ЮРИЙ МАГДА

bhv

РАЗРАБОТКА ПРИЛОЖЕНИЙ Microsoft® Office 2007 в Delphi

ПРОГРАММИРОВАНИЕ ПРИЛОЖЕНИЙ MS Office: ОБЩИЕ ПОДХОДЫ

ОБРАБОТКА ТАБЛИЦ Excel В Delphi: ТЕХНОЛОГИИ ADO И DataSnap НА ПРАКТИКЕ

РЕАЛИЗАЦИЯ ОБМЕНА ДАННЫМИ МЕЖДУ MS Excel И ПОПУЛЯРНОЙ СУБД MySQL СРЕДСТВАМИ Delphi

ПРОГРАММИРОВАНИЕ ПРИЛОЖЕНИЙ MS Word В Delphi

РАБОТА С ТАБЛИЦАМИ MS Access В Delphi

Delphi И MS Outlook: ВОЗМОЖНОСТИ ОБРАБОТКИ ЭЛЕКТРОННОЙ ПОЧТЫ

PRO

ПРОФЕССИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ

Юрий Магда

РАЗРАБОТКА ПРИЛОЖЕНИЙ Microsoft® Office 2007 в Delphi

Санкт-Петербург
«БХВ-Петербург»
2009

УДК 681.3.06
ББК 32.973.26-018.2
М12

Магда Ю. С.

М12 Разработка приложений Microsoft® Office 2007 в Delphi. — СПб.: БХВ-Петербург, 2009. — 160 с.: ил. — (Профессиональное программирование)

ISBN 978-5-9775-0413-3

Рассмотрены практические аспекты разработки приложений Microsoft Office 2007/2009 в Delphi.

Описаны общие подходы к программированию приложений MS Office. Даны программные методы реализации функций MS Excel, MS Word, MS Access и MS Outlook в среде Delphi. Приведены практические примеры создания приложений, в которых задействованы основные механизмы доступа к объектам пакета Microsoft Office 2007 с помощью библиотек типов и визуальных компонентов. Существенное внимание уделено программированию обмена данными с использованием технологий ADO и DataSnap. Описаны методы передачи данных таблиц MS Excel в базы данных MySQL и дана их практическая реализация в среде Delphi.

Для программистов

УДК 681.3.06
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Евгений Рыбаков*
Зав. редакцией	*Григорий Добин*
Редактор	*Екатерина Капалыгина*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 30.07.09.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 12,9.
Тираж 1500 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.60.953.Д.005770.05.09 от 26.05.2009 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 978-5-9775-0413-3

Оглавление

Благодарности

Автор выражает огромную благодарность сотрудникам издательства "БХВ-Петербург" за подготовку материалов книги к изданию. Особая признательность жене Юлии за понимание и поддержку при подготовке рукописи.

Введение

Пакет программ Microsoft Office 2007 является одним из наиболее широко используемых программных средств для работы с офисными приложениями. Обширные возможности Microsoft Office можно примерить в приложениях, написанных с использованием популярных инструментов разработки, таких, например, как Microsoft Visual Studio .NET или Delphi 2007/2009. Это существенно улучшает функциональность таких приложений за счет использования широких возможностей программной среды Microsoft Office. Данная книга посвящена разработке приложений в среде Delphi 2007/2009, в которых используются возможности приложений Microsoft Office, таких как Word, Excel, Access и Outlook.

От аналогичных изданий книга отличается широтой охвата материала и разнообразием методик программирования, описанных и примененных при разработке демонстрационных примеров.

Книга состоит из пяти глав, краткое описание каждой из них приведено далее.

- *Глава 1. "Объектная модель Microsoft Office".*

  Материал главы посвящен обзору возможностей пакета программ Microsoft Office и базовым методикам программирования приложений, взаимодействующих с приложениями Microsoft Office.

- *Глава 2. "Обработка документов MS Excel 2007 в Delphi".*

  В главе подробно рассматриваются программные методы доступа к объектам MS Excel и манипуляции с ними.

- *Глава 3. "Обработка документов MS Word 2007 в Delphi".*

  Здесь рассмотрены вопросы обработки документов MS Word в программах, написанных в Delphi. На многочисленных примерах демонстрируют-

ся методы программного доступа и установки параметров документов Word.

- *Глава 4. "Обработка данных MS Access 2007 в Delphi".*

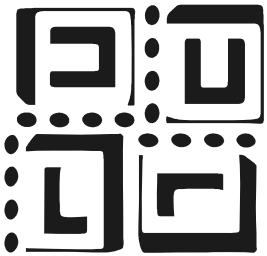
  В этой главе анализируются основные аспекты программирования таблиц баз данных приложения MS Access. Приводятся многочисленные примеры программного кода для обработки данных в среде MS Access.

- *Глава 5. "Программирование приложений MS Outlook 2007 в Delphi".*

  Материал этой главы посвящен программированию клиента электронной почты MS Outlook в среде Delphi. Многочисленные примеры программного кода демонстрируют методы программной обработки сообщений электронной почты.

Материал книги будет полезен всем желающим, которые хотели бы существенно пополнить свои знания в области разработки и программирования эффективных приложений Delphi с расширенными возможностями по обработке данных. Исходные тексты приложений, обсуждаемых в данной книге, находятся на Web-сайте **www.bhv.ru**.

ГЛАВА 1



Объектная модель Microsoft Office

Разработка программ, работающих с пакетом Microsoft Office 2007, независимо от того, в какой среде программирования они выполняются, базируется на применении так называемой *объектной модели Microsoft Office*. Если вы разрабатываете программное обеспечение с использованием VBA в самой среде Microsoft Office или создаете приложение в Visual Studio .NET или Delphi — в любом случае вы должны будете использовать элементы объектной модели Microsoft Office.

Взаимодействие с объектами в такой модели базируется на принципах OLE-автоматизации (Object Linking and Embedding, связывание и внедрение объектов), которая, в свою очередь, использует механизм межпроцессного взаимодействия, известного под аббревиатурой "COM" (Component Object Model, компонентная модель объектов), который изначально предназначался для использования в среде программирования Visual Basic, но затем стал применяться при разработке приложений, написанных на других языках программирования.

Суть этого механизма состоит в том, что приложения Windows, которые называют *контроллерами автоматизации*, могут взаимодействовать с общими для системы объектами автоматизации, экспортируемыми другими приложениями (например, Microsoft Office), и манипулировать их свойствами и методами. Механизм COM предоставляет существенно более широкие возможности по сравнению со своим предшественником, известным под названием DDE (Dynamic Data Exchange).

OLE-автоматизация использует в качестве базовых понятий термины "клиент" и "сервер". *Клиент* представляет собой приложение или процесс, который будет взаимодействовать с сервером. *Сервер*, в свою очередь, представляет собой объект, экспортируемый другим приложением.

Принципы автоматизации изначально были сориентированы на применение их в программах, написанных на языке VBA (Visual Basic for Applications) и предназначенных для использования конечным пользователем. Это очень удобно, поскольку VBA интуитивно понятен, и можно легко создавать приложения, расширяющие возможности базовых приложений, например, того же Microsoft Office.

Объекты автоматизации могут быть реализованы и с использованием популярных языков высокого уровня, например, C++, хотя для программиста здесь возникают существенные трудности. Дело в том, что синтаксис, предлагаемый C++ для реализации COM-объектов, слишком сложен (а следовательно, сложно реализовать и OLE-автоматизацию в приложении C++). В то же время, Visual Basic или Delphi намного более дружественны в этом плане. Например, Delphi позволяет избежать прямого программирования объектов автоматизации за счет использования специальных модулей (`COMObj` или `OleServer`, например), что скрывает детали взаимодействия с интерфейсами COM.

Более строго, объект автоматизации представляет собой COM-объект реализующий интерфейс `IDispatch`. Объекты автоматизации реализованы как объекты ActiveX. В то же время приложение, которое манипулирует с объектами ActiveX, называют клиентом ActiveX. Интерфейс `IDispatch` предоставляет четыре базовых метода, наиболее важным из которых является `Invoke`. Данный метод позволяет вызывать методы (функции) класса по их именам, задавая произвольное число параметров. При этом ни имя метода, ни количество параметров и их типы заранее не известны компилятору. Подобная программная техника называется *поздним связыванием* (late binding).

Для большинства COM-объектов, кроме позднего связывания можно использовать и традиционное *раннее связывание*. Принципиально, как раннее, так и позднее связывание обеспечивают один и тот же уровень функциональности для клиентов автоматизации.

Раннее связывание, в общем, требует больших усилий от программиста, но обеспечивает большую производительность. Позднее связывание несколько снижает производительность, хотя обеспечивает более высокую надежность, поскольку здесь не требуется совместимости на уровне программного кода для различных версий одного и того же компонента.

При использовании позднего связывания клиенту нужно знать только имя необходимого объекта, а также имена методов и названия параметров. При использовании раннего связывания клиенту нужно знать точные параметры вызываемого интерфейса. Серверы автоматизации могут быть реализованы для работы как в однопользовательском, так и в многопользовательском режиме. При работе в однопользовательском режиме каждый клиент получает

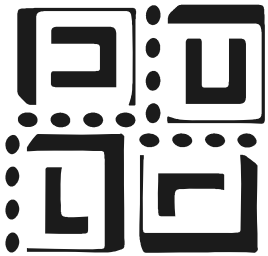
свой экземпляр сервера; в многопользовательском режиме несколько клиентов подсоединяются к единственному серверу и разделяют его программные ресурсы.

Для использования автоматизации в своем приложении (контроллере автоматизации) разработчик должен четко представлять себе объектную модель, экспортируемую сервером автоматизации. Для упрощения программирования объекты автоматизации имеют так называемые *библиотеки типов* (type libraries), которые содержат метаданные об используемых классах, интерфейсах и других параметрах, представленных в объектных библиотеках. Библиотеки типов можно просмотреть с помощью различных программных средств, например, Microsoft OLE/COM Object Viewer или программных средств, входящих в состав Microsoft Visual Studio .NET.

Фирма Microsoft предоставляет документацию по объектным моделям всех приложений Microsoft Office. Объектные модели обычно представлены для контроллеров автоматизации в форме библиотек типов. Мы будем использовать библиотеки типов наряду с поздним связыванием при разработке приложений Delphi 2007/2009, работающих с приложениями Microsoft Excel, Word и Outlook.

При разработке приложений в среде Delphi мы будем часто обращаться к программной реализации взаимодействия клиента и сервера автоматизации посредством VBA. Использование коротких примеров, написанных на Visual Basic на одном из работающих приложений Word, Excel или Outlook, существенно упрощает и программирование в Delphi за счет осмысления механизмов работы автоматизации и понимания объектной модели того или иного приложения Microsoft Office.

ГЛАВА 2



Обработка документов MS Excel 2007 в Delphi

В этой главе мы рассмотрим различные методы доступа и обработки данных приложений Microsoft Excel 2007 средствами программной среды Delphi. Для разработки приложений будем использовать программные средства общие как для Delphi 2007, так и для Delphi 2009. Вначале кратко ознакомимся с возможностями Excel 2007, которые существенно расширились по сравнению с предыдущими версиями этого пакета.

Новый ориентированный на результат интерфейс пользователя значительно облегчает работу в Microsoft Office Excel. Команды и функции, которые часто были спрятаны в сложных меню и панелях инструментов, теперь проще найти на проблемно-ориентированных вкладках, содержащих логические группы команд и функций. Множество диалоговых окон заменены раскрывающимися коллекциями, которые отображают доступные параметры, а наглядные подсказки или демонстрационные примеры помогают в выборе нужного параметра.

Для обработки больших объемов данных в Microsoft Excel 2007 включена поддержка рабочих листов размером до одного миллиона строк и шестнадцати тысяч столбцов. При этом сетка Excel включает 1 048 576 строк и 16 384 столбцов, нумерация которых оканчивается на XFD (IV в Excel 2003). Если в более ранних версиях Microsoft Excel число различных способов форматирования ячеек ограничивалось четырьмя тысячами, то в версии 2007 таких ограничений не существует. Кроме того, максимальное число ссылок на ячейки, равное восьми тысячам в более ранних версиях Excel, теперь ограничивается только объемом доступной памяти.

В Microsoft Excel 2007 значительно улучшен механизм управления памятью, позволяющий использовать до 2 Гбайт оперативной памяти, что в два раза больше, чем в Excel 2003. Что же касается вычислительных алгоритмов, то для Excel 2007 разработаны алгоритмы, позволяющие повысить скорость вы-

числений в больших листах с множеством формул. Обработка данных в таблицах улучшена за счет использования функциональных возможностей, описанных далее.

- Допускается включать или отключать строки заголовка таблицы; отображаемые заголовки таблицы всегда остаются видимыми для данных в столбцах при перемещении по длинной таблице.
- Вычисляемый столбец использует одну формулу, которая применяется к каждой строке. При этом он автоматически расширяется на дополнительные строки, распространяя на них формулу — это позволяет всего лишь один раз ввести формулу.
- Автофильтрация включена по умолчанию — это дает возможность применять мощные средства сортировки и фильтрации табличных данных.
- Допускается использовать в формулах имена заголовков столбцов таблицы вместо ссылок на ячейки.
- Допускается использование пользовательских формул и текста в строке итоговых значений.
- Включены возможности быстрого форматирования таблиц на уровне профессионального дизайнера с применением стилей таблицы.

Кроме рассмотренных возможностей нужно отметить, что в Microsoft Excel 2007 включены новые инструменты для работы с диаграммами, упрощающие создание профессиональных диаграмм с эффективным представлением данных.

Существенные изменения претерпели и форматы файлов Microsoft Excel 2007. Основным стандартным форматом рабочих книг Excel теперь является основанный на XML формат файлов XLSX, хотя по-прежнему можно использовать стандартный формат XLS, совместимый с предыдущими версиями. Кроме того, можно использовать и другие форматы файлов на основе XML, например, формат XLSM, обеспечивающий поддержку макросов, формат XLTX, используемый в шаблонах, или формат XLTM, применяемый в шаблонах с поддержкой макросов.

Для манипуляций с документами Microsoft Excel программными методами очень важно иметь представление об объектной модели Microsoft Excel. Независимо от того, в какой среде создаются программы (VBA Excel, Delphi или Visual Studio .NET), разработчику придется иметь дело с объектами Microsoft Excel. Ознакомимся с этой моделью более подробно и рассмотрим простые примеры манипуляций с объектами Excel, их свойствами и методами. Вначале проанализируем несколько простых программ, разработанных в среде VBA приложения Excel — это поможет лучше понять, как работать с объектами Excel, и окажет существенную помощь при разработке приложений в

среде Delphi. Для этого не понадобится изучать Visual Basic, поскольку все примеры, которые мы рассмотрим, будут наглядными и интуитивно понятными программисту Delphi.

Иерархия стандартных объектов в Excel базируется на следующих основных объектах: `Application` (приложение), `Workbook` (рабочая книга), `Worksheet` (рабочий лист) и `Range` (диапазон). Все объекты этой модели обладают определенными свойствами и могут выполнять определенные действия, вызывая определенные методы. На самом верхнем уровне этой модели находится объект `Application`, который представляет все приложение Excel. Если, например, требуется вызвать Excel из другого приложения, то нужно создавать объект `Excel.Application`.

Следующим по иерархии является `Workbook`, представляющий книгу Microsoft Excel. Объект `Workbook` нужен для получения ссылки на необходимую нам книгу в наборе открытых книг Excel, а также для настройки общих свойств и выполнения общих действий со всеми листами книги. Получить доступ к этому объекту можно через свойство `Workbooks` объекта `Application`, выбрав нужную книгу коллекции `Workbooks` по имени или номеру в коллекции.

Для ввода данных в таблицу Excel служит третий в иерархии объект `Worksheet` (рабочий лист). Объекты `Worksheet` в рабочей книге объединены в коллекцию `Sheets`, из которой можно выбрать нужный лист для ввода данных. В некоторых случаях вначале требуется создать лист, а затем выбрать его для ввода данных.

Нижним в объектной модели Excel является объект `Range`. Данный объект может представлять одну или несколько ячеек таблицы, причем может включать и несмежные ячейки, а также целый лист.

Кроме указанных, существует еще один специальный тип объекта — `Chart`, входящий в коллекцию `Charts`. Этот объект позволяет на основе определенных данных строить диаграммы и графики, что широко используется при анализе данных. Приложение Excel позволяет манипулировать и с целым рядом других, более специализированных объектов, которые здесь рассматриваться не будут. В практическом программировании чаще остальных используется еще один тип объектов, которые называются *коллекциями* (collections). Коллекция представляет собой группу объектов, принадлежащих к одному и тому же типу, и в объектной модели Excel сама является объектом. К наиболее часто используемым коллекциям относятся:

- `Workbooks` — коллекция открытых в данный момент рабочих книг, каждая из которых является объектом `Workbook`;
- `Worksheets` — коллекция всех рабочих листов (объекты `Worksheet`), содержащихся в отдельной рабочей книге, представленной объектом `Workbook`;

- `Charts` — коллекция всех графиков и диаграмм (объекты `Chart`), содержащихся в отдельной рабочей книге, представленной объектом `Workbook`;
- `Sheets` — коллекция всех рабочих листов, независимо от их типа, содержащихся в отдельной рабочей книге, представленной объектом `Workbook`.

Все основные объекты Microsoft Excel, как правило, включают еще целый ряд объектов, которые, в свою очередь, могут включать другие объекты и т. д. Все объекты модели обладают определенными свойствами и могут выполнять определенные действия посредством методов. Методы объекта, кроме выполнения каких-то действий, могут изменять и свойства самого объекта. В среде Microsoft Excel имеется множество объектов с их свойствами и методами, позволяющими выполнять самые разные манипуляции с данными. На практике большая часть манипуляций с данными выполняется с помощью довольно ограниченного круга объектов и их свойств и методов. Кратко ознакомимся с методами обработки данных в среде VBA с применением объектной модели.

Вначале настроим среду разработки приложения Excel. Для демонстрации примеров нужно войти в Visual Basic, который будет доступен, если на ленте (ribbon) инструментов будет присутствовать вкладка **Разработчик**. Добавить эту вкладку просто: нужно нажать кнопку **Microsoft Office** в левом верхнем углу окна приложения Excel и в раскрывшемся окне справа внизу нажать кнопку **Параметры Excel**. В раскрывшемся окне установим опцию **Показывать вкладку "Разработчик"** на ленте (рис. 2.1).

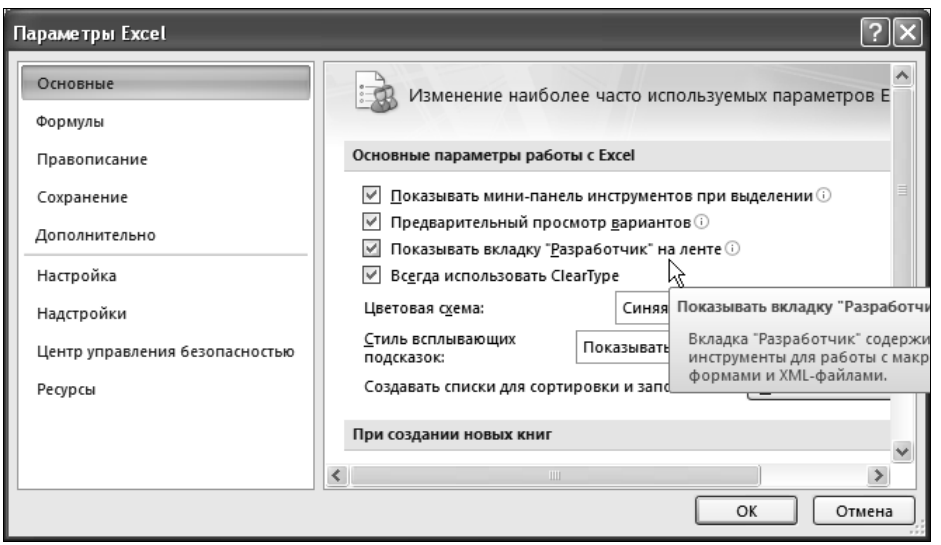


Рис. 2.1

После нажатия кнопки **OK** на ленте появится меню **Разработчик**, в котором мы выберем опцию **Visual Basic**, что позволит начать работу в программной среде VBA (рис. 2.2).

После выбора опции **Visual Basic** открывается окно редактора VB (рис. 2.3).

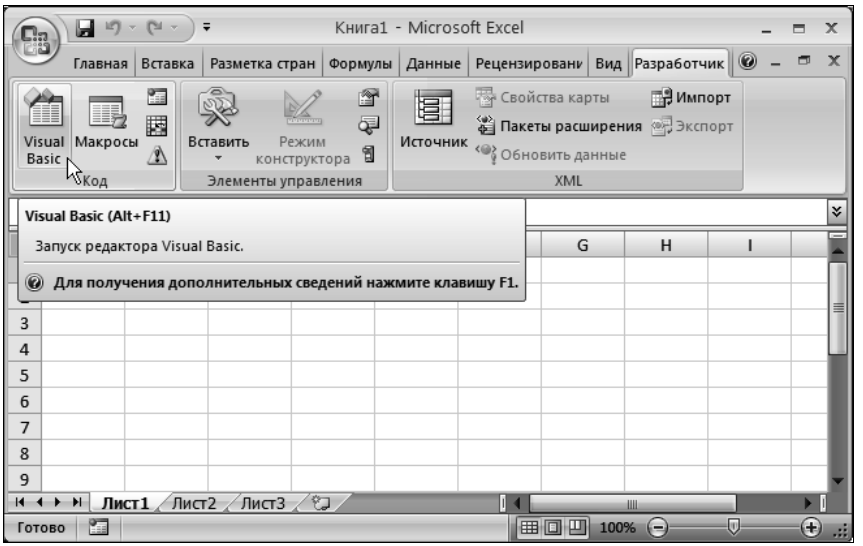


Рис. 2.2

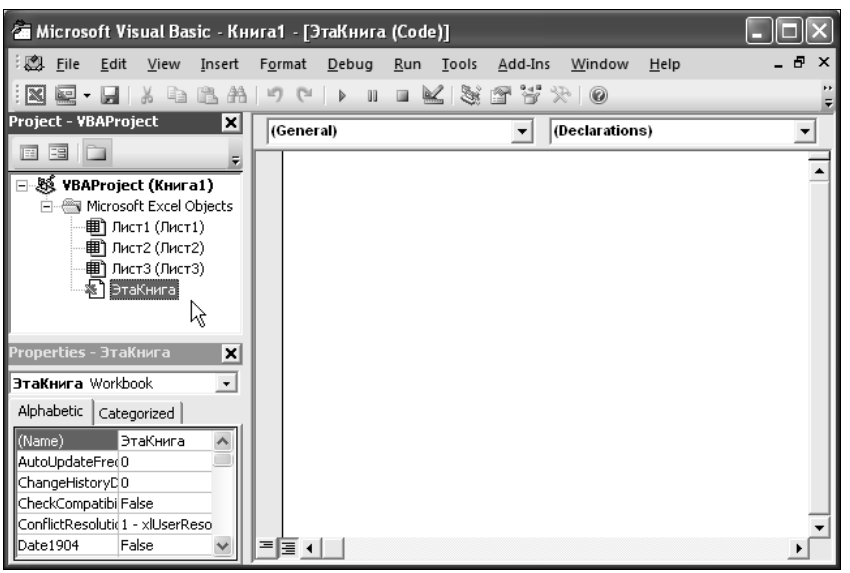


Рис. 2.3

Слева в окне редактора располагаются окна Project Explorer и окно свойств. В окне Project Explorer отображается структура приложения Excel (объект Application), в которую входит объект Workbook (ЭтаКнига) и объекты Worksheet (три рабочих листа Лист1—Лист3). Теперь напишем простейший программный код, выполняющий вывод текстовой строки в ячейку A2 рабочего листа Лист1. На примере этого кода проанализируем особенности объектной модели Excel, что пригодится нам при разработке приложений в среде Delphi.

При разработке приложений в среде Microsoft Office большая часть программного кода реализована в виде процедур. Напишем процедуру с именем FillCell, для чего сделаем двойной щелчок на объекте ЭтаКнига и в поле редактирования введем следующий код:

```
Sub FillCell()
 Worksheets(1).Range("A2").Font.Bold = True
 Worksheets("Лист1").Range("A2").Value = "Текстовая строка"
End Sub
```

Для запуска процедуры нужно нажать клавишу <F5>, при этом курсор должен находиться внутри текста процедуры. После выполнения процедуры ячейка A2 рабочего листа Лист1 будет содержать текст, отображаемый жирным шрифтом.

На этом примере проанализируем особенности программирования Excel-приложений. Как видно из примера, программный код процедуры состоит из двух операторов. Первый из них устанавливает жирный текст в ячейке A2 рабочего листа Лист1, а второй записывает в эту же ячейку текстовую строку. В иерархии объектов для доступа к объектам (в первом операторе это объект Font, а во втором — Range) используется точечная нотация. Большинство объектов идентифицируется или порядковым номером, или именем. В нашем примере доступ к первому листу в первом операторе осуществляется по его порядковому номеру в коллекции (в данном случае первый лист имеет номер 1). Во втором операторе доступ к этому же листу осуществляется по его имени. Если операции выполняются на активном в данный момент листе, то часть ссылок на объекты более высокого уровня в иерархии можно опустить.

Так если активным листом является Лист1, то процедура FillCell может быть записана следующим образом:

```
Sub FillCell()
  Range("A2").Font.Bold = True
  Range("A2").Value = "Текстовая строка"
End Sub
```

При работе с данными таблиц Microsoft Excel часто возникают ошибки, связанные с тем, что программа манипулирует с данными в предположении, что

они расположены на определенном листе, в то время как активный рабочий лист может быть совсем другим. Во избежание таких ошибок нужно явным образом активизировать нужный лист, после чего можно использовать и сокращенную запись операторов программы. Вот следующий пример:

```
Sub FillCell()
 Excel.Worksheets("Лист2").Activate
 ActiveSheet.Range("A2").Font.Bold = True
 Range("A2").Value = "ТЕКСТ на Листе 2"
End Sub
```

В этой процедуре вначале активизируется `Лист2`, а затем ячейка этого листа `A2` заполняется нужным значением. В данном случае можно исключить верхние уровни иерархии объектов во втором и третьем операторе процедуры `FillCell`.

Разберем более подробно свойства и методы объектов Microsoft Excel. Свойства можно рассматривать как определенные параметры объекта, причем имя объекта и свойство разделяются точкой. Например, свойствами объекта `Range` являются `Value` и `Address`. Свойствам можно присваивать определенные значения, как это было сделано в предыдущих примерах и свойства можно читать, присваивая их значения переменным программы. Например, в процедуре

```
Sub FillCell()
 Worksheets("Лист1").Range("A1").Value = "Текст"
  textString = Worksheets(1).Range("A1").Value
 MsgBox "You entered: " & textString
End Sub
```

переменной `textString` присваивается значение `Текст`, которое затем выводится на экран.

Объекты кроме свойств могут выполнять определенные действия посредством методов. Например, метод `Activate` из предыдущих примеров активизирует рабочий `Лист2`. При написании программы, работающей с объектами Microsoft Excel, полезно иметь под рукой список всех объектов, их свойств и методов. Для этого в среде VBA можно вызвать Object Browser (опция **Object Browse** в меню **View** или клавиша <F2>), затем в раскрывшемся окне в строке поиска нужно указать имя для поиска. Вот как, например, будет выглядеть окно Object Browser для объекта `Worksheet` — рис. 2.4.

Object Browser оказывается весьма полезным при разработке Delphi-приложений, когда нужно определить какой объект, свойство или метод следует использовать, особенно учитывая то, что Delphi не дает никаких подсказок по объектам Excel. Следует заметить, что манипуляции с объектами Microsoft

Excel можно выполнять различными способами, используя для этого другие объекты, их свойства и методы.

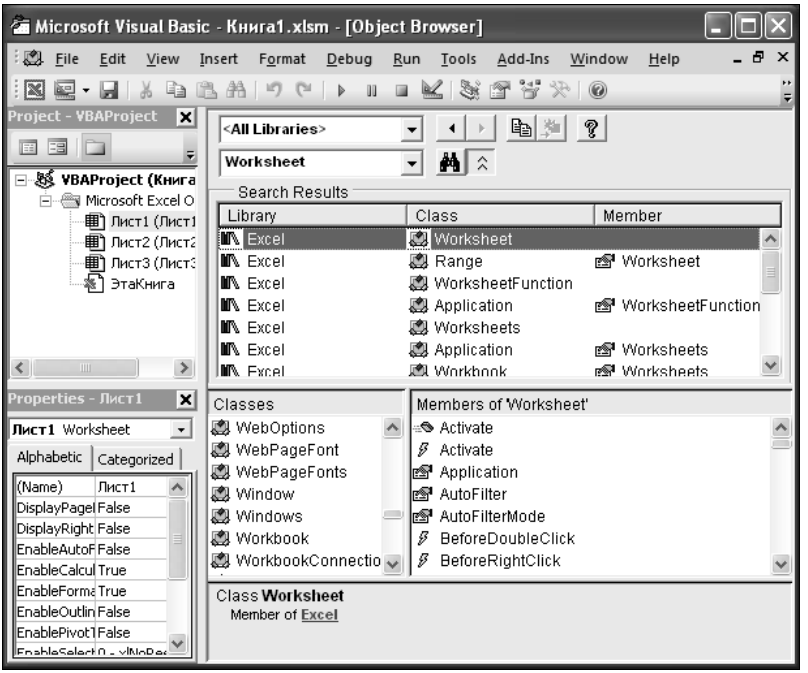


Рис. 2.4

Приложение Delphi может получить доступ к объектам Excel для последующей обработки данных несколькими способами. В данной главе мы рассмотрим два наиболее распространенные из них: использование объектов `Variant` и технология ADO.

2.1. Использование объектов *Variant* для доступа к данным Excel

Один из методов доступа и обработки данных Excel базируется на применении объектов автоматизации OLE. В свою очередь, при использовании OLE можно выбрать один из двух возможных вариантов: применить класс `IDispatch` посредством переменных типа `Variant` или использовать интерфейсы COM. Несмотря на тесную взаимосвязь этих вариантов, между ними есть определенные различия. Использование переменных типа `Variant` упрощает программирование, но несколько замедляет работу приложения. Если же применить COM-интерфейсы, то производительность улучшится, но это по-

требует определенных усилий при программировании. Рассмотрим применение обоих методов на практических примерах и начнем с использования переменных `Variant`.

Создадим графическое приложение в Delphi и поместим на форму две кнопки (рис. 2.5).

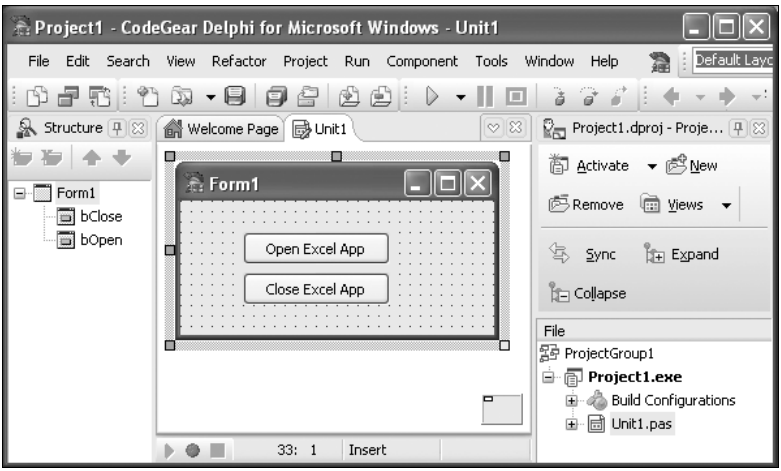


Рис. 2.5

При нажатии кнопки **Open Excel App** будет создаваться новое приложение Excel, содержащее рабочую книгу с тремя рабочими листами, а при нажатии кнопки **Close Excel App** приложение Excel будет закрыто. Исходный текст приложения Delphi приводится в листинге 2.1.

Листинг 2.1. Открытие и закрытие нового приложения Excel

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms, Dialogs, StdCtrls, COMObj;

type
  TForm1 = class(TForm)
    bOpen: TButton;
    bClose: TButton;
    procedure bOpenClick(Sender: TObject);
    procedure bCloseClick(Sender: TObject);
```

```
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  XLApp: OleVariant;

implementation

{$R *.dfm}

procedure TForm1.bCloseClick(Sender: TObject);
begin
 if not VarIsEmpty(XLApp) then
   XLApp.Quit;
end;

procedure TForm1.bOpenClick(Sender: TObject);
begin
 XLApp:= CreateOleObject('Excel.Application');
 XLApp.Visible:= True;
 XLApp.Workbooks.Add;
end;
end.
```

Для доступа к объекту Microsoft Excel в этом приложении используется переменная `XLApp` типа `OleVariant`. Тип `System::OleVariant` наследует класс `Variant`, который, как мы уже знаем, используется для реализации интерфейсов COM или данных, представляемых этим интерфейсом. `OleVariant` наследует `Variant`, но дополнительно запрещает операции с объектами типа `AnsiString`, которые несовместимы с COM. Синтаксис переменных `OleVariant` такой же, как и в `Variant`.

При нажатии кнопки `bOpen` (**Open Excel App**) с помощью функции `CreateOleObject` создается приложение Excel, все манипуляции с которым будут впоследствии выполняться с помощью переменной `XLApp`. Для того чтобы пользователь мог увидеть работающее Excel-приложение, свойство `Visible` следует установить в `True`. Последний оператор обработчика нажатия кнопки добавляет к вновь созданному приложению рабочую книгу. Для за-

крытия приложения Excel необходимо нажать кнопку `bClose` — при этом приложение Delphi закроет Excel перед завершением работы. С помощью функции `VarIsEmpty` проверяется переменная `XLApp`.

Наше приложение пока ничего не делает, кроме создания и закрытия Excel. На следующих шагах мы модифицируем приложение так, чтобы можно было, например, работать с рабочими листами и ячейками Excel.

Следующий пример показывает, как открыть рабочую книгу Microsoft Excel, сохраненную в файле e:\myparts.xls (в таблице содержатся данные по электронным компонентам, и мы часто будем ее использовать в качестве демонстрационной). Создадим графическое приложение, в котором при его запуске будет открываться рабочая книга, а при завершении работы — закрываться. Таким образом, в этом приложении понадобится написать два обработчика событий. Исходный текст программы приведен в листинге 2.2.

Листинг 2.2. Открытие книги Excel из существующего файла

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms, Dialogs, COMObj;

type
  TForm1 = class(TForm)
    procedure FormCreate(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  XLApp, WBook: OleVariant;

implementation

{$R *.dfm}
```

```
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
 if not VarIsEmpty(XLApp) then
   XLApp.Quit;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
 XLApp:= CreateOleObject('Excel.Application');
 XLApp.Visible:= True;
 WBook:= XLApp.Workbooks.Open('e:\myparts.xls');
 end;

end.
```

В этом приложении при его запуске инициируется событие `OnCreate`, в обработчике которого `FormCreate` создается переменная `XLApp` типа `OleVariant`, идентифицирующая объект Excel, затем выполняется привязка рабочей книги, содержащейся в файле myparts.xls к переменной `WBook` типа `OleVariant`. Таким образом, будет открыта существующая рабочая книга Excel с активизацией первого листа (по умолчанию). В дальнейшем все манипуляции с рабочей книгой можно осуществлять посредством переменной `WBook`, используя соответствующие свойства и методы.

В обработчике `FormClose` осуществляется закрытие приложения Excel. Для того чтобы при открытии рабочей книги активизировать другой лист, чем назначенный по умолчанию, нужно явным образом указать рабочий лист, используя его имя или порядковый номер в коллекции. Например, для активизации `Листа2` исходный текст обработчика `FormCreate` можно изменить так:

```
procedure TForm1.FormCreate(Sender: TObject);
var
 WSheet: OleVariant;
begin
 XLApp:= CreateOleObject('Excel.Application');
 XLApp.Visible:= True;
 WBook:= XLApp.Workbooks.Open('e:\myparts.xls');
 WSheet:= XLApp.Worksheets['Лист2'];
 Wsheet.Activate;
end;
```

Здесь для удобства мы вводим новую переменную `WSheet` типа `OleVariant` для ссылки на `Лист2`. Эту переменную можно затем использовать при последую-

щих обращениях к данному объекту. В данном случае активизация Листа2 осуществляется операторами

```
WSheet:= XLApp.Worksheets['Лист2'];
Wsheet.Activate;
```

Если не планируется использование объекта Worksheet, то можно обойтись одним оператором

```
XLApp.Worksheets['Лист2'].Activate;
```

Вместо имени можно использовать порядковый номер рабочего листа в коллекции Worksheets:

```
XLApp.Worksheets[2].Activate;
```

В большинстве случаев программа, открывающая рабочую книгу Excel, должна выполнять какие-либо манипуляции с отдельными ячейками рабочего листа. Для обращения к отдельным ячейкам или группе ячеек можно применять несколько методов. Один из них — использовать объект Range коллекции Worksheets. Следующий фрагмент программного кода показывает, как записать в ячейку B2 активного рабочего Листа2 текстовую строку:

```
procedure TForm1.FormCreate(Sender: TObject);
var
 WSheet: OleVariant;
begin
 XLApp:= CreateOleObject('Excel.Application');
 XLApp.Visible:= True;
 WBook:= XLApp.Workbooks.Open('e:\myparts.xls');
 Wsheet:= XLApp.Worksheets['Лист2'];
 Wsheet.Activate;
 WSheet.Range['B1'].Value:= 'Текст в ячейке B1';
end;
```

Здесь для записи в ячейку B1 активного Листа2 текстового значения используется свойство Value объекта Range. Запись данных осуществляется при запуске приложения. Ниже показан альтернативный вариант записи данных в ячейку A3 для неактивного рабочего Листа3:

```
procedure TForm1.FormCreate(Sender: TObject);
var
 WSheet: OleVariant;
begin
 XLApp:= CreateOleObject('Excel.Application');
 XLApp.Visible:= True;
 WBook:= XLApp.Workbooks.Open('e:\myparts.xls');
 XLApp.Worksheets['Лист3'].Range['A3'].Value:= 'Текст в ячейке A3';
end;
```

Данные в отдельную ячейку можно записать и другим способом, используя свойство Cells рабочего листа. Исходный текст программного кода, где применяется Cells, показан далее:

```
procedure TForm1.FormCreate(Sender: TObject);
var
 XLApp, WBook, WSheet: OleVariant;
begin
 XLApp:= CreateOleObject('Excel.Application');
 XLApp.Visible:= True;
 WBook:= XLApp.Workbooks.Open('e:\myparts.xls');
 WSheet:= XLApp.Worksheets['Лист3'];
 WSheet.Cells.Item[1,1].Value:= 'Текст в ячейке A1';
end;
```

Для адресации ячейки используется формат [Строка, Столбец]. Здесь Строка представляет собой индекс строки, а Столбец — индекс столбца, на пересечении которых расположена ячейка. В данном примере при запуске приложения осуществляется переход на рабочий Лист3 и в ячейку A1, которая имеет координаты [1,1], записывается текстовая строка Текст в ячейке A1. Подобную адресацию очень удобно использовать при заполнении группы ячеек значениями из массива или при присвоении группе ячеек определенного значения. Следующий пример показывает, как инициализировать ячейки A1—A10 произвольными числовыми значениями. Исходный текст приложения Delphi показан в листинге 2.3.

Листинг 2.3. Инициализация ячеек произвольными числами

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms, Dialogs, COMObj, StdCtrls;

type
  TForm1 = class(TForm)
    Button1: TButton;
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
```

```
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  XLApp, WBook, WSheet: OleVariant;

implementation

{$R *.dfm}
procedure InitCells();
var
 arr1: array [1..10] of Integer;
 i1: Integer;
begin
 for i1 := 1 to 10 do
   arr1[i1]:= Random(100);

 XLApp:= CreateOleObject('Excel.Application');
 XLApp.Visible:= True;
 XLApp.Workbooks.Add;
 WSheet:= XLApp.Worksheets['Лист3'];
 WSheet.Activate;
 for i1 := 1 to 10 do
  WSheet.Cells.Item[i1,1].Value:= arr1[i1];
 XLApp.Workbooks[1].SaveAs('e:\test.xls');
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
 InitCells();
end;

procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
 if not VarIsEmpty(XLApp) then
   XLApp.Quit;
end;
end.
```

На главной форме приложения размещена кнопка `Button`, при ее нажатии вызывается обработчик `Button1Click`, функция `InitCells` которого и выполняет

все необходимые действия. Первый цикл `for` функции `InitCells` инициализирует элементы массива `arr1` произвольными числовыми значениями. Во втором цикле `for` ячейкам `A1`—`A10` рабочего `Листа3` последовательно присваиваются значения элементов массива. При этом ячейке в `i1`-й строке присваивается значение `i1`-го элемента массива `arr1`.

Для записи формулы в какую-либо ячейку также существует несколько способов. Можно, например, использовать свойство `Formula` объекта `Range`, как в показанной ниже процедуре `AddA1A2`, выполняющей суммирование содержимого ячеек `A1` и `A2` с записью результата в ячейку `A3`:

```
procedure AddA1A2();
var
 XLApp, WBook, WSheet: OleVariant;
begin
 XLApp:= CreateOleObject('Excel.Application');
 XLApp.Visible:= True;
 XLApp.Workbooks.Add;

 if XLApp.ActiveSheet.Name <> 'Лист1' then
   XLApp.Worksheets['Лист1'].Activate;
 XLApp.ActiveSheet.Range['A3'].Formula:= '=SUM(A1, A2) * 3';
 XLApp.Workbooks[1].SaveAs('e:\test.xls');
end;
```

Здесь вначале создается документ Excel, в который добавляется рабочая книга. Оператор `if` написан с демонстрационной целью и показывает, как проверить, какой рабочий лист является активным (проверяется имя). Затем в ячейку `A3` активного листа записывается формула, для чего свойству `Formula` присваивается текстовая строка с формулой, т. к. она должна быть указана в окне формул рабочего листа. Вместо оператора

```
XLApp.ActiveSheet.Range['A3'].Formula:= '=SUM(A1, A2) * 3';
```

можно применить другую запись:

```
XLApp.Worksheets[1].Cells.Item[3,1].Formula := '=SUM(A1, A2) * 3';
```

Следующие несколько примеров демонстрируют выполнение операций перемещения/копирования данных на одном и том же рабочем листе. Первый пример из этой серии показывает копирование содержимого одной ячейки в другую. Исходный текст процедуры с именем `CopyPaste`, выполняющей эту операцию, показан далее:

```
procedure CopyPaste();
var
 XLApp, WSheet: OleVariant;
```

```
begin
 XLApp:= CreateOleObject('Excel.Application');
 XLApp.Visible:= True;
 XLApp.Workbooks.Add;

 WSheet:= XLApp.Worksheets[1];
 if XLApp.ActiveSheet.Name <> 'Лист1' then
   WSheet.Activate;

 WSheet.Range['C1'].Select;
 WSheet.Range['C1'].Value:= 'СТРОКА 1';
 XLApp.Selection.Copy;
 WSheet.Range['D1'].Select;
 XLApp.ActiveSheet.Paste;
 XLApp.CutCopyMode:= False;
end;
```

Копирование выполняется на рабочем `Листе1`, который становится активным (если не был таковым) после выполнения оператора `WSheet.Activate`. Далее выполняется копирование содержимого ячейки `C1` в ячейку `D1`. Перед копированием в ячейку `C1` записывается текстовая строка, которая будет помещена в ячейку `D1`. Для копирования используется метод `Copy`, а для вставки — метод `Paste`. Последний оператор процедуры отменяет режим копирования (визуально это проявляется в том, что снимается выделение с копируемой ячейки).

Для выполнения перемещения содержимого одной ячейки в другую можно использовать следующий программный код, показанный в процедуре `CutPaste`:

```
procedure CutPaste();
var
 XLApp, WSheet: OleVariant;
begin
 XLApp:= CreateOleObject('Excel.Application');
 XLApp.Visible:= True;
 XLApp.Workbooks.Add;

 WSheet:= XLApp.Worksheets[1];
 if XLApp.ActiveSheet.Name <> 'Лист1' then
   WSheet.Activate;

 WSheet.Range['C1'].Value:= 'СТРОКА 1';
 WSheet.Range['C1'].Cut;
 WSheet.Range['D1'].Select;
 XLApp.ActiveSheet.Paste;
```

```
 XLApp.CutCopyMode:= False;
 end;
```

Здесь содержимое ячейки `C1` активного `Листа1` помещается в буфер обмена, а затем вставляется в ячейку `D1`. Содержимое ячейки `C1` в результате операции будет потеряно. Для копирования содержимого исходной ячейки в буфер обмена используется метод `Cut`, а для вставки — метод `Paste`. Обратите внимание, к каким объектам относятся методы `Copy`, `Cut` и `Paste`.

Для копирования содержимого группы ячеек можно воспользоваться программным кодом, представленным в процедуре `CopyGroup`:

```
procedure CopyGroup();
var
 XLApp, WSheet: OleVariant;
begin
 XLApp:= CreateOleObject('Excel.Application');
 XLApp.Visible:= True;
 XLApp.Workbooks.Add;
 WSheet:= XLApp.Worksheets[1];
 if XLApp.ActiveSheet.Name <> 'Лист1' then
   WSheet.Activate;

 WSheet.Range['A1', 'A6'].Select;
 XLApp.Selection.Copy;

 WSheet.Range['C1', 'C6'].Select;
 XLApp.ActiveSheet.Paste;
 XLApp.Application.CutCopyMode:= False;
end;
```

Здесь с помощью оператора `WSheet.Range['A1', 'A6'].Select` вначале выбирается требуемый диапазон ячеек, содержимое которых будет скопировано. Следующий за ним оператор копирует данные в буфер обмена. Затем с помощью оператора `WSheet.Range['C1', 'C6'].Select` выбирается диапазон ячеек назначения, в данном случае `C1`—`C6`. Наконец, оператор `XLApp.ActiveSheet.Paste` помещает данные из буфера обмена в ячейки `C1`—`C6`. Последний оператор процедуры отменяет режим копирования/вставки.

Если необходимо переместить содержимое группы ячеек в другую группу ячеек, то достаточно в процедуре `CopyGroup` заменить оператор

```
XLApp.Selection.Copy;
```

на

```
XLApp.Selection.Cut;
```

В этом случае данные в исходных ячейках `A1`—`A6` будут потеряны.

Во всех рассмотренных операциях перемещения/копирования данных мы работали с содержимым активного рабочего листа (в данном случае, это был `Лист1`). Для выполнения таких же операций, но на нескольких листах потребуется немного изменить исходный текст примеров. Мы рассмотрим один пример перемещения данных между рабочими листами `Лист1` и `Лист3`. За основу возьмем исходный текст процедуры `CopyGroup` и модифицируем его соответствующим образом. Новая процедура (назовем ее `Cut1Paste2`) будет иметь следующий исходный текст:

```
procedure Cut1Paste2();
var
 XLApp, WSheet: OleVariant;
begin
 XLApp:= CreateOleObject('Excel.Application');
 XLApp.Visible:= True;
 XLApp.Workbooks.Add;
 WSheet:= XLApp.Worksheets[1];
 if XLApp.ActiveSheet.Name <> 'Лист1' then
   WSheet.Activate;

 WSheet.Range['A1', 'A6'].Select;
 XLApp.Selection.Cut;

 WSheet:= XLApp.Worksheets['Лист3'];
 WSheet.Activate;
 WSheet.Range['D1', 'D6'].Select;
 XLApp.ActiveSheet.Paste;
 XLApp.Application.CutCopyMode:= False;
end;
```

Здесь содержимое ячеек `A1`—`A6` активного `Листа1` перемещается в ячейки `D1`—`D6` `Листа3`, который становится активным. Читатели без труда смогут проанализировать этот код самостоятельно.

Следующая группа примеров посвящена программированию различных свойств ячеек (цвета фона, цвета и типа шрифта и т. д.) рабочего листа. Цвет фона ячейки можно задать несколькими способами. Например, для активной (выделенной) ячейки зеленый цвет можно установить следующим образом:

```
procedure FillCell();
var
 XLApp, WSheet: OleVariant;
begin
 XLApp:= CreateOleObject('Excel.Application');
```