns:=2);
```

Если пытаться создать таблицу в документе этим способом после того, как ранее была создана хотя бы одна таблица, возникнет ошибка, связанная с использованием первого аргумента метода Add, который определяет область, где будет создана таблица. В приведенном варианте такой областью является весь документ — но поскольку таблица не может быть создана внутри таблицы, возникает ошибка. Для решения этой проблемы необходимо определить область создания новой таблицы за пределами таблиц, созданных ранее.

Создание таблицы в конце документа

```
var MyRange:variant;
...
MyRange:=W.ActiveDocument.Range(W.ActiveDocument.Range.End-1,
                                W.ActiveDocument.Range.End-1);
W.ActiveDocument.Tables.Add(Range:=MyRange,NumRows:=2,NumColumns:=2);
...
```

В этом случае объект MyRange указывает на конец документа, где нет ни одной созданной ранее таблицы.

Как решить типичную проблему настройки размеров диаграммы?

У меня возникла проблема с диаграммами. Пишу на Delphi 6, MS Office 2000. Создаю диаграмму, затем пытаюсь изменить ее координаты и размеры, но возникает ошибка.

Вот мой обработчик события:

```
procedure TGraph.ExcelExportClick(Sender: TObject);
var
Excel,WorkBook,Sheet1,Sheet2,Chart:variant;
ChartName:string;
Sheet,SeriesCollection:Variant;
vrange:variant;
i,j:integer;
begin
// Создаем экземпляр объекта автоматизации
Excel:=CreateOleObject('Excel.Application');
Excel.Visible:=true;
Excel.SheetsInNewWorkbook := 2; // Количество листов в создаваемой книге
WorkBook:=Excel.Workbooks.Add;  // Создание новой рабочей книги
Sheet1:=WorkBook.WorkSheets[1];
Sheet2:=WorkBook.WorkSheets[2];
ChartName:=Excel.Charts.Add.Name; // Создание новой диаграммы
// А далее при попытке поменять любое свойство диаграммы возникает
// ошибка типа "Нельзя установить свойство Left класса ChartArea"
// и т. д. для всех свойств.
Excel.Charts.Item[ChartName].ChartArea.Left:=10;
Excel.Charts.Item[ChartName].ChartArea.Top:=10;
Excel.Charts.Item[ChartName].ChartArea.Width:=20;
Excel.Charts.Item[ChartName].ChartArea.Height:=20;
Excel.Charts.Item[ChartName].HasTitle:=true;
Excel.Charts.Item[ChartName].ChartTitle.Text:='XXX';
.....................................................
```

Что-то здесь не так. Подскажите, в чем дело.

Диаграмма создается на отдельном листе, поэтому изменить ее размеры и положение нельзя, это не имеет смысла. Для включения заголовка необходимо определить область данных для диаграммы и только после этого включать заголовок.

Вот текст программы.

Создание диаграммы, определение области данных и включение заголовка

```
uses ComObj;
var Excel:variant;
  Sheet1,Sheet2:variant;
  WorkBook:variant;
  ChartName:string;
```

```
procedure TForm1.Button1Click(Sender: TObject);
begin
Excel:=CreateOleObject('Excel.Application');
Excel.Visible:=true;
Excel.SheetsInNewWorkbook := 2;
WorkBook:=Excel.Workbooks.Add;
Sheet1:=WorkBook.WorkSheets[1];
Sheet2:=WorkBook.WorkSheets[2];
ChartName:=Excel.Charts.Add.Name;
// Задаем область данных диаграммы
Excel.Charts.Item[ChartName].SeriesCollection.
              Add(Source:=Sheet1.Range['D4:D12']);
Excel.Charts.Item[ChartName].HasTitle:=True;
Excel.Charts.Item[ChartName].ChartTitle.Characters.Text:='XXX';
end;
```

В результате выполнения данной процедуры будет создана диаграмма и изменен заголовок в том случае, если задана область данных диаграммы.

Как копировать лист в Excel?

Есть рабочая книга Excel, в ней — "Лист 1" с выполненным шаблоном. Этот лист нужно перенести/скопировать в "Лист 2", в этой же книге. В Excel все идет нормально, а из Delphi не получается.

Копирование одного листа рабочей книги выполняется методом Copy элемента коллекции Sheets. Этот метод копирует, создает полную копию выбранного листа со всеми элементами настроек и всей информацией в ячейках таблицы.

Копирование листа рабочей книги

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  Excel.sheets.item['лист1'].copy
end;
```

В представленном примере копия листа рабочей книги по умолчанию создается в новой рабочей книге. Чтобы создать копию листа в той книге, где находится оригинал, необходимо использовать метод Copy с параметром, представляющим собой ссылку на лист, рядом с которым необходимо поместить копию. Следующие два примера создают копии листа в той же ра-

бочей книге, где находится копируемый лист, и располагают их после или до оригинала.

Расположение копий листа после или до оригинала

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  MySheet:=Excel.sheets.item['лист1'];
  MySheet.copy(after:=mysheet);
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  MySheet:=Excel.sheets.item['лист1'];
  MySheet.copy(Before:=mysheet);
end;
```

Результат выполнения данных процедур представлен на рис. П2.6.

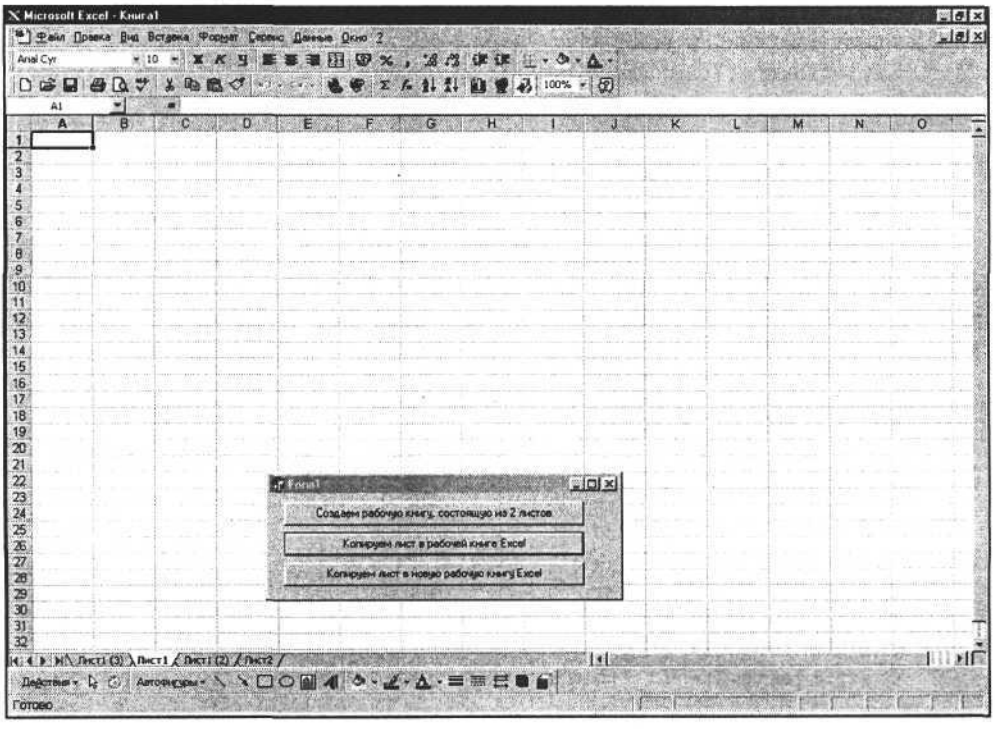


Рис. П2.6. Создаем копии листа рабочей книги

Как обратиться к существующей диаграмме в открытой книге?

Уже несколько лет делаю все отчеты с Delphi через Excel. Сейчас проблема — есть шаблон с подготовленной диаграммой, и не знаю, как к ней обратиться.

Когда раньше создавал ее сам (из программы), проблем не было, а вот до уже существующей на листе диаграммы почему-то не достучаться — все ошибки получаются, если по индексу или по имени (как в макросах Excel сам ее называет). Мне надо максимальное значение шкалы Y установить на заданную величину. Вот фрагмент кода:

```
XLApp:=Excel_TLB.CoApplication_.Create;
WorkBks:=XLApp.WorkBooks as WorkBooks;
WorkBks:=XLApp.WorkBooks;
Workbk:=XLApp.Workbooks.Add(ExtractFilePath(ParamStr(0))+
                    'Shablon\pareto.xlt', 0);
ASheets:=Workbk.WorkSheets;
WorkSheet:=ASheets.Item['изделия'] as _Worksheet;
// А дальше все ошибки при разных вариантах
// AChart:=ASheets.Item['Диаграмма1'] as Chart;
// AChart:=XLApp.Workbooks.Item[1].Charts[1] As Chart;
// AChart:=WorkSheet.ChartObjects('Диагр. 1',EmptyParam) As Chart;
```

Доступ к созданным ранее диаграммам обеспечивается через элементы коллекции ChartObjects с помощью позднего связывания. В свою очередь эта коллекция принадлежит листу рабочей книги. Посредством свойств и методов этой коллекции мы получаем список диаграмм, а затем, используя имя или индекс диаграммы, получаем доступ к выбранной диаграмме.

Получение списка диаграмм

```
procedure TForm1.Button2Click(Sender: TObject);
 var a_:integer;
begin
  ListBox1.Items.Clear;
  for a_:=1 to E.ActiveWorkBook.ActiveSheet.ChartObjects.Count do begin
     ListBox1.Items.Add(E.ActiveWorkBook.ActiveSheet.
      ChartObjects.Item[a_].name);
  end;
end;
```

На основании списка диаграмм, предварительно загруженного в компонент ListBox1, получаем ссылку на диаграмму, выбранную пользователем, и изменяем ее свойства.

Получение доступа к существующей диаграмме и изменение ее свойств

```
procedure TForm1.ListBox1Click(Sender: TObject);
begin
  Chart:=E.ActiveWorkBook.ActiveSheet.ChartObjects.
    Item[ListBox1.ItemIndex+1];
end;

procedure TOKBottomDlg2.SpinEdit1Change(Sender: TObject);
begin
  Chart.Left:=SpinEdit1.Value;
end;

procedure TOKBottomDlg2.SpinEdit2Change(Sender: TObject);
begin
  Chart.Top:=SpinEdit2.Value;
end;
```

Как в выбранной ячейке таблицы документа Word писать снизу вверх?

Для того чтобы изменять направление текста, используйте свойство Orientation объекта Range, если этот объект связан с областью ячейки или областью выбранных ячеек. Алгоритм построения приложения в данном случае выглядит так: сначала определяем ссылку на объект Range выбранной ячейки для заданной таблицы, затем изменяем свойство самого объекта Range.

Изменение направления текста в ячейке таблицы документа Word

```
procedure TForm1.Button3Click(Sender: TObject);
begin
  Range:=W.ActiveDocument.Tables.Item(1).Cell(1,1).Range;
  Range.Text:='Как в выбранной ячейке таблицы документа Word писать ↵
    снизу вверх?';
end;

procedure TForm1.TextOrientationChange(Sender: TObject);
begin
```

```
  case TextOrientation.ItemIndex of
  0:Range.Orientation:=wdTextOrientationHorizontal;
  1:Range.Orientation:=wdTextOrientationUpward;
  2:Range.Orientation:=wdTextOrientationDownward;
  end;
end;
```

Выполняя данные процедуры, получим результат, представленный на рис. П2.7.

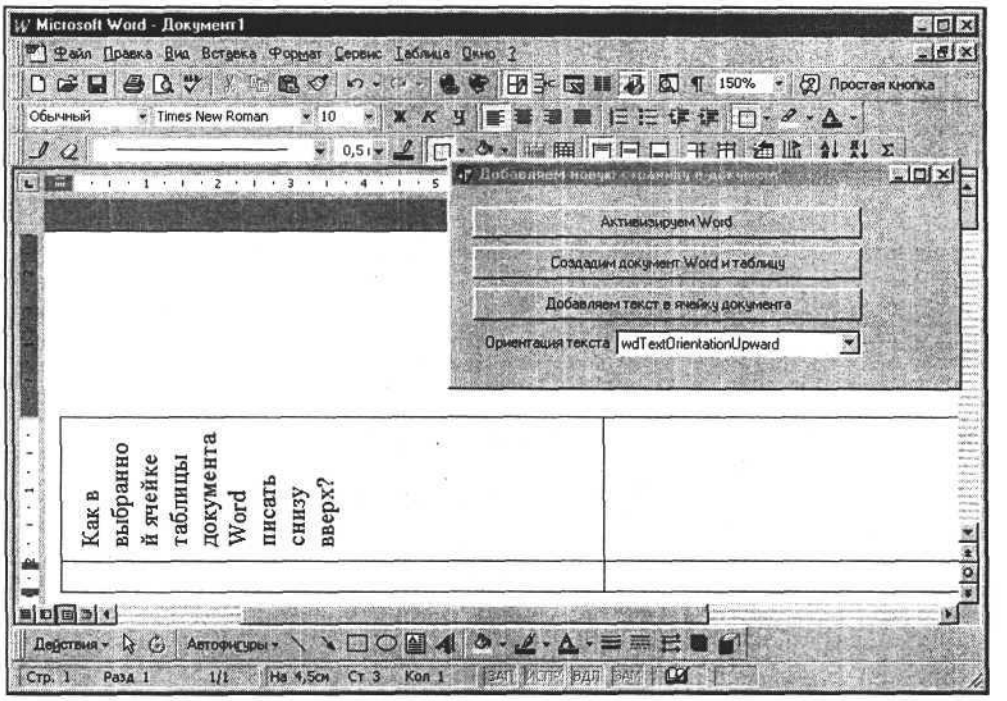


Рис. П2.7. Изменяем направление текста в ячейке таблицы

Как заполнять ранее созданные надписи книги Excel из проекта Delphi?

Есть некоторый шаблон в формате Excel, в котором уже вставлены надписи. Требуется заполнять эти надписи данными из программы на Delphi. Здесь встает вопрос: как обратиться к уже существующим надписям? Я делаю следующее.

Включаю в Excel запись макроса, заполняю надпись текстом. По строке макроса, например ActiveSheet.Shapes('Текст 9').Select, узнаю имя надписи.

В Delphi пишу:

```
E.ActiveWorkBook.ActiveSheet.Shapes('Текст 9').Select;
E.ActiveWorkBook.ActiveSheet.Selection.Characters.Text := 'Текст';
```

Появляется ошибка "Не найден член группы".

Также хотелось бы узнать, как можно изменять имена надписей.

Для того чтобы изменять текст надписи, принадлежащей листу рабочей книги Excel, нужно получить доступ к изменяемой надписи. Доступ можно получить, используя имя или индекс объекта в коллекции Shapes. Последний вариант надежнее. Также надо учесть, что коллекция Shapes может содержать объекты не только типа TextBox, но и другие, например геометрические фигуры. Поэтому при загрузке списка имен объектов TextBox необходимо учитывать тип загружаемого объекта, т. е. свойство Type элемента коллекции Shapes.

Загрузим список имен объектов типа TextBox в компонент ListBox1.

Получение списка имен надписей для листа рабочей книги

```
procedure TForm1.Button3Click(Sender: TObject);
 var a_:integer;
begin
  ListBox1.Items.Clear;
  for a_:= 1 to E.ActiveSheet.Shapes.Count do
    if E.ActiveSheet.Shapes.Item(a_).type = 17
    then ListBox1.Items.Add(E.ActiveSheet.Shapes.Item(a_).Name)
    else ListBox1.Items.Add('-');
end;
```

Чтобы не загружать имена объектов, не являющихся надписями, нужно проверять тип объекта. Если свойство Type загружаемого объекта равно 17, то этот объект является надписью (TextBox) и его имя добавляется в список компонента ListBox1, в противном случае в список добавляется прочерк ('-').

После того как имена объектов TextBox загружены, можно получить доступ к любому из этих объектов. В следующей процедуре, вызываемой при выборе элемента списка компонента ListBox1, мы получаем ссылку на выбранный пользователем объект TextBox (надпись), затем считываем его имя и текст, который содержится в свойстве TextFrame этого объекта.

Считывание имени и текста выбранной надписи

```
procedure TForm1.ListBox1Click(Sender: TObject);
begin
  TextBox:=E.ActiveSheet.Shapes.item(ListBox1.ItemIndex+1);
```

```
  Edit1.Text:=TextBox.Name;
  Memo1.Text:=TextBox.TextFrame.Characters.Text;
end;
```

С помощью считанных имени и текста надписи можно оценить эту информацию, а также при необходимости изменить ее. Далее представлены фрагменты исходного текста процедур, позволяющих изменить содержание свойства TextFrame, т. е. текст, отображаемый надписью, и имя объекта TextBox. В обоих случаях можно использовать механизм обработки исключительных ситуаций, позволяющий корректно исправить ошибки пользователя.

Редактирование текста и имени надписи

```
procedure TForm1.Memo1Change(Sender: TObject);
 var text_:string;
begin
  text_:=Memo1.Text;
  try
    TextBox.TextFrame.Characters.Text:=Memo1.Text;
    except
    Memo1.Text:=text_;
  end;
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
  try
    TextBox.Name:=Edit1.Text;
    ListBox1.Items.Strings[ListBox1.ItemIndex]:=TextBox.Name;
    except
  end;
end;
```

Выбранный в приложении Delphi объект документа Excel можно выделить. Для этого воспользуемся методом Select объекта TextBox в следующей процедуре.

Выделение надписи

```
procedure TForm1.Button5Click(Sender: TObject);
begin
  try
    TextBox.select;
```

```
    except
  end;
end;
```

Результат всех приведенных здесь действий показан на рис. П2.8.

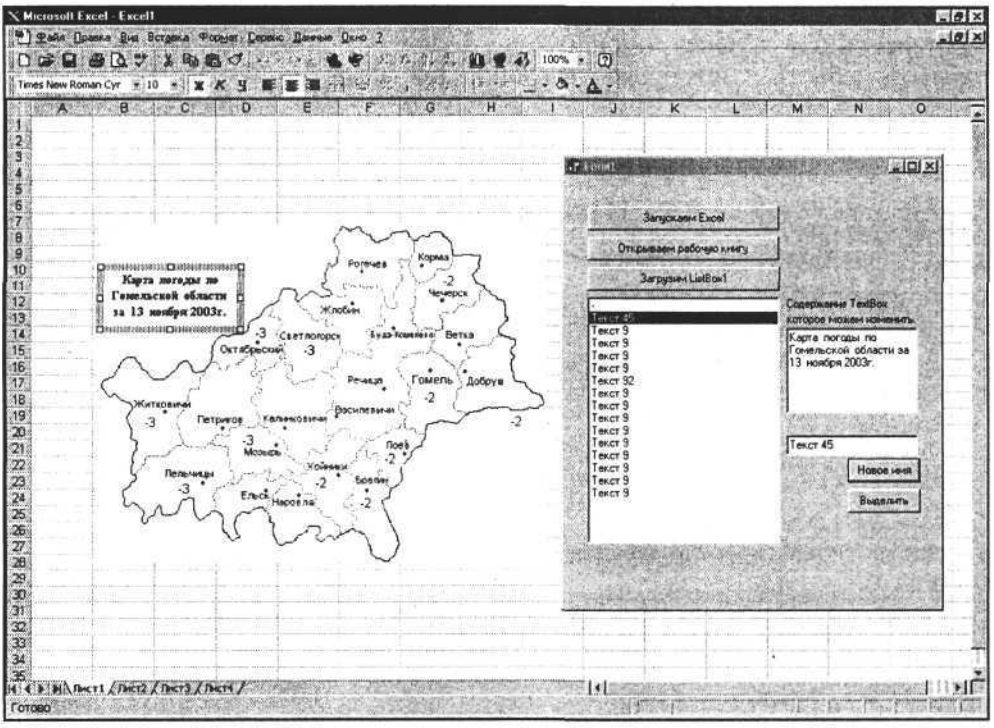


Рис. П2.8. Работа с существующими объектами TextBox

Как работать с абзацами?

Часто возникает необходимость форматирования документа и изменения свойств абзацев. Для написания формул приходится использовать перевод символов в верхний или нижний индекс (например, икс в квадрате). Как автоматизировать эти действия из приложений Delphi?

Для работы с абзацами используется коллекция Paragraphs, которая определяет, каким образом документ разбит на абзацы и способ отображения текста в каждом отдельном абзаце. Количество абзацев в документе определяется свойством Count коллекции Paragraphs, например: W.ActiveDocument.Range.Paragraphs.Count. Также можно использовать спи-

сок абзацев не только для всего документа, но и для его определенной части. Для этого определяем размеры и положение области Range.

Метод Add коллекции Paragraphs добавляет новый абзац, метод CloseUp — устраняет интервал между абзацами. Для доступа к свойствам конкретного абзаца используйте метод Item(i:integer), где i — индекс абзаца в коллекции Paragraphs.

Загрузим список абзацев в приложение Delphi и попробуем изменить некоторые их свойства с помощью следующих процедур.

Получение списка абзацев активного документа и задание их свойств

```
procedure TForm1.Button3Click(Sender: TObject);
 var a_:integer;
begin
if not OpenDialog1.Execute then exit;
W.Documents.Open(OpenDialog1.FileName);
for a_:=1 to W.ActiveDocument.Range.Paragraphs.Count do begin
  ListBox1.Items.Add('Абзац - '+inttostr(a_));
  end;
end;

// Изменим стиль буквицы
procedure TForm1.DropCap_PositionChange(Sender: TObject);
begin
  W.ActiveDocument.Range.Paragraphs.Item(ListBox1.ItemIndex).
    DropCap.Position:=DropCap_Position.ItemIndex;
end;

// Зададим отступ буквицы от левой границы
procedure TForm1.FirstLineIndentChange(Sender: TObject);
begin
  W.ActiveDocument.Range.Paragraphs.Item(ListBox1.ItemIndex).
    FirstLineIndent:=FirstLineIndent.Value;
end;

// Отобразим линию границы абзаца
procedure Tline_.Borders_EnableClick(Sender: TObject);
begin
  Borders.Enable:=Borders_Enable.Checked;
end;

// Зададим тип линии границы абзаца
procedure Tline_.SpinEdit1Change(Sender: TObject);
```

```
begin
  Borders.OutsideLineStyle:=LineStyle.Value;
end;
```

Пример результата таких преобразований показан на рис. П2.9.

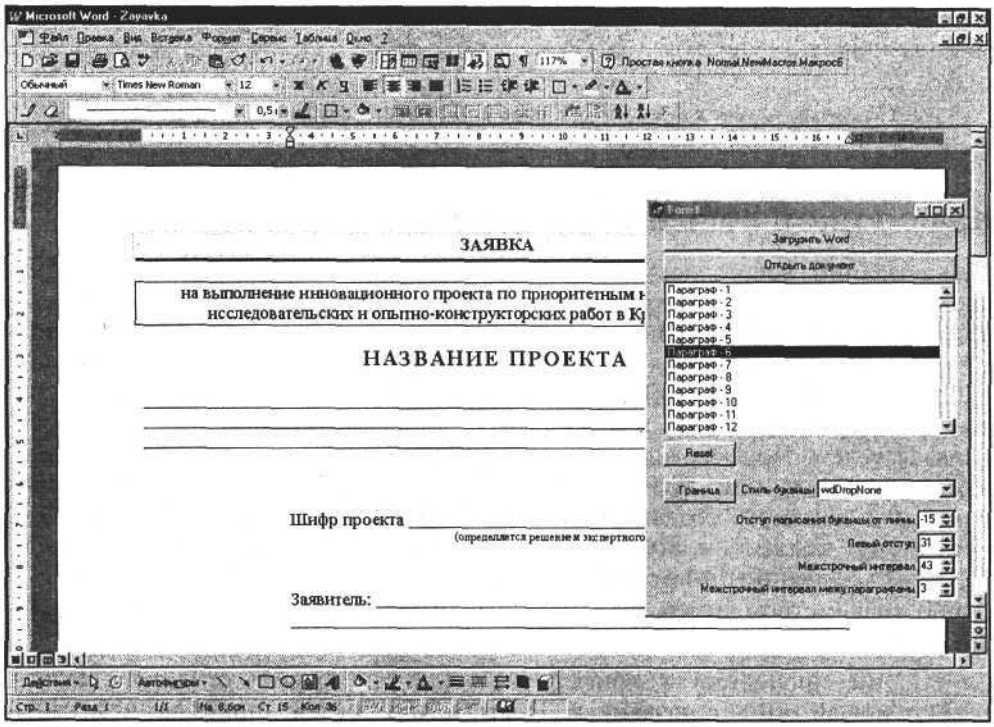


Рис. П2.9. Изменяем свойства абзацев в открытом документе

Как перевести символы текста в верхний или нижний индекс?

Перевод символов в верхний или нижний индекс производится путем использования объекта Font и его свойств Subscript и Superscript, имеющих логический тип. Запись значения True в свойство Subscript переведет символы текста в нижний индекс, а запись True в свойство Superscript — в верхний индекс. Перед использованием этих свойств объекта Font необходимо определить фрагмент (область) текста, к которому будут применены эти изменения. Область текста определяется объектом Range, а объект Font (шрифт) в свою очередь является свойством объекта Range. Поэтому процедуры Delphi, реализующие перевод символов в верхний или нижний индекс, могут быть, например, следующими.

Перевод символов текста в верхний и нижний индекс

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  W.ActiveDocument.Range(1,10).Font.Subscript:=True;
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
  W.ActiveDocument.Range(11,20).Font.Superscript:=True;
end;
```

Как создать новый стиль текста?

При создании нового документа или редактировании созданного и сохраненного ранее часто приходится изменять разные фрагменты документа. Можно задавать фрагментам текста такой вид, который выделит их и придаст им отличия по смыслу и назначению, для этого потребуется изменять различные параметры шрифта и фона. Для упрощения этой процедуры можно применять стили, имеющиеся в приложении Word или созданные пользователем. Новый стиль можно создать и программно из внешних приложений. Коллекция Styles объекта Document позволяет создать новый стиль или изменить созданный ранее. Новый стиль создается с помощью метода Add коллекции Styles, возвращающего ссылку на созданный стиль (элемент коллекции). Метод Item(i) обеспечивает доступ к созданному ранее стилю (параметром этого метода может быть как индекс, так и название стиля).

Следующие процедуры — пример работы с текстовыми стилями. Первая создает новый стиль с названием 'Новый стиль текста', шрифтом Arial (размер 36) и текстурной заливкой абзаца 10% серого цвета. Вторая процедура изменяет имя шрифта, его размер и текстуру абзаца.

Создание нового стиля и настройка его свойств

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  MyStyles:=W.ActiveDocument.Styles.Add('Новый стиль текста');
  MyStyles.Font.Name:='Arial';
  MyStyles.Font.Size:=36;
  MyStyles.Shading.Texture:=wdTexture10Percent;
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
  MyStyles:=W.ActiveDocument.Styles.Item('Новый стиль текста');
```

```
  MyStyles.Font.Name:='Tahoma';
  MyStyles.Font.Size:=26;
  MyStyles.Shading.Texture:=wdTexture20Percent;
end;
```

Результат выполнения процедур представлен на рис. П2.10.

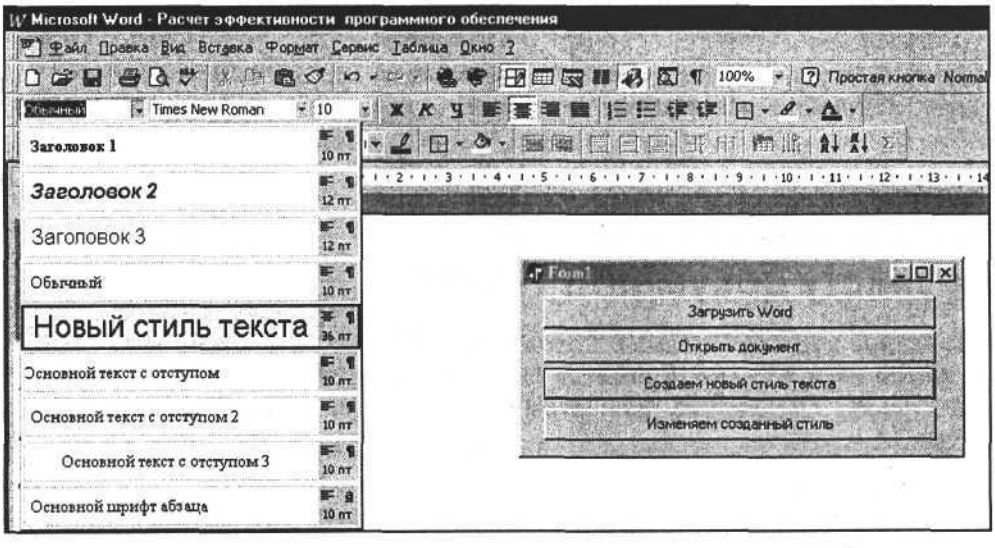


Рис. П2.10. Создаем новый стиль для использования в документе Word

Как определить координаты положения для ячейки таблицы Excel?

Координаты ячейки относительно верхнего левого угла листа рабочей книги содержатся в ее свойствах Top и Left. Чтобы получить координаты ячейки в активном окне, необходимо провести вычисления, т. е. из значений координат положения ячейки вычесть значения координат ячейки, которая в данный момент находится в верхнем левом углу окна листа рабочей книги.

Следующие процедуры определяют и отображают значения координат ячейки в окне (полный исходный текст приложения представлен на сопроводительном компакт-диске книги).

Определение координат ячейки

```
procedure TForm1.colChange(Sender: TObject);
begin
```

```
  x.Caption:=currtostr(E.ActiveWorkBook.ActiveSheet.↵
Cells[row.Value,col.Value].Left-E.ActiveWorkBook.ActiveSheet.Columns.↵
Item[E.ActiveWindow.ScrollColumn].Left);
end;

procedure TForm1.rowChange(Sender: TObject);
begin
  y.Caption:=currtostr(E.ActiveWorkBook.ActiveSheet.↵
Cells[row.Value,col.Value].Top-E.ActiveWorkBook.ActiveSheet.Rows.↵
Item[E.ActiveWindow.ScrollRow].Top);
end;
```

Как перемещать курсор по тексту документа Word?

Положение курсора связано с положением области Selection, поэтому методы MoveLeft и MoveRight объекта Selection позволяют перемещать положение курсора. Следующие процедуры позволяют перемещать курсор влево и вправо.

Перемещение курсора влево и вправо

```
const wdCharacter=1;
procedure TForm1.MoveLeftClick(Sender: TObject);
begin
  W.Selection.MoveLeft (Unit:=wdCharacter, Count:=1);
end;

procedure TForm1.MoveRightClick(Sender: TObject);
begin
  W.Selection.MoveRight(Unit:=wdCharacter, Count:=1);
end;
```

Как выделить область листа, заполненную данными?

Объект UsedRange позволяет обращаться к той области таблицы Excel, которая заполнена данными. Этот объект имеет тип Range и обладает соответствующими свойствами и методами. Метод Select объекта UsedRange позволяет выделить область ячеек, заполненную информацией. Свойство Column

возвращает количество ячеек этой области, коллекции Columns и Rows возвращают ссылки на столбцы и строки этой области.

Следующая процедура позволяет выделить область листа рабочей книги, заполненной информацией, и возвращает общее количество ячеек, количество строк и столбцов, а также интервал этой области.

Получение некоторых характеристик области, заполненной данными

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  E.ActiveWorkbook.ActiveSheet.UsedRange.Select;
  Count.Caption:=E.ActiveWorkbook.ActiveSheet.UsedRange.Count;
  ColCount.Caption:=E.ActiveWorkbook.ActiveSheet.UsedRange.Columns.Count;
  RowCount.Caption:=E.ActiveWorkbook.ActiveSheet.UsedRange.Rows.Count;
  Address.Caption:=E.ActiveWorkbook.ActiveSheet.UsedRange.Address;
end;
```

Выполнив эту процедуру, мы получим данные, которые можно использовать в приложении (рис. П2.11).

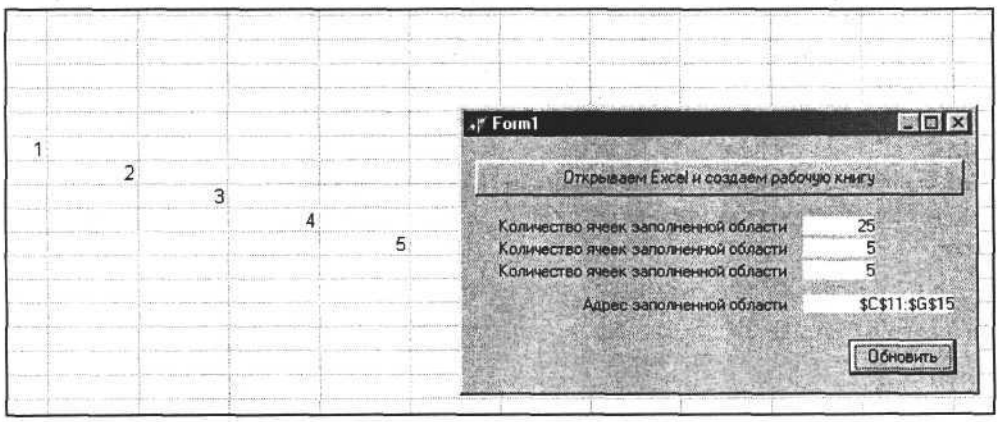


Рис. П2.11. Количественная характеристика заполненной области листа

Как вычислить адрес и размеры выделенной области?

Выделенная область ячеек описывается объектом Selection. Поэтому с помощью свойств и методов этого объекта можно определить численные характеристики и адрес выделенной области.

Следующая процедура позволяет определить количество строк и столбцов выделенной области ячеек, а также адрес верхней левой ячейки выделенной области.

Определение адреса и размеров выделенной области

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  rowscount:=E.Selection.Rows.Count;
  columncount:=E.Selection.Columns.Count;
  range_st:=E.Selection.Range['A1:A1'].Address;
end;
```

Как закрепить на экране область листа Excel?

Закрепить область листа на экране можно так: выделить необходимую область, а затем использовать свойство FreezePanes объекта ActiveWindow. Запись значения True в это свойство закрепит выбранную область.

Закрепление области листа

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  E.ActiveWorkbook.ActiveSheet.Range['A1:E5'].Select;
  E.ActiveWindow.FreezePanes:=True;
end;
```

ПРИЛОЖЕНИЕ 3

Описание компакт-диска

На сопроводительном диске представлены исходные тексты примеров, рассматриваемых в книге, и исходные тексты программ для печати платежных документов.

Для того чтобы начать работу с исходными текстами, расположенными на диске, достаточно будет вставить его в CD-привод. Программа установки запустится автоматически и предложит меню в виде списка, соответствующее главам книги, в которых используются примеры исходных текстов. После открытия списка вам достаточно выбрать пункт и нажать соответствующую кнопку. Исходный текст приложения для выбранной главы будет скопирован в соответствующий подкаталог папки *Мои документы* на жесткий диск вашего ПК. По окончании копирования папка будет открыта и вам останется только загрузить проект.

Если по каким-то причинам после установки компакт-диска в дисковод программа не запускается, то вы можете запустить ее сами. Для этого откройте папку *Programms* на компакт-диске, а затем запустите файл MYFIRSTBOOK.EXE.

Если же вы решили не использовать программу установки, а просто просмотреть содержимое диска с помощью Проводника Windows, то воспользуйтесь описанием содержимого папок сопроводительного компакт-диска, представленным ниже.

Prog03

Глава 3. Примеры доступа к Word и Excel.

Prog05

Глава 5. Создание объекта Word, запуск и визуализация приложения, создание нового или открытие ранее созданного документа, работа со списком

открытых документов, запись текста в документ, сохранение документа, закрытие документа и приложения MS Word.

Prog06

Глава 6. Поиск, копирование, вставка текста и другие функции работы с открытым документом. Свойства и методы объекта Selection. Использование шаблона документа для создания простых документов в MS Word.

Prog07

Глава 7. Формат, редактирование, положение таблицы в документе. Свойства ячеек таблицы. Шрифт. Разработка приложения с выводом информации в документ, содержащий табличную часть, на примере программы формирования счета-фактуры.

Prog08

Глава 8. Создание надписей, выносок, геометрических фигур, рисунков и других объектов в документе Word, настройка их свойств. Создание и отображение внешних объектов в документах Word. Параметры страницы. Печать. Пример создания приложения, использующего объект коллекции Shapes.

Prog09

Глава 9. Использование объекта Word.Basic для создания нового документа Word, поиска и редактирования текста, создания и редактирования таблиц, работы с внешними объектами. Печать документа. Запись документа на диск и окончание работы. Пример простого приложения.

Prog10

Глава 10. Программирование панелей и кнопок. Программирование главного меню. Создание и использование макроса Visual Basic средствами Delphi. Программирование диалогов. Пример программирования панелей.

Prog11

Глава 11. Создание объекта Excel.Application. Создание новой рабочей книги, создание рабочей книги на основе шаблона. Открытие существующей книги и получение доступа к ней. Работа с окнами и со списком листов ра-

бочей книги. Доступ к информации в ячейках рабочей книги. Сохранение, закрытие рабочей книги и приложения MS Excel.

Prog12

Глава 12. Чтение и запись данных. Формат отображения данных ячеек, запись и использование формул. Запись и чтение комментариев. Работа с областью ячеек. Поиск и замена текста. Работа со свойством текста, шрифт. Изменение свойств линий границы и области заливки ячеек. Разработка приложения "Налоговая декларация".

Prog13

Глава 13. Объект TextBox, свойства и методы, линия границы и заливка объекта. Создание выносок, линий, геометрических фигур, объектов WordArt в книге Excel, изменение их свойств.

Prog14

Глава 14. Создание диаграммы, выбор типа, определение области данных, настройка свойств диаграммы. Заголовок и другие области диаграммы, их свойства. Изменение диаграммы объемного вида. Некоторые особенности некоторых типов диаграмм. Некоторые дополнительные элементы рядов.

Prog15

Глава 15. Деление листа на страницы, разрыв страницы. Объект PageSetup, настройка свойств листа и параметров печати. Выбор и настройка принтера, выбор качества печати. Вывод на печать. Печать объектов рабочей книги Excel.

Prog16

Глава 16. Список элементов управления коллекции CommandBars; панели и кнопки. Исследование главного меню. Создание новой панели, кнопки или элемента меню. Использование объекта Visual Basic Editor в приложениях Delphi. Коллекция диалогов. Пример программирования панелей.

Prog17

Глава 17. Создание пользовательской библиотеки для работы с документами MS Office. Обработка ошибок. Создание пользовательской динамической библиотеки. Неявная загрузка модуля DLL. Явная загрузка модуля DLL.

Prog18

Глава 18. Разработка в среде Delphi динамических библиотек для использования в макросах документов Excel. Пример создания и использования библиотеки с функцией диалога выбора даты и функцией преобразования числа в его строковый эквивалент для денежных единиц.

Скопируйте файл LIBFOROF.DLL в папку, доступную для MS Excel, например *System32.* Откройте рабочую книгу LIBFOROF.XLS и работайте с макросами.

Prog19

Приложение 2. Ответы на наиболее часто возникающие вопросы при программировании контроллеров автоматизации.

Prog20

Приложение "Платежные документы". Платежное поручение, требование, инкассовое требование, счет-фактура, приходно-кассовый ордер.

В этой папке представлены полные исходные тексты нескольких проектов как примеры использования материалов книги в целом.

Список литературы

1. Елманова Н., Трепалин С., Тенцер А. Delphi 6 и технология COM (+CD) (серия "Мастер-класс"). — СПб.: Питер, 2002. — 640 с.
2. Матросов А. В. и др. MS Office XP: разработка приложений / Матросов А. В., Новиков Ф. А., Усаров Г. Е., Харитонова И. А. / Под ред. Ф. А. Новикова. — СПб.: БХВ-Петербург, 2003. — 944 с.
3. Персон Р. Excel 7.0 для Windows 95 в подлиннике: пер. с англ. — СПб.: BHV — Санкт-Петербург, 1996. — 1056 с.
4. Корняков В. Серия статей в компьютерном издании "Компьютерные Вести". — Минск, 2003.

Предметный указатель

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

А

Б

В

Г

Д

З

И

К

Л

М

Н

О

П

P

C