

Microsoft Office Automation with Visual FoxPro



*Tamar E. Granor &
Della Martin*

*Edited by
Ted Roche*

Hentzenwerke Publishing

Автоматизация Microsoft Office с Visual FoxPro

Тамар Э. Гранор и Делла Мартин

Перевод Михаил Леменёв

2000 г.

Оглавление

Введение	11
Глава 1. Знакомство с Автоматизацией	16
1.1 Если все, что у вас есть, – это молоток, значит, все становится гвоздем?	16
1.2 Краткая история Автоматизации	16
1.3 OLE, ActiveX, COM или что?	17
1.4 Использование Автоматизации на практике.	17
1.5 Куда идти дальше?.....	18
Глава 2. Офисные серверы.....	20
2.1 Изучение серверов Office	20
Прочтите подробное руководство (RTFM).....	20
Пусть сервер напишет код	22
Объекты по умолчанию	25
Именованные параметры.....	25
Определенные константы	25
Советы по записи макросов	26
Рассмотрим обозреватель объектов	27
Что просматривает браузер?	28
По вашей команде	29
Интернет-ресурсы и печатные ресурсы	31
2.2 Собираем коллекцию.....	32
Изменение коллекции	33
2.3 Самоссылающиеся объектные модели.....	33
2.4 Двигаемся дальше	35
Глава 3. Visual FoxPro как клиент Автоматизации	36
3.1 Управление серверами	36
Отображение серверов Office	37
Мы уже на месте?	39
3.2 Работа с серверами	39
SET OLEOBJECT	39
Используйте WITH ... ENDWITH	40
Используйте переменные для ссылок на объекты	41
Цикл с FOR EACH	41
3.3 Отладка	42
Работа в командном окне	42
Коллекции и объекты	42
Это работает, но это не так.....	42

Проблемы с параметрами	42
Бесполезные сообщения об ошибках	43
Используйте базу знаний	43
Поспрашивать	44
3.4 Время для кода	44
Глава 4. Основы Word	45
4.1 Объектная модель Word	45
4.2 Переходим к Word	46
4.3 Управляющие документы	47
4.4 Доступ к частям документа	48
Манипулирование текстом	49
Перемещение в диапазоне или выделенном фрагменте	51
Закладки	54
4.5 Форматирование	54
Настройка страниц	54
Настройка шрифтов	58
Форматирование абзацев	61
Работа со стилями	64
Создание пользовательских стилей	65
Границы и штриховка	69
Создание границ	69
Затенение текста	73
Верхние и нижние колонтитулы	74
Нумерация страниц	75
Организация текста со списками	77
Определение структуры списка	77
Создание списков	80
4.6 Вывод данных	83
Печать	83
Создание конвертов	84
Предварительный просмотр печати	85
4.7 Собираем все вместе	85
Глава 5. Промежуточный Word	89
5.1 Шаблоны документов	89
Что входит в шаблон?	89
Поиск шаблонов	90
Использование шаблонов	91

Создание шаблонов	91
Использование шаблонов в работе	92
5.2 Таблицы	92
Нестандартные таблицы	100
5.3 Создание документов с несколькими колонками.....	101
5.4 Сноски и концевые сноски.....	103
5.5 Добавление графики в документы	107
Два лица графики.....	108
Работа с InlineShapes.....	108
Работа с фигурами.....	110
5.6 Работа с существующими документами.	113
Поиск и замена	113
Изучение структуры документа	118
Обход документа с коллекциями	120
5.7 Автоматизация макросов	121
5.8 Собираем все вместе.....	122
Глава 6. Продвинутый Word.....	128
6.1 Организация документа с помощью стилей.....	128
Работа со структурой	131
Создание оглавления	133
6.2 Создание указателей.....	135
Решение о том, что индексировать.....	135
Маркировка элементов для индекса.....	137
Генерация индекса	137
Форматирование индексов	138
6.3 Объединение документов с данными.....	139
Структура слияния почты Word	139
Замена данных с помощью функции «Найти и заменить»	140
Отступите на 10 ярдов и нанесите удар.....	141
Документы, участвующие в слиянии почты	141
Объекты, участвующие в слиянии почты	141
Создание основного документа	142
Прикрепление источника данных	142
Выполнение слияния почты	147
Определение типа документа.....	147
Спасение брошенных документов слияния почты	148
6.4 Собираем все вместе.....	149

Глава 7. Основы Excel	157
7.1 Объектная модель	157
7.2 Дежавю	157
7.3 Получение информации о приложении	158
7.4 Управление рабочими книгами и рабочими листами.....	159
Открытие новой книги	159
Открытие существующей книги	160
Сохранение книги	161
Как избежать пользовательских диалогов Excel.....	162
Работа с рабочими листами	163
Строки, столбцы, ячейки и диапазоны.....	164
Условно говоря.....	165
Множество диапазонов.....	165
7.5 Обработка значений и формул	165
Добавление, копирование и перемещение данных (включая формулы)	166
Заполнение листа по ячейкам	166
Получение сгенерированного XLS.....	168
Перемещение листов.....	168
Наполни ее	169
7.6 Форматирование	171
Ширина и выравнивание столбцов	171
Выравнивание	172
Шрифты	173
Форматирование значений	174
Границы	178
Затенение.....	179
Стили	181
7.7 Вывод данных	182
Настройка страницы.....	182
Предварительный просмотр печати.....	185
Печать.....	187
Сохранение данных в разных форматах	187
7.8 Собираем все вместе.....	187
Глава 8. Расширенные функции Excel	193
8.1 Сводные таблицы.....	193
Данные сводной таблицы	194
Создание сводной таблицы.	196

8.2 Анализ «что если» с использованием таблиц данных	202
8.3 Поиск цели	206
8.4 Защита	207
Защита клетки.....	207
Защита листа	208
Защита книги.....	208
8.5 Шаблоны	209
Что входит в шаблон?	209
Поиск шаблонов.....	209
Использование шаблонов	209
Создание шаблонов	210
8.6 Автоматизация макросов	210
8.7 Добавление фигур	211
8.8 Добавление изображений	215
8.9 Собираем все вместе.....	216
Глава 9. Графический механизм Excel	221
9.1 Объектная модель	221
Это диаграмма или лист?.....	221
9.2 Настройка данных	222
Имена диапазонов – ваши друзья.....	224
9.3 Создание диаграммы.....	225
Свойство ActiveChart	226
Выходим к мастеру.....	226
Исходный диапазон	228
Типы и форматы диаграмм	229
PlotBy - выбор строк или столбцов.....	232
Ярлыки категорий и рядов.....	232
Есть ли у него легенда?	233
Параметры заголовка	233
9.4 Анатомия диаграммы.....	234
9.5 Форматирование компонентов	235
9.6 Объект Chart.....	236
Типы диаграмм	237
Свойства объекта Chart доступные только для трехмерных диаграмм.....	238
Область диаграммы	240
Объект Border	240
Объект Interior	241

Объект Font.....	242
Объект PlotArea	243
Ряд данных.....	244
Как хранятся данные и метки	244
Форматирование Series.....	245
Метки данных	246
Полосы ошибок	247
Форматирование маркеров данных (линейные, лепестковые и точечные диаграммы).....	249
Сглаживание линий на графиках.....	250
Форматирование круговых диаграмм	250
Форматирование отдельных точек.....	252
Объект легенды	253
Оси координат.....	254
Объект TickLabels	256
Линии сетки	257
Заголовки	257
9.7 Собираем все вместе.....	258
Глава 10. Основы PowerPoint.....	264
10.1 Объектная модель PowerPoint.....	264
10.3 Как разобраться с приложением	266
10.4 Управление презентациями и слайдами	268
Представление: объект Presentation	268
Открытие существующей презентации.....	269
Добавление слайдов	270
Сохранение презентации	272
Закрытие презентации	272
Закрытие объекта приложения	273
Сделать так, чтобы это выглядело хорошо для пользователей	275
10.5 Работа с содержимым слайдов.....	275
Использование макетов слайдов	276
Формы, формы и еще раз формы	278
Объект Shape	280
Добавление линий	281
Вопросы для размышления.....	282
Форматирование линий	282
Добавление автофигур.....	286

Форматирование границы формы	288
Заполнение формы.....	288
Добавление и форматирование текста	289
Добавление текста.....	289
Поиграйтесь с текстовыми сегментами	290
Форматирование абзацев	291
Маркеры.....	292
Нумерованные маркеры.....	292
Маркер-картинка	294
Веселье со шрифтами	294
10.6 Делаем это презентабельно	296
Запуск режима слайд-шоу	296
Печать.....	297
10.7 Собираем все вместе.....	298
Глава 11. Расширенные функции PowerPoint.....	304
11.1 Достижение согласованности с мастер-слайдами	304
Фон.....	304
Текстуры	305
Заливка изображения.....	306
Градиентная заливка	306
Стандартизация внешнего вида текста	310
Стандартизация цветов с помощью ColorSchemes	312
Подробнее о формах SlideMaster.....	313
11.2 Необычные функции	314
Анимации.....	314
Настройка эффекта	315
Изменение эффектов	315
Автоматическое определение времени	317
Переходы.....	317
Выполнение действий	318
11.3 Мультимедиа.	320
Отключение звука.....	320
Движение	322
11.4 Добавление примечаний	324
11.5 Собираем все вместе.....	326
Глава 12. Автоматизация Outlook.....	333
12.1 Объектная модель Outlook	333

12.2 Знакомство с Outlook	334
12.3 Доступ к содержимому Outlook	335
12.4 Отправка электронной почты	337
Получатели и контакты	339
Прикрепление файлов	341
12.5 Управление папкой Contacts	341
12.6 Добавление встреч и задач	345
12.7 Собираем все вместе	351
Глава 13. Межофисная связь	355
13.1 Взаимодействие между приложениями Office	355
Копирование и вставка между приложениями	355
Настройка исходных данных	355
Скопируйте данные в буфер обмена	357
Подготовка целевого приложения	357
Вставка данных из буфера обмена	358
PasteSpecial	359
Использование AddOLEObject	363
13.2 Информирование о мероприятиях с помощью VFPCOM	367
Получение VFPCOM.DLL	367
Использование VFPCOM.DLL	367
VFPCOM с Excel	371
VFPCOM с PowerPoint	371
13.3 Последнее слово	373
Глава 14. Обработка ошибок Автоматизации	374
14.1 Ошибки, связанные с Автоматизацией	374
14.2 Война 1426-1429	375
Тонкие различия	375
Некоторые интересные наблюдения за ошибками	377
14.3 Стратегии обработки ошибок	378
ON ERROR и метод Error	379
Предотвращение ошибок	382
Обеспечение существования сервера	383
14.4 Подведем итоги	389
Глава 15. Подведение итогов работы с серверами	390
15.1 Зачем обертывать серверы?	390
15.2 Структурирование классов-оболочек	391
15.3 Фундаментальные операции	391

Разговор с пользователем	392
Создание экземпляра сервера	393
Закрытие сервера	395
Открытие документа	395
Сохранение документа	396
Другие основные операции	398
15.4 Создание подклассов	398
Закрытие серверов	398
Открытие документа, часть 2	400
Закрытие документа	400
Создание новых документов	402
Проверка сервера	405
Проверка открытых документов	406
Отслеживание активного документа	406
Отображение сервера	407
15.5 Использование основных операций	408
15.6 Добавление операций, специфичных для сервера	409
Обёртка Word - от таблицы к таблице	410
Обёртка Excel - от таблицы к графику	414
Обёртка PowerPoint - стандартизация текста	418
15.7 Подводя итоги	423
Приложение А. Интернет-сообщества пользователей	425
А1. Ресурсы FoxPro	425
А2. Ресурсы Office	427
А3. Онлайн-этикет	427
Приложение В. Значения ChartWizard	429

Введение

Что это за книга? Зачем тебе это? Как Вы этим пользуетесь?

Эта книга выросла из другой книги. В 1995 году, когда FoxPro превратился в Visual FoxPro, Тамар вместе с Тедом Рошем, техническим редактором этой книги, написали книгу под названием «*Руководство хакера по Visual FoxPro 3.0*» (Аддисон-Уэсли). Большая часть этой книги, почти 700 из 900 с лишним страниц, представляла собой алфавитную ссылку на каждую команду, функцию, свойство, событие, метод и системную переменную в Visual FoxPro. Как ни странно, Тамар и Тед не просто сели и создали единый документ Word, начиная с «А» и заканчивая «Z». Фактически в этот раздел книги вошло около 600 документов.

Когда работа была закончена и дым рассеялся, Теду пришла в голову блестящая идея. (На самом деле, эта идея пришла ему в голову задолго до того, как все было написано.) VFP 3 была первой версией продукта, поддерживающей Автоматизацию. Он написал код, который взял эти 600 документов и соединил их в правильном порядке, чтобы создать справочный раздел «Руководства хакера».

В 1998 году Тамар и Тед снова взялись за дело, выпустив обновленную и расширенную версию под названием *Hacker's Guide to Visual FoxPro 6.0* (Hentzenwerke). На этот раз Тамар взяла на себя задачу собрать книгу, а Тед с помощью Автоматизации превратил все это в файл справки HTML. В новом материале VFP 5 и 6 было задействовано не 600 документов; их было более 800.

Кроме того, изменения в способах управления книгой означали, что перед выпуском окончательной версии нужно было проделать большую работу по очистке. Например, Microsoft изменила название продукта во время бета-тестирования. (Visual FoxPro 98 стал Visual FoxPro 6.) Редактор копий проделал замечательную работу по поиску несоответствий в терминологии, уловив такие понятия, как «textbox» и «text box», но некоторые разделы, отредактированные на ранних этапах процесса, необходимо было исправить. Самая большая проблема возникла очень поздно, когда было принято решение сделать книгу размером 8,5 x 11 дюймов вместо 7 x 9 дюймов. Это означало, что нужно было изменить поля, изменить размеры таблиц и многое другое. Изменения были необходимы в каждом из этих более чем 800 документов.

Внесение всех этих изменений вручную заняло бы месяцы, и книга была бы безнадежно запоздавшей. Автоматизация спешит на помощь. Тамар написала код VFP, позволяющий открывать каждый документ в Word, выполнять необходимую обработку и сохранять его как новый документ. После завершения всей обработки обновленная версия исходного ассемблерного кода Теда создала справочный раздел.

Попутно, влюбившись в Автоматизацию, Тамар регулярно билась головой о стену. Грубая сила была в порядке вещей. Документация Word оказалась полезной, как и люди на форуме MSWord компании CompuServe. Но нигде не было настоящего ресурса для написания такого кода Автоматизации, особенно для тех, кто писал его из Visual FoxPro.

Тем временем, вернувшись на ранчо

Пока Тамар и Тед путешествовали с книгами, Делла работала над, пожалуй, самым известным в мире приложением FoxPro. Joint Flow and Analysis System for Transport (JFAST) – это логистическое приложение, написанное для Министерства обороны США. Оно помогает в планировании движения людей и техники, дает длинный

список того, что вам нужно (от танков до солдат и столовых), а также где оно находится сейчас, где оно должно быть и когда оно должно туда добраться, и составляет подробный график погрузки людей и техники на корабли и самолеты для доставки их к месту назначения. (Это предложение слишком смело описывает то, что делает JFAST. Наблюдение за этим приложением в действии – потрясающий опыт.)

Часть инструмента Деллы создает отчеты и брифинги, объясняющие рекомендуемый план. Автоматизация используется для проведения брифингов в PowerPoint, подходящих для представления плана высокопоставленным генералам и даже главнокомандующему. Автоматизация также используется для создания документа Word, в котором анализируются сильные и слабые стороны плана. Другие модули были разработаны для Автоматизации Microsoft Project и Microsoft Schedule+, чтобы обеспечить еще больше аналитических представлений.

Все это произошло, когда Брайан Джонс, гений JFAST (и его руководитель проекта), пришел в офис Деллы с идеей. Он знал, что можно взять данные из таблиц FoxPro и поместить их в другую программу, и что Автоматизация (на самом деле в то время доминирующей технологией была DDE) была ключевой. Он описал, чего хочет, и Делла отправилась выполнять свою миссию.

Сначала появилась AutoBrief, автоматизированная система брифингов. Идея состоит в том, что пользователь может выбрать серию слайдов из основного списка и нажать кнопку «Старт». AutoBrief генерирует слайды на основе текущих данных в таблицах FoxPro. На момент зарождения AutoBrief текущей версией был Office 95. Документация по объектным моделям Office была скудной; например код, какой пример кода? Даже примеры VB было трудно найти. Но благодаря большому упорству Делла сумела овладеть объектом PowerPoint и моделью Excel. Excel используется для создания множества графиков, необходимых в презентации.

Затем появился автоанализ, который создает документ Word, содержащий списки, таблицы, диаграммы и даже словесный анализ. Опять же, используя ограниченную документацию, код Деллы создает очень профессиональный документ одним нажатием кнопки, включая оглавление, указатель, верхние и нижние колонтитулы, списки, а также графики Excel и карты данных.

По крайней мере, в Word и Excel была функция записи макросов, что значительно облегчало изучение объектной модели; В PowerPoint 95 вообще не было средства записи макросов. (См. главу 2 «Офисные серверы», чтобы узнать, насколько полезны средства записи макросов.)

Затем появился Office 97. Вся объектная модель Excel, Word и PowerPoint была радикально изменена. Изменение было к лучшему, поскольку объектная модель каждого приложения была *гораздо* более согласованной с другими приложениями Office (то есть имела лучший полиморфизм). Однако какими бы замечательными ни были изменения, код все равно оставался нарушенным. Долгое время. Именно тогда Делла осознала ценность написания кода-оболочки. К счастью, код-оболочку было писать немного проще, поскольку у Office 97 была гораздо лучшая документация как по количеству, так и по качеству. Обработка ошибок стала проблемой, поскольку все больше и больше пользователей в значительной степени полагались на эту функцию Автоматизации, иллюстрируя необходимость проверки поврежденных реестров, неправильной установки Office и других ошибок, которые разработчики FoxPro не привыкли проверять.

Делла ежедневно разрабатывает код Автоматизации в FoxPro с 1995 года. Она по-прежнему желает получить хороший ресурс по Автоматизации для разработчика FoxPro. Поэтому она ухватилась за шанс написать эту книгу.

Что это за книга?

В этой книге мы попытаемся избавить вас от боли, через которую мы уже прошли. Мы сделаем это несколькими способами. Во-первых, мы поделимся ключевыми моментами Автоматизации приложений Office, которые, по нашему мнению, должен знать каждый, кто с ними работает. На самом деле, мы считаем, что большая часть из этого актуальна независимо от того, автоматизируете ли вы Office с помощью Visual FoxPro, Visual Basic, Visual C++ или Visual SquidPro.

Во-вторых, мы расскажем вам все, что знаем о том, как узнать больше о серверах Office Automation. Мы поделимся своими приемами, позволяющими выяснить, с каким объектом вам нужно поговорить, какой метод вызвать и какое свойство действительно важно.

В-третьих, мы расскажем вам, что нас сбило с толку. Мы расскажем вам о методах, которые казались нам интуитивно очевидными, но на самом деле оказались неправильными.

Версии

Эта книга была написана с использованием Visual FoxPro 6.0 (на самом деле VFP 6.0 с пакетом обновления 3, но это применимо и к исходной версии) и Microsoft Office 2000. Почти все здесь применимо и к Office 97. При наличии серьезных различий в версиях в тексте вы найдете значок, предупреждающий вас.

Что касается VFP, все немного сложнее. Автоматизация работала в VFP 5 достаточно хорошо, и велика вероятность, что почти все здесь работает и там, но большую часть этого мы там не тестировали. Когда мы пишем эту книгу, мы надеемся вскоре увидеть ранние бета-версии следующей версии Visual FoxPro. Все в нашем прошлом опыте говорит нам, что все это будет работать и там, и на самом деле будет работать лучше, то есть любые проблемы, которые могут возникнуть у нас в VFP 6, скорее всего, будут исправлены в VFP 7 (или как они решили это назвать).

Использование примеров

Поскольку мы программисты FoxPro и поскольку во всех других книгах, представленных на рынке, в качестве примеров используются Visual Basic, примеры в этой книге написаны на VFP. Но что удивительно, так это то, что вам нужно внимательно изучить примеры кода, чтобы понять это. В этом дивном новом мире Автоматизации и совместимости код VFP и код VB не выглядят такими разными, как раньше. Но об этом позже.

В основных примерах в этой книге используется база данных TasTrade, поставляемая с Visual FoxPro. Если вы работали с другим продуктом Microsoft (скажем, Access), фактические данные могут показаться вам знакомыми. Это потому, что данные TasTrade VFP практически такие же, как база данных Northwinds Access. Есть некоторые различия, но клиенты, сотрудники и данные внутри выглядят довольно похоже.

В VFP 6 и более поздних версиях системная переменная `_SAMPLES` указывает на каталог, в который были установлены примеры программ и данных. В примерах в этой книге эта переменная используется для поиска данных TasTrade. Чтобы установить пример данных, необходимо выполнить часть MSDN установки VFP, при которой также устанавливается файл справки.

Если вы используете более раннюю версию VFP, замените ссылки на `_SAMPLES` на `HOME()+"\SAMPLES\"`. В этих версиях образцы устанавливались как часть основной установки VFP. Никакой установки MSDN не требовалось.

В большинстве встроенных примеров в книге предполагается, что вы уже создали соответствующий сервер Автоматизации и сохранили ссылку на него в переменной. В первой главе, посвященной каждому из серверов, мы покажем вам, как это сделать, и представим переменную, которую мы используем для этого сервера (например, oWord для Word). После этого мы предполагаем существование переменной и наличие у нее допустимой ссылки. Если во время чтения вы просматриваете примеры из командного окна, это предположение должно показаться вам вполне комфортным.

Для более крупных примеров нам пришлось сделать выбор. В реальных приложениях почти всегда требуется открыть сервер Автоматизации, сделать то, что необходимо, и закрыть сервер. В этой ситуации вы можете использовать локальную переменную для хранения ссылки на сервер. Однако для наших целей мы обычно хотим, чтобы сервер оставался открытым и доступным, как показано в примере, чтобы вы могли просматривать результаты и ссылаться на сервер из командного окна. Однако мы не хотели оставлять несколько экземпляров серверов работающими, заброшенными и использующими ресурсы вашей системы. Поэтому в большинстве примеров, включенных в файлы для загрузки для разработчиков, удаляются все переменные, которые могут быть ссылками на объекты Автоматизации, а затем создаются общедоступные переменные для ссылки на сервер. Мы не рекомендуем использовать этот метод в ваших приложениях.

Это посвящается тем, кого мы любим.

Благодарности

Как и в случае с любой другой работой такого рода, мы не смогли бы написать эту книгу без помощи многих людей.

Мы начнем с общей благодарности людям, которые так или иначе внесли свой вклад в содержание книги, а затем закончим нашей более личной благодарностью.

Наш технический редактор Тед Рош протестировал массу примеров, задавал трудные вопросы, ловил нас, когда мы вели себя небрежно, и вообще делал то, что положено делать техническим редакторам. Но на самом деле он сделал гораздо больше, потому что именно он первым показал Тамар, насколько элегантно может быть Автоматизация, и подсадил ее на нее. Без Теда эта книга никогда бы не появилась.

Хотя Хентцен был моим другом гораздо дольше, чем издательским магнатом. Спасибо, что еще раз предоставили возможность людям, которым просто нужно написать слова на страницах.

Многие люди отвечали на наши вопросы, пока мы пытались понять, как работает Office. Многие из них получили награды Microsoft MVP за помощь, которую они оказывают другим. Мы благодарим Криса Вудмана, Боба Бакленда, Брайана Рейли, Рольфа Келлера, Джесси Луизу МакКленнан, Майка Шерилла и Стивена Стерна. Особая благодарность Синди Мейстер, которая сделала обзор некоторых материалов Word в более ранней версии, а также ответила на массу вопросов. Наши извинения всем, кого мы пропустили.

Несколько человек дали больше, чем просто ответы. Рик Страл позволил нам включить GetConstants.EXE, приложение, извлекающее константы из библиотеки типов, в файлы для загрузки для разработчиков, доступные по адресу www.hentzenwerke.com. Дополнительную информацию см. в главе 2 «Офисные серверы».

Джон В. Петерсен познакомил нас с VFPCOM.DLL, которая обеспечивает двустороннюю связь между VFP и серверами Автоматизации (см. главу 13

«Межофисная связь»), и написал специальную DLL, позволяющую работать с PowerPoint. Библиотека Джона включена в файлы загрузки для разработчиков. Роберт Грин из Microsoft также предоставил нам некоторую информацию, необходимую для включения VFPCOM.DLL.

Иногда самая большая помощь приходит не от ответов на наши вопросы, а от предоставления идей и возможностей. В 1995 году Брайан Джонс обратился к Делле со своими идеями относительно компонента офисной Автоматизации знаменитой программы JFAST. С тех пор Делла пишет код Автоматизации. Благодаря идеям Брайана и способности реализовать их в столь долгосрочном проекте, Делла обладает знаниями и опытом, которые можно задокументировать в этой книге для других.

За прошедшие годы нам помогло слишком много людей в сообществе FoxPro, чтобы мы могли перечислить их всех здесь. Вы знаете, кто вы, и мы ценим всю вашу помощь. Однако особая благодарность Маку Рубелю, который помог сформулировать некоторые из этих идей на самом раннем этапе процесса, и Дэну Фримену, который очень многому научил Тамар в работе с объектными моделями Office.

Многие люди из различных продуктовых групп Microsoft участвовали в том, чтобы довести Автоматизацию до такой степени, что она стала жизнеспособной технологией. Мы не знаем их имен, но определенно ценим их работу.

К тому времени, как вы пишете третью книгу, ваша семья уже привыкает к ней и начинает задаваться вопросом, не является ли это какой-то зависимостью. (Это так.) Еще раз спасибо моему мужу Маршаллу, который делает все это возможным, и моим сыновьям Соломону и Натаниэлю, которые делают это стоящим.

Моя большая семья и друзья всегда ценят вашу готовность позволить мне болтать о моей работе, а также изменение темпа, которое вы обеспечиваете.

Как всегда, спасибо команде Advisor Media, а также другим моим друзьям и коллегам из профессионального сообщества.

Тамар

Поскольку это моя первая книга, моя семья только привыкает к ней и, благодаря влиянию Тамар, они предполагают, что это станет образом жизни. (Я надеюсь на это.) Мой муж Майк был потрясающим источником ободрения, вдохновения и поддержки. Моя дочь Келси и мой сын Керри были удивительно терпеливы, слыша, как я много раз говорила: «Мы сделаем это позже, дорогая. Мама сейчас пишет свою книгу».

Спасибо моим маме и папе за всю их любовь и огромную поддержку. Моя большая семья заслуживает огромной похвалы за то, что терпела мой ноутбук на многих семейных встречах.

Есть так много других из вас, которых стоит поблагодарить. Большое спасибо CompuServe FoxGang за всю уверенность и знания, которые вы мне помогли обрести. Огромное спасибо моим коллегам из Института системного развития Университета Теннесси и особенно всей команде JFAST. И всем друзьям и знакомым, которых я встретила в Интернете и на DevCons, спасибо за вашу поддержку.

Тамар, Тед и Уил: особая благодарность вам за приглашение сотрудничать с вами и осуществить мою мечту о написании книги.

Дела

Глава 1. Знакомство с Автоматизацией

Автоматизация является последней и наиболее успешной в серии попыток взаимодействия между приложениями. Это позволяет одному приложению управлять другим.

Когда-то каждое приложение работало отдельно. Если ему нужно было что-то сделать, оно должно было сделать это само.

Альтернативы не было, поскольку каждое приложение брало на себя управление компьютером при запуске. Это было более или менее «давным-давно в далекой-далекой галактике».

Со времен DOS было предпринято множество попыток позволить приложениям взаимодействовать друг с другом. Каждая последующая попытка была улучшением предыдущей. (Ну, в любом случае, большинство из них были улучшениями.) Автоматизация – это новейшая и лучшая разработка в бесконечном поиске приложений, которые хорошо выполняют одну задачу и взаимодействуют с другими приложениями, которые хорошо делают что-то другое.

1.1 Если все, что у вас есть, — это молоток, значит, все становится гвоздем?

Когда вы действительно знакомы с продуктом, возникает соблазн использовать его для всех своих нужд. Многие люди годами делают это со своими любимыми приложениями. Мы знаем людей, которые обрабатывают тексты в Quattro Pro, и других, которые выполняют «обработку баз данных» с помощью слияния почты в Word.

Конечно, это справедливо и для программистов FoxPro. Нужен калькулятор чисел? Напишите это в FoxPro. Нужен текстовый процессор? Напишите это в FoxPro. Нужен файловый менеджер? Напишите это в FoxPro. То, что мы можем делать так много разнообразных вещей, является свидетельством силы и универсальности FoxPro. Но использование для этого молотка (FoxPro) не забивает все эти гвозди.

Растут как потребности пользователей, так и возможности приложений. Хотя в 1994 году имело смысл писать простой текстовый процессор для редактирования заметок клиентов в FoxPro/DOS, в Visual FoxPro 6.0 в 2000 году делать это не имеет смысла. Существует слишком много альтернатив, которые можно просто подключить к приложению VFP.

1.2 Краткая история Автоматизации

Вначале был DDE, *динамический обмен данными*. DDE был первой попыткой Windows разрешить приложениям напрямую взаимодействовать друг с другом. (До этого, конечно, приложения могли взаимодействовать, обмениваясь данными через общие форматы файлов и буфер обмена.) С помощью DDE документ можно было связать из клиентского приложения с документом в серверном приложении. Сервер предоставил одну или несколько «тем», которые он мог сделать доступными; затем клиент мог попросить «обсудить» любую из этих тем. Ряд приложений по-прежнему поддерживают DDE.

OLE, преемник DDE, включил в себя несколько подходов к взаимодействию между приложениями. Однако служба, которая дает OLE акроним, – это возможность помещать документ из одного приложения (сервера) в документ из другого (клиента). Вы можете *связать* документ сервера с документом клиента, чтобы документ клиента обновлялся при изменении документа сервера. Альтернативно вы

можете *встроить* серверный документ в клиентский документ, который создаст копию серверного документа, разрывая его связь с оригиналом. Эти два варианта привели к появлению названия «*Связывание и внедрение объектов*». (Майкрософт фактически дезавуировал эту версию имени. Что касается OLE, то это просто OLE, но нам легче запомнить его значение.)

Со временем OLE был расширен для поддержки *редактирования на месте*, что позволяет просматривать и редактировать объекты из одного приложения в другом. Когда пользователь решает отредактировать связанный объект, меню клиентского приложения заменяется или дополняется элементами из меню серверного приложения. Пользователь может редактировать серверный документ по мере необходимости, а затем вернуться к клиентскому приложению, после чего возвращаются исходные меню и панели инструментов.

Однако вишенкой на торте OLE является Автоматизация, которая появилась в OLE 2.0. С помощью Автоматизации команды можно выдавать в одном приложении и отправлять в другое. Они переписываются в стандартном объектном коде с использованием соответствующего синтаксиса для основного языка (того, который их выдает, в нашем случае это FoxPro). Думайте об Автоматизации как о том, что одно приложение хватается мегафон и говорит другому, что делать. Количество приложений, которые работают с Автоматизацией, либо как приложение, владеющее мегафоном, либо как приложение, слушающее на другом конце, все время увеличивается.

1.3 OLE, ActiveX, COM или что?

Поскольку все это исходит от Microsoft, само собой разумеется, что названия вещей со временем изменились. Когда Автоматизация была впервые представлена, она называлась «OLE-Автоматизация». Затем «OLE» было изменено на «ActiveX», и OLE Automation на короткое время стала называться ActiveX Automation. В конечном итоге Microsoft исключила слово «ActiveX» из названия, и эта технология стала называться просто «Automation» («Автоматизация»).

Технология, которая объединяет все это, – это COM (компонентная объектная модель или общая объектная модель, в зависимости от фазы луны), которая определяет способы совместной работы приложений. Зонт COM включает в себя методы OLE/ActiveX и даже обеспечивает привязку к объектам операционной системы, многие из которых также работают с Автоматизацией.

Последней и самой популярной версией всего этого является DCOM (Distributed COM), которая позволяет COM работать на нескольких машинах. С помощью DCOM Автоматизация может включать приложения, работающие на двух разных машинах.

1.4 Использование Автоматизации на практике.

В любом сеансе Автоматизации участвуют два приложения. Приложение, отвечающее (выдающее заказы), называется *клиентом Автоматизации* (или просто *клиентом*). Приложение, которым манипулируют, – это *сервер Автоматизации* (или просто *сервер*). Клиент обращается к экземпляру сервера, как если бы это был любой другой объект, считывая и устанавливая свойства и вызывая методы. Этот простой метод означает, что приложение Visual FoxPro (клиент) может обращаться к чему угодно: от приложений Office до файловой системы Windows, Lotus Notes и многих других вещей. Фактически, сам VFP можно использовать в качестве сервера Автоматизации.

Для начала клиент создает экземпляр серверного приложения. (Подробную информацию о том, как это сделать для каждого из приложений Office, см. в

различных главах «Основы» в разделах этой книги, посвященных конкретным продуктам.) Внутренне, когда клиент пытается создать экземпляр сервера, Windows обращается к реестру и говорит: «Помогите! Кто-то хочет создать такой-то сервер». Реестр ищет сервер по имени, а затем обнаруживает что это за программа и где она хранится. Windows выполняет рассматриваемую программу (при условии, что она находит такую запись в реестре и находит указанную программу там, где она должна быть). Затем она возвращает ссылку на вновь выполняющуюся программу клиенту, который удерживает ее, чтобы иметь возможность найти ее позже.

Очевидно, что на этом пути есть много мест, где что-то может пойти не так. Имя сервера может отсутствовать в реестре, программа может быть не найдена там, где должна быть, может не хватить системных ресурсов для запуска программы и так далее и тому подобное. Если что-то пойдет не так, выдается ошибка и сервер не запускается. (См. главу 14 «Обработка ошибок Автоматизации», где представлены идеи о том, что делать в таких случаях.)

Однако если все пойдет хорошо, клиент получит ссылку на сервер и сможет начать управлять сервером. Это похоже на то, как будто вы находитесь в ресторане, когда официант представился. Вы знаете его имя и знаете, как он выглядит, поэтому вы можете позвонить ему, когда он вам понадобится, и начать говорить ему, что делать. За исключением одного. Вы не знаете, что в меню. Вы можете сделать некоторые обоснованные предположения, основываясь на том, как выглядит это место. Для приложения это соответствует догадкам, основанным на том, что это за приложение и что, как вы знаете, оно делает. Но чтобы эффективно использовать ресторан, нужно меню. Чтобы эффективно работать с сервером приложений, необходимо знать, на что он способен. В главе 2 «Офисные серверы» рассматривается, как найти меню сервера приложений.

1.5 Куда идти дальше?

Остальные главы раздела I посвящены серверам Office в целом, а также командам и функциям VFP для написания кода Автоматизации. В главе 2 показано, как исследовать серверы Office, чтобы узнать, что они могут делать и как они это делают. В главе 3 «Visual FoxPro как клиент Автоматизации» рассматривается роль FoxPro как клиента Автоматизации.

Каждый из разделов II-V посвящен одному из приложений Office. Word, Excel и PowerPoint имеют много общего, поэтому эти три раздела структурированы довольно схожим образом: каждый начинается с главы «Основы», а затем переходит в более сложные темы для этого приложения. Outlook имеет свой взгляд на мир. (Мы не обсуждаем Access в этой книге, поскольку Автоматизация Access из Visual FoxPro не кажется острой необходимостью, хотя это возможно.)

Также важно понимать, что для Автоматизации любого приложения вы должны сначала ознакомиться с этим приложением. В этой книге объясняется, как работают различные функции приложений Office, прежде чем показывать, как их автоматизировать, но в ней также предполагается, что вы уже использовали их раньше. Если вы никогда не открывали PowerPoint и не работали с ним в интерактивном режиме, раздел IV, например, покажется вам довольно сложным. Потратьте некоторое время в качестве конечного пользователя, прежде чем пытаться его запрограммировать. Мы будем возвращаться к этой теме снова и снова, потому что она важна.

Раздел VI охватывает ряд сложных тем. Сначала мы рассмотрим создание документов с использованием нескольких серверов Office, включая такие задачи, как размещение диаграмм из Excel в презентации PowerPoint или документы Word. В главе 14 рассматривается обработка ошибок Автоматизации. Ваш существующий

обработчик ошибок, вероятно, не сможет справиться с ними за вас, потому что внутри FoxPro они не происходят. Наконец, в главе 15 «Подведение итогов работы с серверами» мы покажем, почему, среди прочего, стоит писать классы-оболочки для серверов Автоматизации. Это защитит вас, когда новая версия Office изменит интерфейс Автоматизации.

Глава 2. Офисные серверы

Изучение серверов Office – непростая задача, но существует множество доступных подходов, инструментов и ресурсов. Некоторые функции приложений Office, такие как коллекции, являются общими для всех, их предварительное освоение быстро окупится.

Microsoft Office включает в себя целый ряд приложений и апплетов, но основной причиной, по которой люди покупают и используют его, являются два больших: Word и Excel. Наряду с ними есть еще несколько других, которые используются все большим числом людей, которые, возможно, изначально не планировали это делать в PowerPoint и Outlook.

Word – невероятно мощный текстовый процессор. Он предоставляет пользователям возможность создавать что угодно: от простой заметки до сложных отчетов и юридических документов, а также все, что между ними.

Excel – сложная программа для работы с электронными таблицами. Она включает в себя интегрированные возможности построения графиков и значительный набор встроенных финансовых, статистических и математических функций.

PowerPoint создает и отображает презентации. Как и другие приложения, её возможности варьируются от простых до сложных: она предлагает все: от простых слайдов с несколькими пунктами и текстом до полноценных мультимедийных шоу.

Outlook сложнее описать, потому что в нем есть всего понемногу. Для некоторых это просто почтовый клиент. Для других это календарь и список дел. Он также включает в себя адресную книгу, журнал, блокнот и множество возможностей для настройки.

2.1 Изучение серверов Office

Все приложения Office используют язык программирования Visual Basic для приложений или VBA. Для разработчиков Автоматизации наличие общего языка программирования не так впечатляет, как кажется, потому что, хотя каждое приложение Office использует один и тот же синтаксис и команды, это полезно только в том случае, если вы действительно пишете на этом языке программирования. Что касается Автоматизации, то нет. Вы пишете на языке клиентского приложения, управляющего процессом Автоматизации.

С другой стороны, по ряду причин очень удобно читать VBA при написании кода Автоматизации. Поскольку мы здесь все программисты, это не большая проблема. Но есть несколько особенностей VBA (во всяком случае, с точки зрения разработчика FoxPro), которые затрудняют чтение для тех, кто с ним не знаком. Мы рассмотрим каждую из этих проблем по мере изучения серверов Office.

Таким образом, при использовании серверов Office ключевой проблемой является выяснение того, какие методы и свойства они имеют, и определение параметров для передачи их методам. Существует три основных подхода. В большинстве случаев вам придется объединить все три, чтобы получить то, что вам нужно. Помимо этого, есть еще несколько способов узнать об Автоматизации, а затем проверить полученные знания.

Прочтите подробное руководство (RTFM)

На каждом из серверов Office имеется файл справки, в котором документированы его участники. Способ доступа к этому файлу справки зависит от приложения. В большой тройке Word, Excel и PowerPoint вы можете получить к ним доступ из главного меню справки приложения. На странице «Содержание» опуститесь почти до самого низа. В Word и Excel 2000 найдите «Информацию о программировании»

чуть выше нижней части списка содержания. В PowerPoint 2000 нужно найти запись «Справочник по Microsoft PowerPoint Visual Basic», также расположенную в нижней части списка «Содержимое». Во всех трех случаях вы можете открыть указанный элемент, чтобы увидеть список тем Автоматизации. Справка по Автоматизации Outlook хорошо скрыта. Вам нужно открыть раздел «Расширенная настройка», чтобы найти «Справочник по Microsoft Outlook Visual Basic».



Доступ к файлам справки в Office 97 немного отличается. В Word, Excel и PowerPoint вы начинаете одинаково: выбираете страницу «Содержание» в справке и прокручиваете страницу вниз. Затем во всех трех найдите «Справочник по Microsoft X Visual Basic», где X – это продукт, который вы используете. Когда вы выбираете этот элемент, он открывается и показывает «Справочник по Visual Basic». При выборе этого элемента открывается новое окно справки, содержащее только справку по Автоматизации для этого продукта. Доступ к справке Visual Basic для Outlook 97 чрезвычайно сложен. Эти шаги описаны в главном файле справки по поиску «Visual Basic». Однако следует понимать, что VBA не является основным языком программирования для Outlook 97, и автоматизировать его с помощью VBA в некотором смысле сложнее, чем автоматизировать другие продукты Office 97.

Если вы не можете найти соответствующий элемент в содержимом справки, это означает, что вы не установили файл справки VBA вместе с приложением. Для этого вам нужно выбрать Выборочную установку. Стандартная установка не включает эти файлы. (В Office 97 стандартная установка называлась «полной», но это не так.) Файлы справки VBA можно установить в любое время, повторно запустив программу установки. В Office 2000 это можно сделать, выбрав «Справка Visual Basic» в разделе «Инструменты Office». (По умолчанию файлы справки VBA настроены на установку при первом использовании. Если вы знаете, что планируете выполнять работу по Автоматизации, мы предлагаем вам сначала установить их на свой жесткий диск. Весь набор из четырех [Word, Excel, PowerPoint и Outlook] для Office 2000 составляет менее 5 МБ.)

К этому моменту вы, возможно, заметили, что доступ к разделу справки, посвященному VBA, может оказаться довольно трудным.

В Office 2000 справка VBA интегрирована в обычную справку и не открывает отдельное окно, поэтому работать в ней тоже затруднительно. В Office 97, когда вы попадаете в справку VBA, ее слишком легко закрыть, просто нажимите ESC, и она исчезнет.

К счастью, здесь есть простая альтернатива. Хотя справка VBA для каждого приложения Office доступна из основного файла справки соответствующего приложения, на самом деле она находится в отдельном файле. В Office 97 имя файла имеет вид VBAapp8.HLP, где «app» – это некоторая форма имени приложения. Для Office 2000 это VBAapp9.CHM, поскольку приложения Office 2000 используют справку HTML; именно поэтому вы больше не можете закрыть справку с помощью ESC. Учитывая все эти трудности с доступом к справке VBA, мы настоятельно рекомендуем вам создать на рабочем столе ярлыки для каждого файла справки, который вы, вероятно, будете использовать. Тогда они находятся на расстоянии всего лишь двойного щелчка мыши.



По большей части переход с Office 97 на Office 2000 носит либо положительный, либо нейтральный характер. Нам также нравится HTML-справка в

целом. Фактически, электронная версия этой книги использует HTML-справку. Однако файлы справки Office 2000, включая файлы справки VBA, гораздо менее полезны, чем их эквиваленты Office 97. В Office 97 файлы справки имели три страницы: «Содержание», «Указатель» и «Поиск». Index представлял собой алфавитный список терминов из файла справки (например, указатель книги), а «Поиск» обеспечивал полнотекстовый поиск. Для файлов справки VBA страница индекса предлагала быстрый способ перехода непосредственно к любому объекту, свойству или методу.

Файлы справки Office 2000 имеют только две страницы: «Содержание» и «Поиск». Команда Microsoft объединила страницы «Указатель» и «Поиск», чтобы создать единую страницу поиска, которая предлагает полнотекстовый поиск без компонента страницы «Поиск», работающего в режиме реального времени. Невозможно быстро перейти к ключевому слову. Мы подозреваем, что в файлах справки для основных продуктов есть несколько хороших аргументов в пользу нового механизма поиска. Конечные пользователи, вероятно, сочтут такой подход более продуктивным. Однако мы не можем понять, как программисты могут продуктивно использовать файлы справки VBA без индекса. (Лучшей заменой индексной страницы являются отдельные алфавитные списки объектов, свойств и методов на странице содержания. Хотя мы сохранили файлы справки VBA на странице указателя в Office 97, в Office 2000 мы обычно оставляем их на странице «Содержание».)

Каждый файл справки VBA включает диаграмму объектной модели для этого продукта. Модель «живая» – при нажатии на нее происходит переход либо на другой уровень диаграммы, либо к соответствующей записи в Справке. На рисунке 1 показан верхний уровень диаграммы объектной модели Word.

Записи справки для объектов включают небольшие фрагменты диаграммы объектной модели, и нажатие на них также приводит к переходу к указанным записям. На рисунке 2 показана запись коллекции Documents в файле справки Word VBA. Щелчок по пунктирному окну с надписью «Несколько объектов» открывает диалоговое окно объектов, содержащихся в объекте Document.

Несмотря на трудности с переходом непосредственно к конкретному элементу справки Office 2000 (задача, которая была невероятно простой в Office 97), файлы справки VBA в целом ясны и правильны. Если вы знаете, что ищете, вы можете найти это там. Таким образом, настоящая задача состоит в том, чтобы выяснить, что вы ищете.

Пусть сервер напишет код

В Word, Excel и PowerPoint есть средства записи макросов, которые позволяют превращать действия пользователя в код VBA. (Outlook поддерживает макросы, но, к сожалению, не включает в себя средство записи макросов.)

Поэтому один из способов понять, как что-то автоматизировать, – это записать выполняющий это макрос, а затем изучить его.

В меню выберите «Инструменты|Макрос|Записать новый макрос», чтобы начать запись. Дайте макросу имя (желательно осмысленное, чтобы вы могли найти его снова и знать, что это такое, когда встретите его через шесть месяцев). Затем в интерактивном режиме выполните операцию, которую хотите автоматизировать. Когда вы закончите, нажмите кнопку «Остановить запись» на панели инструментов «Макрос» (которая появляется автоматически при начале записи).

Microsoft Word Objects

See Also

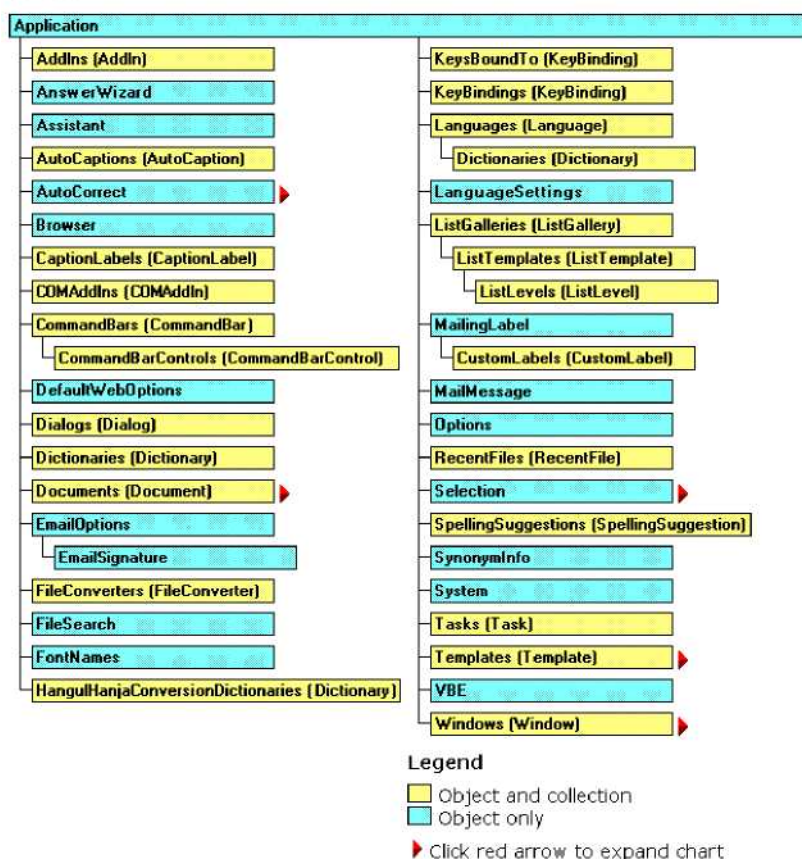


Рисунок 1. Диаграмма объектной модели Word. Файл справки для каждого продукта Office содержит диаграмму объектной модели для этого продукта. Нажав на один из прямоугольников, вы перейдете к записи справки для этого объекта. Щелкнув стрелку, вы перейдете на другой уровень диаграммы объектной модели.

Чтобы просмотреть записанный макрос, выберите в меню Инструменты|Макрос|Макросы; откроется диалоговое окно «Макросы», показанное на рисунке 3. Выделите только что созданный макрос (видите, вам уже нужно знать имя), а затем выберите «Редактировать». Вы попадете в редактор Visual Basic (VBE). На рисунке 4 показан выделенный макрос с рисунка 3 в том виде, в котором он отображается в VBE. Конкретно этот макрос был записан в Word. (Вы также можете перейти в VBE, выбрав «Инструменты|Макрос|Редактор Visual Basic» или нажав Alt-F11.)

расшифровать с помощью нескольких хитростей.

Преобразование макроса VBA в код VFP сложнее, чем должно быть, по нескольким причинам. Макрос, показанный на **рисунке 4**, демонстрирует все из них.

Рассмотрите эту строку кода Word VBA, читая следующие разделы. Линия перемещает точку вставки (вертикальную полосу, определяющую место вставки следующего символа) на шесть слов влево, выделяя слова в процессе:

```
Selection.MoveLeft Unit:=wdWord, Count:=6, Extend:=wdExtend
```

Объекты по умолчанию

Во-первых, в отличие от VFP, VBA делает некоторые предположения о том, с каким объектом вы разговариваете. В предыдущей строке Selection преобразуется в This.Selection VFP, который представляет текущий выбор (выделенную область) экземпляра Word.

В каждом приложении Office определенные объекты считаются объектами по умолчанию в среде VBA. К вашему коду не будут относиться так благосклонно – вам нужно четко указать, к какому объекту вы обращаетесь. Код, который вы отправляете на сервер Автоматизации, должен быть адресован правильному объекту.

Именованные параметры

VBA позволяет методам использовать так называемые *именованные параметры*. В строке кода примера вызывается метод MoveLeft. Передаются три параметра, каждый в следующей форме:

имя параметра := значение параметра

Этот синтаксис позволяет программистам VBA включать только те параметры, которые им нужны, и не беспокоиться о порядке параметров. Поскольку некоторые методы VBA имеют дюжину и более параметров, это очень удобный вариант.

Однако VFP не поддерживает именованные параметры; вы должны указать параметры в правильном порядке. К счастью, файлы справки показывают параметры в необходимом порядке. (Это было не так в версиях Office до Office 97; в справке по многим методам параметры отображались не в порядке, и найти правильный порядок было чрезвычайно сложно.)

При переводе из VBA в VFP добавьте круглые скобки вокруг списка параметров и удалите имена параметров и символы «:=». Обратитесь к справке (или к обозревателю объектов, который обсуждается далее в этой главе), чтобы определить правильный порядок и количество параметров. Обычно средство записи макросов размещает параметры в правильном порядке; однако некоторые параметры можно опустить, поскольку именованные параметры позволяют разработчикам VBA опускать любые параметры, которые должны принимать значения по умолчанию. Обязательно проверьте справку на предмет пропущенных значений. Также проверьте, действительно ли средство записи макросов разместило их в правильном порядке; иногда нет.

Определенные константы

Первый параметр в строке примера показывает третью проблему, возникающую при трансляции VBA в VFP. Он указывает, что параметр Unit должен иметь значение wdWord. Но что такое wdWord? Это одна из тысяч определенных констант, доступных в версии VBA для Word. (Оказывается, что wdWord равен 2.)

Файлы справки VBA не предоставляют значения определенных констант.

Фактически, Help использует исключительно их и нигде не показывает их действительные значения (то же самое касается и средства записи макросов). (Outlook здесь является исключением. В нем есть раздел справки под названием «Константы Microsoft Outlook», который включает полный список.) Чтобы узнать, что означают wdWord и все остальные, используйте обозреватель объектов, доступный через редактор Visual Basic. (См. раздел «Рассмотрим Обозреватель объектов» далее в этой главе.)



Если когда-либо была причина использовать файлы заголовков, то это константы VBA. Однако для построения заголовочного файла вам необходимо найти значения констант. Рик Страл из West Wind Technologies создал бесплатный инструмент, который считывает библиотеку типов COM, извлекает константы и создает заголовочный файл Visual FoxPro. Инструмент под названием GetConstants включен в файлы загрузки для разработчиков, доступные по адресу www.henzenwerke.com а также его можно скачать с сайта Рика (www.west-wind.com). Файлы загрузки для разработчиков также включают файлы заголовков для каждого приложения Office.

Мы *не* рекомендуем использовать эти файлы в вашей работе по Автоматизации. Количество содержащихся в них констант ошеломляет. Самый маленький набор – для Outlook, он содержит 251 определение констант. Размер файла Word почти в 10 раз больше, и в нем определено более 2400 констант. На одном из наших тестовых компьютеров сохранение формы без элементов управления, но с указанием полного файла определения констант Word в качестве включаемого файла заняло более пяти секунд. (Напротив, на той же машине сохранение совершенно пустой формы заняло меньше секунды.)

Однако иметь в своем распоряжении полный набор констант очень удобно. Вы можете вырезать и вставлять их для создания заголовочных файлов, соответствующих выполняемым вами задачам. Инструмент Рика также позволяет легко поддерживать актуальность файлов заголовков по мере того, как Microsoft добавляет новые константы.

Советы по записи макросов

Прежде чем перейти к обозревателю объектов, стоит отметить несколько вещей, касающихся средства записи макросов и редактора Visual Basic. Во-первых, имейте в виду, что средство записи макросов не всегда создает *лучший* код для задачи. Код, который он создает, дает вам представление о том, что можно сделать, но часто существуют лучшие способы выполнить ту же задачу.

Средство записи макросов ориентировано на интерактивных пользователей. Хотя оно не просто записывает нажатия клавиш пользователя, ему не хватает интеллекта, чтобы разобраться в поставленной задаче и составить полную, логичную программу для ее выполнения. Генерируются строки и строки кода. В большинстве случаев несколько строк кода можно заменить одним вызовом метода. В других случаях создается множество строк кода, которые задают все возможные свойства объекта. Возможно, установлено 15 свойств, когда вы изменили только одно. Во многих случаях вам не потребуется устанавливать их все. Однако когда вы пытаетесь выяснить, какие свойства необходимо установить, обязательно учтите, что многие пользователи меняют настройки приложения по умолчанию, поэтому не думайте ни о чем!

Во-вторых, в редакторе Visual Basic есть функция IntelliSense, которая упрощает написание кода. Когда вы вводите имя объекта и точку (например, «oRange»),

появляется соответствующий список свойств и методов. Когда вы выбираете метод из списка (используйте Tab, чтобы выбрать, не заканчивая строку), список параметров для этого метода появляется в виде всплывающей подсказки, которая поможет вам. Когда вы вводите параметры, ваша позиция в списке параметров выделяется, чтобы вы не сбились с пути. Это может быть очень удобно, когда вы пытаетесь выяснить, какие параметры передать методу. К сожалению, на момент написания этой статьи Visual FoxPro изначально не поддерживает IntelliSense (хотя ранние демо-версии VFP 7 включают его). Напишите код в VBE, вырежьте и вставьте его в VFP.

Рассмотрим обозреватель объектов

Одним из самых мощных инструментов, доступных для анализа кода Автоматизации, является обозреватель объектов (см. рисунок 5). Он позволяет детализировать различные объекты в иерархии, чтобы определить их свойства и методы, просмотреть параметры методов, определить значения констант и многое другое.

Самый простой способ узнать о конкретном предмете – ввести его в раскрывающийся список поиска, затем нажать Enter или кнопку «Найти» (бинокль). Средняя панель заполняется потенциальными совпадениями. Выберите один, чтобы узнать больше о нем в главном разделе браузера ниже. Левая панель заполнена свойствами, методами, коллекциями и константами. На правой панели описано, что доступно для выделенного элемента на левой панели. На рисунке 5 Обозреватель объектов использовался для определения значения константы wdWord. В самой нижней панели вы можете видеть, что это константа со значением 2.

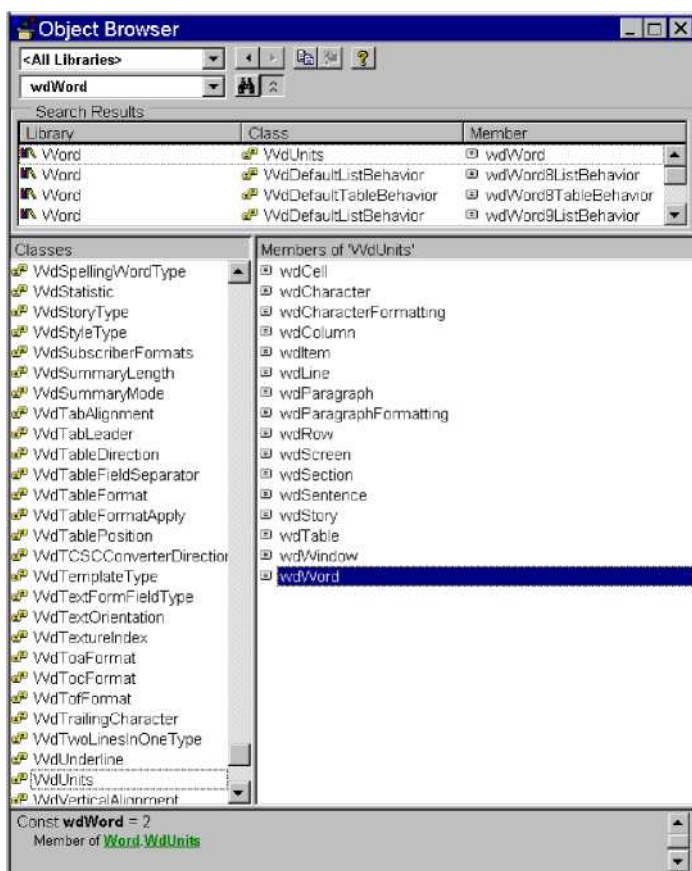


Рисунок 5. Обозреватель объектов. Этот мощный инструмент позволяет детализировать объекты, находить постоянные значения и определять параметры. Здесь показано, что wdWord является константой, членом группы констант под

названием WdUnits и имеет значение 2.

Обозреватель объектов также полезен для перемещения по иерархии объектов и понимания того, что могут делать различные объекты. На рисунке 6 показан обозреватель объектов с объектами Excel, а не Word (редактор Visual Basic был открыт из Excel). Члены объекта Workbook отображаются на правой панели. Метод PrintOut выделен, поэтому на самой нижней панели показан его (сложный) синтаксис вызова. Преимущество этого подхода перед справкой заключается в том, что обозреватель объектов фактически просматривает библиотеку типов, поэтому отображаемый им список с большей вероятностью будет правильным, чем справка. Более того, обозреватель объектов и справка могут работать вместе. Нажмите F1 в браузере, и откроется справка по выделенному элементу.

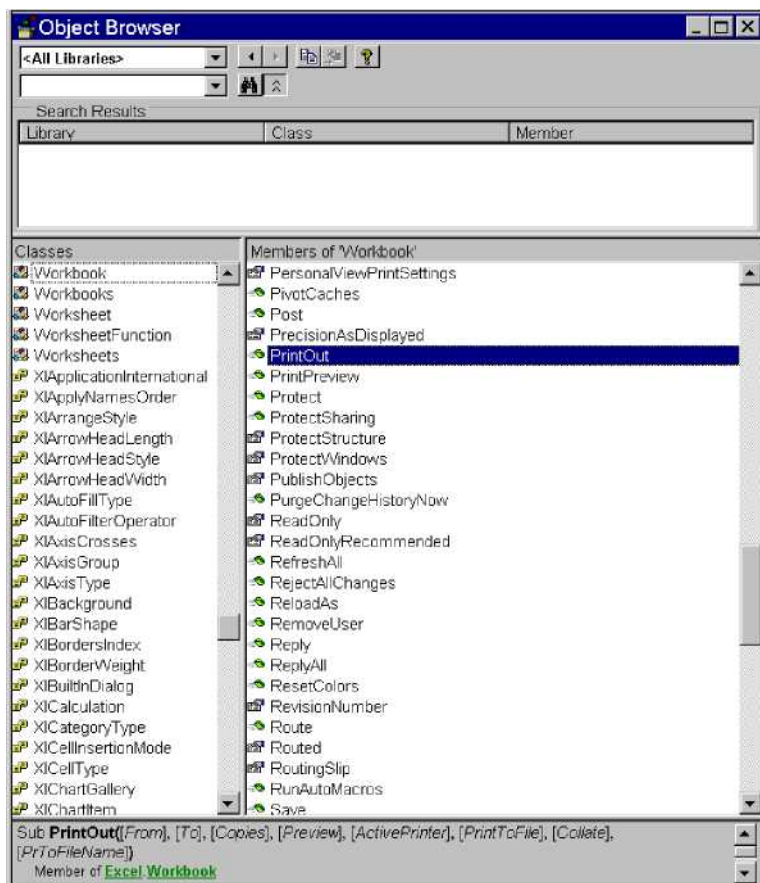


Рисунок 6. Использование Обозревателя объектов для определения параметров.

Когда метод выделен, на нижней панели отображается синтаксис вызова.

Поскольку браузер получает информацию непосредственно с сервера, это не может быть ошибкой.

Браузер также полезен для изучения самой иерархии объектов. На рисунке 7 показана версия обозревателя объектов PowerPoint (на этот раз VBE был открыт из PowerPoint). Члены объекта Presentation показаны на правой панели. Свойство Slides выделено. На нижней панели мы узнаем, что Slides – это ссылка на коллекцию Slides. Нажав на подчеркнутые Slides, мы попадем в коллекцию слайдов, показанную на рисунке 8.

Что просматривает браузер?

На рисунке 6 мы отметили, что обозреватель объектов был открыт из Excel. В этом не было особой необходимости. Вы можете использовать Обозреватель объектов из любого инструмента Office, чтобы открыть и изучить любую зарегистрированную библиотеку типов. Вы даже можете просматривать объекты из нескольких

библиотек типов одновременно.

Чтобы открыть библиотеку типов, чтобы обозреватель объектов мог отображать ее содержимое, выберите «Инструменты|Ссылки» в редакторе Visual Basic. Откроется диалоговое окно, показанное на рисунке 9. Проверьте библиотеки, которые вы хотите добавить в обозреватель объектов, а затем нажмите «ОК». (Имейте в виду, что если вы на самом деле пишете код в VBE, такое обращение к библиотекам типов имеет последствия для ваших проектов. Поэтому будьте осторожны с тем, что вы на самом деле сохраняете.)

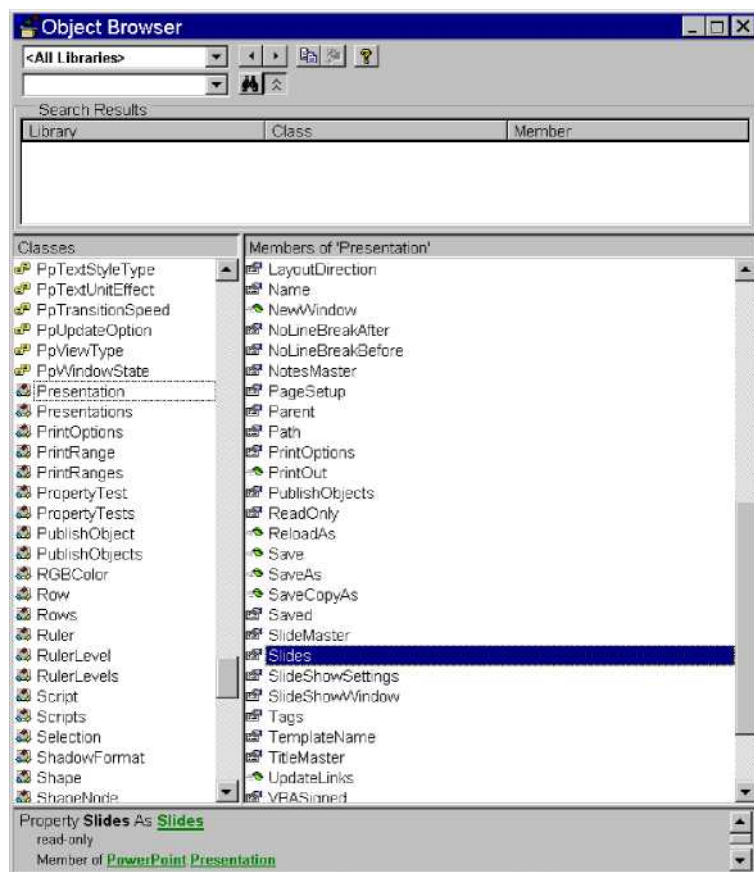


Рисунок 7. Изучение объектной модели. Если элемент подчеркнут на нижней панели, вы можете щелкнуть по нему и изменить фокус в браузере. Нажмите «Слайды», подчеркнутые здесь, чтобы перейти к отображению, показанному на рисунке 8.

В обозревателе объектов вы определяете, будете ли вы видеть информацию из одной библиотеки типов или из всех, с помощью раскрывающегося списка в верхнем левом углу. Невозможно выбрать разнообразное подмножество открытых библиотек, однако ваш выбор – все или одна.

По вашей команде

Командное окно Visual FoxPro – еще один мощный инструмент для изучения серверов Автоматизации. После того, как вы прочитали справку и просмотрели ее в Обозревателе объектов, иногда вам просто нужно попробовать. Вот тут-то и появляется командное окно.

Как и в любом другом аспекте работы в VFP, командное окно позволяет вам пробовать что-то и видеть, что происходит, без затрат на создание целых приложений или настройки сложных сценариев. Создайте ссылку на соответствующий сервер и попробуйте выполнить последовательность команд одну за другой, наблюдая за результатами по ходу выполнения.

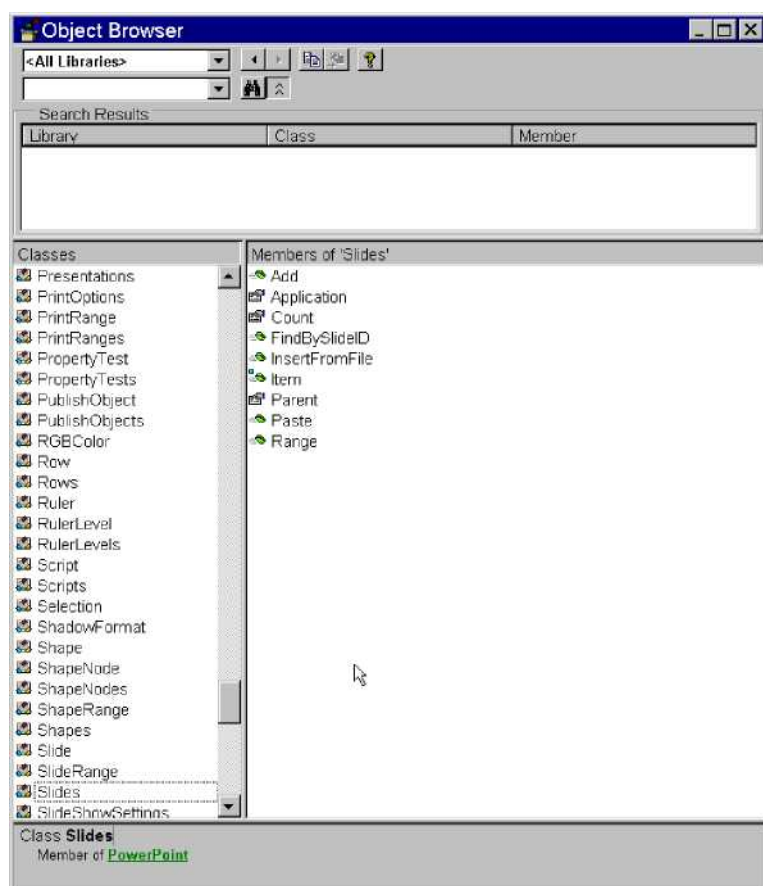


Рисунок 8. Коллекция Slides. Щелчок по ссылке на коллекцию Slides, показанную на рисунке 7, приводит к такому отображению в обозревателе объектов. Браузер позволяет легко исследовать отношения между объектами в иерархии.

Вы можете запросить значение свойства с помощью «?» (при условии, что значение можно распечатать) или выполнить метод. Даже отладчик VFP можно использовать ограниченно. Ограничением является то, что свойства COM-объектов видны в отладчике только после доступа к ним из VFP. Вы не можете просто детализировать COM-объекты в отладчике, как это можно сделать с собственными объектами VFP. Очень жаль.

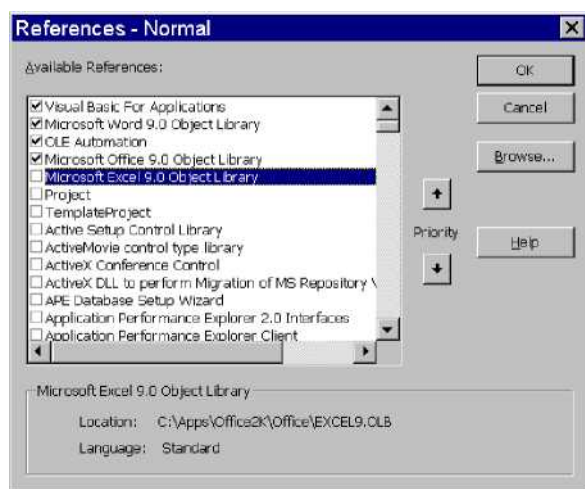


Рисунок 9. Добавление библиотек типов. Вы можете изучить все виды библиотек типов с помощью обозревателя объектов Office.

Мы сочли очень полезным держать командное окно видимым во время написания кода. Некоторые команды, особенно с большим количеством указанных

именованных параметров (и большим количеством пропущенных параметров), могут быть особенно трудными для правильного выполнения. Попробуйте их в командном окне, и когда они наконец окажутся правильными, вырежьте и вставьте их в свой код. Один большой недостаток: командное окно не выполняет `#DEFINE`. Либо задайте переменную в командном окне, либо используйте соответствующие значения (не забывая менять их на константы `#DEFINE` при вставке в код).

Вы можете обнаружить, что получите гораздо больше от этой книги, если «поработаете» над кодом. Вы увидите, что мы предоставили значения для каждой константы в операторах `#DEFINE` в верхней части кода. Хотя строка `#DEFINE` сама по себе не приносит никакой пользы в командном окне, вырезание и вставка целого раздела из HTML-справочной версии этой книги в командное окно, а затем щелчок правой кнопкой мыши и выбор «Выполнить» запускают код правильно.

Еще одна интересная особенность заключается в том, что вы можете переключаться между выполнением действий в интерактивном режиме и выполнением их с помощью Автоматизации. То есть, когда у вас есть экземпляр сервера, доступный из командного окна и видимый, вы можете переключиться на серверное приложение и работать с ним в интерактивном режиме, затем вернуться в VFP и проверить значения свойств, которые только что были установлены вашим действием, или выполнить метод, или установить некоторые другие свойства.

Пытаясь понять, как работает та или иная функция, мы часто пробуем что-то из командного окна, затем переключаемся на серверное приложение, чтобы увидеть результат, затем нажимаем Ctrl-Z (Отменить) на сервере, чтобы отменить это действие, прежде чем мы вернемся к VFP и попробуем другой параметр, значение или подход. Возможно, больше, чем любая другая, эта способность подтверждает тот факт, что Автоматизация на самом деле – это еще один способ делать то же самое, что пользователь может делать в интерактивном режиме.

Мы рекомендуем вам изучить серверы в командном окне. Это действительно помогает мгновенно увидеть, что делает эта строка кода (или это становится мгновенным подходом к поиску синтаксической ошибки, что по-прежнему очень полезно, но смотреть уже не так интересно).

Интернет-ресурсы и печатные ресурсы

Помимо соответствующих файлов справки, для серверов Office доступен ряд справочных материалов. Microsoft Press предлагает *Руководство программиста Visual Basic* как для Office 2000, так и для Office 97. Оба документа доступны как в виде книги, так и в Интернете. Поскольку Microsoft имеет привычку регулярно обновлять свой веб-сайт, лучший способ найти онлайн-версии – это поискать на сайте microsoft.com «Руководство программиста по Visual Basic» ("Visual Basic Programmer's Guide").

Файлы справки VBA также доступны в печатном виде. Если вы предпочитаете работать с бумажной копией, вы можете заказать ее в Microsoft Press. Найдите *справочник по языку Office* (или *справочник по языку* для отдельного приложения, которое вас интересует). Справочные руководства *по языкам* также доступны на веб-сайте Microsoft на случай, если вы застряли где-то без файла справки.

Веб-сайт Microsoft по разработке Office находится по адресу msdn.microsoft.com/officedev/ (по крайней мере, на момент написания этой статьи). Посетите его для получения официальной поддержки, технических статей, исправлений ошибок и т. д., а также указателей на другие полезные сайты.

Как только вы освоитесь с чтением кода VBA, полезными ресурсами могут стать

различные журналы по Office и Visual Basic. Взгляните на *Microsoft Office u Visual Basic для разработчиков приложений* (www.OfficeVBA.com) и *Woody's Office Watch* (www.woodyswatch.com) для начала. Кроме того, новый журнал Advisor Media, *Advisor Expert: Microsoft Outlook and Exchange*, похоже, будет посвящен Автоматизации Outlook (www.Advisor.com). Крупнейшие журналы FoxPro, *FoxPro Advisor* (www.Advisor.com) и *FoxTalk* (www.pinpub.com), иногда освещают вопросы Автоматизации. Серверы Office являются объектом Автоматизации только для некоторых из этих статей.

Если вы застряли, для получения немедленной помощи лучше всего посетить форумы и группы новостей FoxPro. Сообщество FoxPro имеет заслуженную репутацию благодаря своей полезности. Достаточное количество участников выполняют работу по Автоматизации, поэтому на простые вопросы можно быстро получить ответы. Более сложные вопросы иногда остаются без ответа. Также существуют онлайн-сообщества по приложениям Office. У нас меньше опыта работы с ними, но мы обнаружили, что те, с кем мы имели дело, были дружелюбными и знающими. См. Приложение А, «Онлайн-сообщества пользователей», где приведен список онлайн-сообществ пользователей для Visual FoxPro и приложений Office.

2.2 Собираем коллекцию

Объектные модели приложений Office (наряду с большинством COM-серверов) содержат множество коллекций – ООП-версию массивов. Коллекция содержит ссылки на несколько объектов, обычно одного типа, и обеспечивает доступ к этим объектам. Обычно коллекция имеет имя во множественном числе, например «Слайды», и содержит объекты, на которые имеются ссылки в единственном числе: коллекция «Слайды» содержит серию объектов «Слайд». Просто чтобы убедиться, что вы хорошо разбираетесь в файле справки и не запутались, в большинстве случаев вы получаете доступ к коллекции с помощью свойства с тем же именем: объект Presentation имеет свойство Slides, которое ссылается на коллекцию Slides. Коллекция Slides ссылается на серию объектов Slide. Просто помните, что коллекция имеет множественное число (как обычно и свойство доступа к ней), а объект – единственное, и у вас не должно возникнуть особых проблем.

Большинство коллекций, в том числе в Office, имеют несколько стандартных методов и свойств. Свойство Count сообщает вам, сколько элементов находится в коллекции. Item, который является методом в одних коллекциях и свойством в других (это метод в приложениях Office), обеспечивает доступ к отдельным членам коллекции.

Item обычно принимает один параметр или индекс (число) и возвращает ссылку на этот член коллекции. Во многих случаях вы можете указать нужный элемент по имени, а не по номеру. Например, рассмотрим коллекцию Projects Visual FoxPro (которая является коллекцией COM, а не собственной коллекцией VFP). Если проект TasTrade открыт и является первым открытым проектом, вы можете получить к нему доступ как `_VFP.Projects.Item[1]` или `_VFP.Projects.Item["TasTrade.PJX"]`.

Кроме того, большинство коллекций позволяют опустить ключевое слово Item и просто присоединить параметр/индекс к имени коллекции. Таким образом, вы можете написать `_VFP.Projects["TasTrade.PJX"]` или `_VFP.Projects[1]` и при этом получить ссылку на проект.

Как и в случае с массивами Visual FoxPro, вы можете использовать квадратные или круглые скобки для доступа к элементам коллекции. В этой книге мы используем квадратные скобки как для массивов, так и для коллекций, а также оставляем

круглые скобки для обозначения функций и методов.

Изменение коллекции

Как только вы преодолеете Count и Item, появится больше разнообразия. Большинство, но не все, коллекций имеют метод Add, который позволяет добавить в коллекцию новый элемент соответствующего типа. Например, вы используете метод Add коллекции документов Documents Word для создания нового пустого документа, а метод Add коллекции Workbooks Excel – для создания новой пустой книги. Фактически, вы также используете метод Add коллекции Worksheets Excel, чтобы добавить новый лист в существующую книгу.

Не существует единого метода удаления элементов из коллекций. Это связано с тем, что некоторые типы элементов удаляются сами, когда они больше не принадлежат коллекции. Например, когда вы закрываете Workbook в Excel, она удаляется из коллекции Workbooks.

Методы, удаляющие объекты из коллекций, обычно принадлежат самому объекту, а не коллекциям. Хотя добавление и удаление могут показаться нам взаимодополняющими операциями, с объектной точки зрения это не так. При добавлении у вас есть только коллекция для работы; у вас еще нет того, что вы добавляете. Когда вы будете готовы удалить его, он будет у вас под рукой и вы сможете попросить его удалиться самостоятельно. Таким образом, методы Add принадлежат коллекциям, а методы Remove (или методы, вызывающие удаление членов) принадлежат членам коллекции.

Некоторые коллекции имеют фиксированное количество записей. Например, коллекция Borders в Excel представляет четыре границы диапазона. В этой коллекции всегда четыре предмета; нет возможности добавлять в него элементы или удалять их. (Word и PowerPoint также имеют аналогичные коллекции Borders с фиксированным количеством элементов.)

Другие коллекции изменяются другими действиями в системе. Например, коллекция Revisions Word («Редакции») содержит изменения, внесенные в документ. Вы не можете добавлять или удалять элементы напрямую, поскольку они обрабатываются с помощью другого механизма.

2.3 Самоссылающиеся объектные модели

Когда вы начнете копаться в объектных моделях Office, это может немного сбить с толку (ладно, очень сбить с толку!). Вы обнаружите, что у одного объекта есть свойство, которое является ссылкой на коллекцию объектов другого типа или, возможно, это ссылка на другой объект. Свойство часто (но не всегда) имеет то же имя, что и коллекция или другой объект. Когда вы копаетесь в справке, иногда требуется два или три прыжка, чтобы найти что-то полезное.

Вот пример. Допустим, вы начинаете с диаграммы объектной модели Word и нажимаете на коллекцию «Документы». Вы перейдете на страницу справки для коллекции документов (показано на рисунке 2, вид за диалоговым окном). Однако вас, вероятно, интересует объект Document, а не коллекция, поэтому на странице «Документы» вы нажимаете «Документ». Вы попадете на страницу справки по документу. Отсюда вы можете узнать о свойстве «Абзацы», поэтому нажимаете «Свойства». На рисунке 10 показана страница «Документ» и появившееся диалоговое окно.

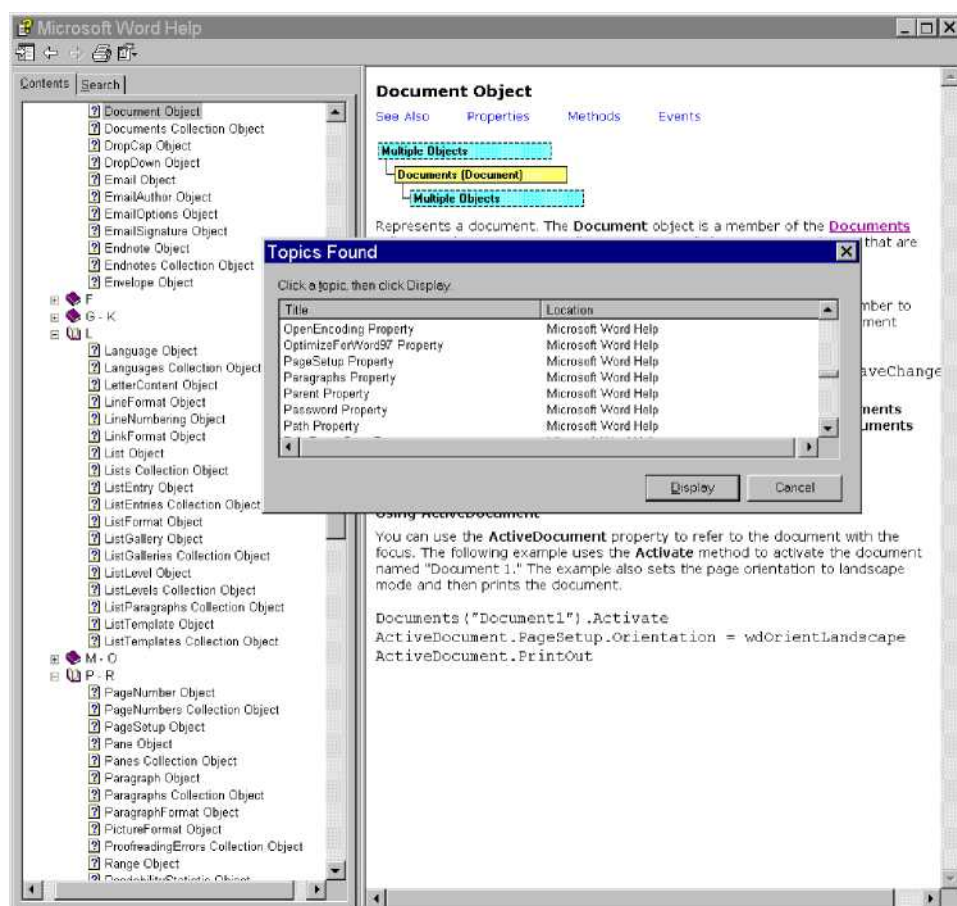


Рисунок 10. От документа к абзацам. Многие объекты имеют свойства, которые ссылаются на коллекции, используя то же имя, что и коллекция. Хотя это имеет смысл, это может сделать получение помощи немного затянутым, особенно потому, что часто в конечном итоге мы направляемся к отдельному свойству, а не к коллекции.

При выборе свойства «Абзацы» в диалоговом окне открывается справка по этому свойству, как показано на рисунке 11. Чтобы перейти к справке по коллекции «Абзацы», необходимо нажать «Абзацы»; Чтобы добраться до фактического объекта «Абзац», обычно для нашего фактического пункта назначения требуется еще один щелчок, как только мы доберемся до «Абзацев». (Свойство «См. также» для «Абзацев» на самом деле предлагает прямой переход к «Абзацу», но чтобы это узнать, вам придется нажать «См. также».) Это одна из наших наименее любимых особенностей справочной системы VBA. Нам бы хотелось, чтобы у неё хватило ума предложить какую-то консолидированную помощь для свойства и коллекции.

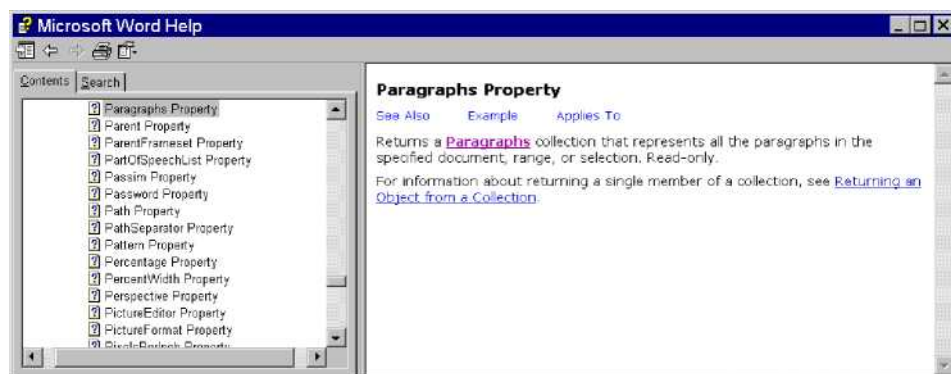


Рисунок 11. Не очень полезно. Эта запись справки и другие подобные ей, хотя и точны, добавляют дополнительный шаг при навигации с одного объекта на другой в

справочной системе.

Есть и обратная сторона проблемы одноимённых свойств и коллекций/объектов. В некоторых ситуациях свойства, которые обращаются к коллекциям или другим объектам, не имеют тех же имен, что и их целевые объекты. Например, ряд объектов ссылаются на объект ParagraphFormat Word через свойство Format. Поскольку ссылочные свойства почти всегда используют имена объектов, обратите внимание на особые случаи.

2.4 Двигаемся дальше

Один из самых важных уроков, которые мы усвоили при написании кода Автоматизации, заключается в том, что невозможно автоматизировать то, чего не понимаешь. В этой главе мы рассмотрели различные способы знакомства с серверами Автоматизации в Microsoft Office, но есть еще один, о котором мы не упомянули.

Используйте продукты. Мы используем Office. Word, в частности, является частью нашей обычной рабочей среды, и многое из того, что мы знаем об его Автоматизации, основано на приобретенном с трудом опыте его использования. (С другой стороны, мы бы солгали, если бы не признали, что стали более опытными пользователями Word благодаря вещам, которые мы узнали из программирования Автоматизации.)

Если вы застряли в сложной задаче Автоматизации, проделайте ее несколько раз в интерактивном режиме, пока не поймете по-настоящему и процесс, и результат. Вы не ожидаете, что сможете писать хорошие программы на языке, в среде которого вам неудобно работать. Почему вы ожидаете сделать это с помощью Автоматизации?

Добавьте такую практику к инструментам исследования, описанным в этой главе, и вы почувствуете себя комфортно в среде Office Automation еще до того, как узнаете об этом. Теперь перейдем к роли Visual FoxPro как клиента Автоматизации.

Глава 3. Visual FoxPro как клиент Автоматизации

Некоторые аспекты Visual FoxPro актуальны независимо от того, какой продукт вы автоматизируете.

Независимо от того, что вы автоматизируете, многое из того, что вы делаете в Visual FoxPro, остается неизменным. В этой главе рассматриваются вещи, которые остаются одинаковыми на всех серверах и на серверах Office.

3.1 Управление серверами

Первый шаг при работе с любым сервером – получение ссылки. Существует две возможности для сервера Автоматизации: вы можете создать новый экземпляр или получить ссылку на существующий сервер, если он существует. Вы также можете использовать две функции VFP: `CreateObject()` и `GetObject()`, но они не соответствуют этим двум методам.

`CreateObject()` всегда создает новый экземпляр сервера и возвращает ссылку на него. Вот синтаксис для создания объектов Автоматизации:

```
oServer = CreateObject(cServerClass)
```

`cServerClass` – подходящее имя для основного класса сервера Автоматизации. Для приложений Office это «<имя приложения>.Application», где <имя приложения> – это «Word», «Excel» или что-то еще. Например, чтобы открыть PowerPoint, введите следующую команду:

```
oPowerPoint = CreateObject("PowerPoint.Application")
```

Обратите внимание, что синтаксис создания объектов Автоматизации практически идентичен синтаксису создания собственных объектов. Единственная разница – это класс создаваемого объекта. Это полиморфизм (один из столпов ООП) в действии.

Если сервер уже открыт и вы хотите использовать существующий экземпляр, вместо этого вы можете использовать функцию `GetObject()`. `GetObject()` принимает два параметра. В этом случае опустите первый параметр и передайте класс сервера в качестве второго. Вот синтаксис:

```
oServer = GetObject( , cServerClass)
```

Если сервер уже открыт, `GetObject()` находит его и возвращает ссылку на этот экземпляр. Однако если сервер не открыт, эта версия функции выдает ошибку.

Например, чтобы прикрепиться к открытому экземпляру Excel, используйте следующую команду:

```
oExcel = GetObject( , "Excel.Application")
```

Есть еще один способ запустить сервер, передав `GetObject()` имя файла и позволив ему определить, какой сервер использовать:

```
oDocument = GetObject(cFileName)
```

VFP ищет расширение файла в реестре, чтобы выяснить, какому приложению он принадлежит, открывает это приложение, затем открывает файл; это работает почти так же, как если бы вы дважды щелкнули файл в проводнике Windows, за исключением того, что вы получаете объектную ссылку на файл как на «документ», так что вы можете манипулировать им как объектом. В этом примере открывается файл «C:\Documents\MyFile.DOC» в Word и возвращается ссылка на него:

```
oDocument = GetObject("C:\Documents\MyFile.DOC" )
```

В некоторых ситуациях имени файла может быть недостаточно для определения

того, какое приложение открыть или что делать с файлом. В этом случае вы также можете передать имя класса на сервере. Например, если файл имеет расширение TXT, но вы хотите, чтобы Word открывал его как документ, передайте «Word.Document» в качестве второго параметра, например:

```
oDocument = GetObject( "Пример.TXT", "Word.Document")
```

Однако вы не можете просто передать тип файла автоматически. В большинстве случаев либо VFP, либо серверное приложение зависает, когда получает второй параметр без необходимости.

Следует отметить, что в этом семействе есть третья функция VFP — CreateObjectEX(). Она используется для создания объектов на другом компьютере и является частью DCOM. Поскольку в этой книге не рассматривается DCOM, мы не обсуждаем её здесь.

Отображение серверов Office

Когда вы создаете экземпляр сервера Автоматизации, по умолчанию он невидим. Он не отображается на панели задач. В Windows NT он отображается на странице «Процессы» диспетчера задач, но не на странице «Приложения». Он отображается в диалоговом окне «Заккрытие программы» в Windows 95/98. В целом хорошо, что он держит себя в некотором роде скрытым. Часто вы делаете что-то за кулисами, и у пользователя нет причин видеть это. Однако во время отладки и в некоторых других случаях вам может потребоваться сделать процесс Автоматизации видимым.

Почему бы просто не сделать серверное приложение видимым постоянно? Скорость. Неудивительно, что манипулирование документами происходит быстрее, когда вы их не видите. Это также более аккуратно, хотя первые несколько раз наблюдать за создаваемой электронной таблицей или презентацией довольно здорово, но через некоторое время пользователи, вероятно, устанут от просмотра.

Это одна из областей, где Outlook является лишним. Эти команды применимы к Word, Excel и PowerPoint (и, если уж на то пошло, к Visual FoxPro, когда он используется в качестве сервера), но не к Outlook. Чтобы сделать сервер Автоматизации видимым, установите для его свойства Visible значение .T.; установите Visible на .F. чтобы выключить его. (На самом деле в PowerPoint вы можете установить для параметра Visible значение .T., но установка значения .F. приведет к появлению сообщения об ошибке. Word и Excel дают вам полный контроль над их видимостью.) Outlook не имеет аналогичного свойства.

Свойство WindowState объекта Application определяет, является ли приложение свернутым, развернутым или «нормальным», что означает некоторый размер, определяемый пользователем. Манипулирование WindowState, когда приложение невидимо, обычно делает его видимым (по крайней мере, в той степени, в которой оно отображается на панели задач). Каждое из приложений имеет набор констант для трех возможных значений WindowState. Например, для Excel вы можете сделать такие определения:

```
#DEFINE xlMaximized -4137
#DEFINE xlMinimized -4140
#DEFINE xlNormal -4143
```

Для Word необходимо следующее:

```
#DEFINE wdWindowStateNormal 0
#DEFINE wdWindowStateMaximize 1
#DEFINE wdWindowStateMinimize 2
```

Объект Application всех трех приложений имеет свойства Left и Top, определяющие

его расположение на экране, а также свойства Height и Width. Вы можете манипулировать ими, если только приложение не развернуто.



Следующий код (SetSize.PRG в файлах загрузки для разработчиков, доступных по адресу www.hentzenwerke.com) открывает Word, устанавливает окно приложения в обычный режим, размещает его немного ниже левого верхнего угла экрана, а затем делает его видимым.

```
* SetSize.PRG
* © 2000, Tamar E. Granor and Della Martin
* From: Microsoft Office Automation with Visual FoxPro
* Hentzenwerke Publishing. www.hentzenwerke.com

#DEFINE wdWindowStateNormal 0

LOCAL nScreenHeight, nScreenWidth
LOCAL nWindowHeight, nWindowWidth

* Вычисление размеров
nScreenHeight = SYSMETRIC(2)
nScreenWidth  = SYSMETRIC(1)

* Сделайте его равным двум третям размера экрана в каждом
* измерении.

nWindowHeight = INT(.67 * nScreenHeight)
nWindowWidth  = INT(.67 * nScreenWidth)

RELEASE ALL LIKE o*
PUBLIC oWord
oWord = CreateObject( "Word.Application" )

WITH oWord
    .WindowState = wdWindowStateNormal

    .Height = .PixelsToPoints( nWindowHeight, .T. )
    .Width  = .PixelsToPoints( nWindowWidth,  .F. )

    .Top    = 10
    .Left   = 10

    .Visible = .T.
ENDWITH

RETURN
```

Как видно из примера, Height, Width, Top и Left измеряются в пунктах. Word предоставляет ряд методов для преобразования других измерений в пункты (в том числе в Word 2000 метод PixelsToPoints, использованный в примере). Другие приложения предлагают меньше таких методов; фактически, ни Excel, ни PowerPoint не включают PixelsToPoints. Коэффициент преобразования между пикселями и пунктами составляет около 0,75.

Если вы не выполняете очень точную работу, вы, вероятно, сможете жить с этим коэффициентом преобразования, поэтому в Excel и PowerPoint просто определите свое собственное преобразование, например:

```
#DEFINE autoPixelsToPoints .75
```

Вы можете выполнить преобразование между дюймами и пунктами таким же образом. В одном дюйме 72 пункта. Определите свою собственную константу для этой задачи, например:

```
#DEFINE autoInToPts 72
```

Даже когда вы автоматизируете Word, есть аргумент в пользу того, что вам лучше выполнять собственные преобразования в VFP, чем использовать встроенные методы Word. Каждый вызов Word стоит дорого. В наших тестах арифметика в VFP выполнялась примерно в 100 раз быстрее, чем вызов методов преобразования Word.

Мы уже на месте?

Все три приложения запускаются без открытия документа и без окна документа, когда вы вызываете CreateObject(), но они ведут себя по-разному, когда GetObject() используется для открытия определенного файла.

В Word после завершения GetObject() появляется окно документа, содержащее указанный документ. Просто установите для Visible значение .T. для объекта Application, и окно документа отобразится:

```
oDocument.Application.Visible = .T. && это необходимо для любого  
&& приложения но измените переменную соответствующим образом
```

PowerPoint и Excel – это разные вещи. Установки параметра Visible на .T. недостаточно. В PowerPoint вам необходимо вызвать метод NewWindow, чтобы предоставить окно документа для указанной презентации:

```
oPresentation.NewWindow()
```

Excel использует еще один подход к той же проблеме. Есть окно документа; вы просто не можете его увидеть. Вызовите метод Activate для первого окна в коллекции Windows:

```
oWorkbook.Windows[1].Activate()
```

3.2 Работа с серверами

Некоторые функции языка Visual FoxPro и компоненты интерфейса упрощают написание и поддержку кода Автоматизации. Ниже мы рассмотрим некоторые вещи, которые следует помнить, работая в диком мире Автоматизации.

SET OLEOBJECT

Эта команда VFP определяет, будет ли VFP выполнять поиск в реестре при использовании CreateObject() или GetObject(). Если для него установлено значение ON, VFP выполняет поиск в реестре; когда для него установлено значение OFF, этот поиск пропускается. Почему существует эта команда? Последнее место, где VFP ищет объект, – это реестр. Перед поиском в реестре загружается поддержка OLE, которая занимает память. Если вашему приложению не требуется поддержка OLE, установка для OLEOBJECT значения OFF обеспечит некоторое повышение производительности. Однако если вы используете Автоматизацию, вам потребуется поддержка OLE, а также возможность найти сервер в реестре. Если вы попытаетесь создать экземпляр сервера Автоматизации, когда для параметра SET OLEOBJECT

установлено значение OFF, генерируется ошибка типа «Определение класса WORD.APPLICATION не найдено».

Используйте WITH ... ENDWITH

Большая часть кода, который вы пишете для Автоматизации любого сервера, включает установку свойств или вызов методов. Обычно длинные фрагменты кода состоят не более чем из ссылок на свойства и методы, возможно, с добавлением небольшого количества арифметики или логики. Вы можете сделать этот код гораздо более читабельным (и, следовательно, более простым в отладке и сопровождении) с помощью команды VFP WITH ... ENDWITH.

Серия команд, начинающаяся с чего-то вроде oWord.ActiveDocument.Tables[3].Rows[7], просто не обеспечивает удобочитаемости. Вместо этого окружите группу следующим образом:

```
WITH oWord.ActiveDocument.Tables[ 3 ].Rows[ 7 ]
    * поместите сюда команды с точкой перед каждой
ENDWITH
```

Фактически, вы можете вкладывать команды WITH. По мере того, как вы спускаетесь по иерархии объектов, выполняя несколько действий на каждом уровне, настройте оператор WITH для каждого уровня примерно так:

```
#DEFINE wdAlignRowCenter 1
#DEFINE wdRowHeightExactly 2

WITH oWord.ActiveDocument
    * Сделайте что-нибудь с документом в целом, например
    .Save && сохраните, используя текущее имя файла

    * Затем перейдите к таблице.
    WITH .Tables[ 3 ]
        * Теперь выдавайте команды, направленные на таблицу в целом.
        .AllowPageBreaks = .T. && Разрешить таблице разбиваться на
            && страницы.

        * Затем, когда вы будете готовы поговорить с одной строкой.
        WITH .Rows[ 7 ]
            * Теперь выполните команды для одной строки
            .Alignment = wdAlignRowCenter
            .HeightRule = wdRowHeightExactly
            .Height = .5
        ENDWITH
    ENDWITH
ENDWITH
```

Вложенность дает понять читателю, к какому WITH принадлежит каждое свойство. VFP, конечно, без труда во всем этом разбирается. Помимо читабельности, есть дополнительный бонус. Подобный код работает быстрее. FoxPro не нужно перебирать несколько уровней иерархии, чтобы определить, какому объекту принадлежит данное свойство или метод. В наших тестах довольно простой пример, в котором запрашивалось около дюжины свойств на четыре уровня ниже объекта Application, работал примерно в два раза быстрее с использованием вложенных операторов WITH, чем при непосредственном обращении к каждому свойству.

Также обратите внимание, что не существует правила, согласно которому свойство или метод должно быть первым в строке кода. Часто так и получается, но их можно использовать и в другом месте. Например, допустим следующий код:

```
WITH oExcel  
    nHeight = .Height  
    nWidth  = .Width  
ENDWITH
```

Фактически, как показано в примере из раздела «Отображение серверов Office» ранее в этой главе, вы можете вызывать методы, выполнять вычисления и вообще делать все, что вы обычно делаете внутри пары WITH ... ENDWITH, если вы обязательно ставите точку перед именем свойства или метода.

Используйте переменные для ссылок на объекты

Еще один способ сделать ваш код более читабельным и ускорить его – назначить сложные ссылки на объекты локальным переменным. Даже если пара WITH ... ENDWITH не требуется, возможно, лучше присвоить что-то вроде oPowerPoint.ActivePresentation переменной VFP с именем типа oPres (или oPresentation, если вы предпочитаете). Имя переменной легче набирать и легче читать. Как и в случае с оператором With, он дает VFP прямой маршрут к интересующему вас объекту, а не просит его спуститься вниз по иерархии объектов.

Мы протестировали тот же пример, что и для With (запрос свойств на различных уровнях), и обнаружили, что установка локальной переменной происходит так же быстро или даже немного быстрее, чем при использовании With. Мы подозреваем, что точная точка компромисса варьируется в зависимости от таких факторов, как доступная память, количество ссылок внутри With на локальную переменную и т. д. Однако нет никаких сомнений в том, что любой подход значительно быстрее, чем запись длинной ссылки на объект Автоматизации. Чем глубже вложена ссылка и чем больше раз она вам понадобится, тем больше времени вы сэкономите.

Если вы используете локальные переменные, вам может потребоваться впоследствии выполнить очистку. В некоторых ситуациях эти ссылки могут помешать закрытию сервера при вызове метода Quit.

Цикл с FOR EACH

В главе 2 «Серверы Office» мы обсуждали распространенность коллекций в объектных моделях Office (и в объектных моделях COM в целом). Если вам нужно обработать все элементы коллекции, лучше всего подойдет цикл VFP FOR EACH. FOR EACH позволяет просматривать коллекцию (или массив), не используя счетчик и не беспокоясь о том, сколько в ней элементов. Вот синтаксис FOR EACH:

```
FOR EACH oMember IN oCollection  
    * выдайте команды для oMember  
ENDFOR
```

Например, чтобы отобразить имя каждого открытого документа в Word, вы можете использовать этот код:

```
FOR EACH oDocument IN oWord.Documents  
    ? oDocument.Name  
ENDFOR
```

Обратите внимание, кстати, что использование FOR EACH подразумевает использование локальной переменной, как описано в предыдущем разделе, ссылки на объект, используемой в качестве переменной цикла.

3.3 Отладка

Отладка кода Автоматизации представляет собой сложную задачу. Как мы отмечали в главе 2, отладчик VFP не слишком помогает. В окнах «Watch» и «Locals» свойство объекта Автоматизации отображается только после того, как к этому свойству был получен доступ из VFP. Таким образом, вы не можете углубиться в эти объекты, чтобы увидеть, что происходит, и по-настоящему изучить их, как это можно сделать с собственными объектами. Мы также иногда обнаруживали, что помещение COM-объектов (будь то объекты Автоматизации или другие COM-объекты) в окно Watch может быть хорошим способом аварийного завершения работы отладчика.

Так как же отлаживать код Автоматизации? Тщательно и с большим заблаговременным планированием.

Работа в командном окне

Мы проводим большую часть тестирования и разработки в командном окне. Гораздо проще понять, что происходит, если мы выдаем команды по одной и проверяем результаты. Как только мы уверены, что знаем, какие методы вызывать и какие свойства устанавливать для создания нужных документов, мы помещаем их в код. Мы считаем, что этот подход гораздо более необходим (смеем сказать, обязателен) для кода Автоматизации, чем для чистого кода VFP.

Коллекции и объекты

Следите за коллекциями и отдельными объектами. Поскольку большая часть кода Автоматизации включает в себя коллекции, очень легко вызвать метод коллекции, когда вы хотите вызвать его для объекта, или наоборот. Например, когда вы видите что-то вроде следующего, имейте в виду, что объект, на который оно ссылается, является объектом, *а не* коллекцией:

```
oPowerPoint.Presentations[1]
```

(На самом деле это еще один аргумент в пользу использования локальных переменных. Установка oPresentation для ссылки на эту презентацию помогает избежать путаницы.) Полезный совет: обычно объект коллекции имеет множественное число (например, Presentations), а сам объект – единственное число (например, Presentation). В файле справки довольно легко запутаться в поисках методов Presentation и методов Presentations, но умение искать букву «s» (или не искать) было для нас большим прорывом.

Это работает, но это не так

Если ваш код выполняется, но результаты неверны, убедитесь, что вы манипулируете правильными объектами. Еще раз проверьте соответствующие разделы в этой книге и в справке. Убедитесь, что вы можете выполнить задачу (или хотя бы прототип задачи) в интерактивном режиме. Мы повторим еще раз: нельзя автоматизировать то, чего не понимаешь. Попробуйте записать макрос, выполняющий то, что вы пытаетесь автоматизировать, а затем проверьте полученный код VBA, чтобы посмотреть, проливает ли он какой-то свет на ситуацию.

Проблемы с параметрами

Если у вас возникли проблемы с ошибками несоответствия типов при вызовах

методов, попробуйте изменить порядок параметров. Иногда (хотя и гораздо реже, чем в старых версиях Office) в документации ошибаются. Используйте обозреватель объектов, чтобы убедиться в правильном порядке. (Подробнее об этом см. главу 2 «Офисные серверы».)

Аналогичным образом, если у вас возникли проблемы с недопустимым количеством параметров, вам может потребоваться добавить запятые, чтобы разрешить пропущенные значения по умолчанию. Еще раз зайдите в обозреватель объектов, чтобы увидеть правильный список параметров метода.

Бесполезные сообщения об ошибках

Когда ошибка возникает в команде Автоматизации, она попадает в одну из трех широких категорий: вы попросили VFP сделать что-то, что он не может сделать, вы попросили сервер сделать что-то, что он не может сделать, или что-то пошло не так во взаимодействии между ними.

В первом случае (проблема VFP) вы получаете обычное сообщение об ошибке VFP и можете обработать его так же, как и любое другое. В этой ситуации следует проверить совпадение круглых скобок, несовпадающие типы данных и т. д.

Во втором случае вы увидите одно из немногих сообщений об ошибках, которые появляются при возникновении ошибки OLE. Почти всегда это либо 1426, либо 1429. Вместе с ним вы получаете сообщение об ошибке разного информационного содержания. Иногда это действительно информативно. Например, если вы запросите `oWord.ActiveDocument.Name` при отсутствии активного документа, сообщение об ошибке будет включать в себя: «Эта команда недоступна, поскольку ни один документ не открыт». Это очень хорошо.

Но в других случаях сообщения довольно скудны. Например, запрос несуществующего свойства или неправильное написание метода приводит просто к «Неизвестному имени». Мы думаем, что фраза «Нет такого участника» была бы намного полезнее. Действительно полезно было бы что-то вроде: «У этого объекта нет члена с именем `Mane`». Затем мы быстро заметили, что неправильно набрали «`Name`» как «`Mane`» (снова). Но, похоже, это требует слишком многого.

Реальная проблема заключается в том, что Visual FoxPro не может предоставить вам больше информации об ошибке, чем предоставляет приложение, с которым он взаимодействует. Это потому, что Visual FoxPro знает только то, что ему говорит сервер. Когда возникает одна из этих ошибок, вам нужно внимательно изучить синтаксис вашей команды, сравнить его с ожидаемым синтаксисом и попытаться выяснить, где вы ошиблись.

Третья группа ошибок отражает трудности взаимодействия двух приложений. Это может произойти, если сервер неожиданно отключается (скажем, если пользователь закрывает сервер вне вашего приложения) или если возникла проблема с открытием сервера. См. главу 14 «Обработка ошибок Автоматизации» для получения идей о том, как корректно решать эти проблемы.

Используйте базу знаний

Хотя найти информацию в базе знаний Microsoft может быть сложно, там собрано огромное количество информации. Если вы застряли в сложной задаче, стоит покопаться и попробовать несколько комбинаций поиска. Если у вас возникла конкретная проблема, скорее всего, она была у кого-то еще до вас.

Попробуйте такие комбинации, как «Automation», «FoxPro» и название приложения, с которым у вас возникли проблемы. Если это не сработает, попробуйте исключить FoxPro, возможно, это проблема и в VBA. Найдите конкретный код ошибки (не

номер ошибки VFP, а код ошибки OLE, возвращаемый сервером) или используйте фразу из описания ошибки, которое возвращается с сервера.

Поспрашивать

Все еще застряли? Существует множество мест, где можно получить онлайн-справку по VFP и приложениям Office. Ознакомьтесь со списком в Приложении А «Интернет-сообщества пользователей». Обязательно предоставьте в своем сообщении достаточно информации, чтобы эффективно описать вашу проблему, но не настолько подробно, чтобы ваше сообщение не было перегруженным. Часто сам процесс попытки проверить возникновение вашей ошибки с помощью наименьшего количества простых шагов приведет вас к решению, хотя на самом деле в самом простом случае ошибка не возникает. Добавляйте сложность обратно, по одной команде за раз, и вы часто сможете изолировать источник вашей проблемы.

3.4 Время для кода

На этом этапе пришло время углубиться и начать изучать продукты Office. В следующих четырех разделах книги рассматриваются все рассматриваемые продукты Office, начиная с Word. Каждый раздел начинается с главы «Основы», а затем развивается в другом направлении, движимом самим продуктом.

Глава 4. Основы Word

Дизайнер отчетов Visual FoxPro не может сделать то, что вам нужно? Вместо этого обратитесь к Word. Его ориентация на документ обеспечивает форматирование от шрифтов до структуры.

Microsoft Word – невероятно мощный текстовый процессор. Он позволяет пользователям создавать документы, начиная от простых заметок и заканчивая сложными корпоративными отчетами, состоящими из нескольких частей. Он также предоставляет инструменты для обработки и управления этими документами различными способами, такими как проверка орфографии и грамматики, объединение с различными источниками данных, отслеживание изменений и многое другое.

Даже для простых документов Word предоставляет множество инструментов, которые позволяют пользователям создавать привлекательные результаты. Пользователи могут работать с несколькими документами одновременно, перемещать и копировать текст внутри документов и между ними, а также отмечать и называть части документа. Форматирование в Word также является весьма мощным инструментом, обеспечивающим локальный контроль, уровень страницы и уровень документа.

4.1 Объектная модель Word

Ключевым объектом в Word является Document, который представляет собой один открытый документ. На сервере Word имеется коллекция документов, содержащая ссылки на все открытые документы. Сервер также имеет свойство ActiveDocument, указывающее на активный в данный момент документ.

Объект Document имеет множество свойств и методов. Многие из его свойств являются ссылками на такие коллекции, как «Абзацы», «Таблицы» и «Разделы». Каждая из этих коллекций содержит ссылки на объекты указанного типа. Каждый объект содержит информацию о соответствующей части документа. Например, объект «Абзац» имеет такие свойства, как KeepWithNext и Style, а также такие методы, как Indent и Outdent.

Объект сервера Word, называемый Application, имеет собственный набор свойств и методов, включая ряд других коллекций. Помимо ActiveDocument, свойства объекта Application включают Visible, StartupPath, Version и WindowState. Объект Application также имеет методы. Самый простой для понимания – Quit, который выключает сервер. Он имеет несколько дополнительных параметров, первый из которых указывает, что делать, если какие-либо открытые документы были изменены, но не сохранены. Другие методы объекта Application преобразуют измерения из одного набора единиц в другой, проверяют грамматику и орфографию и многое другое.

Справка Word Visual Basic содержит диаграмму объектной модели Word. Фигура «живая», когда вы нажимаете на объект, вы попадаете в раздел справки для этого объекта. На рисунке 1 показана часть диаграммы объектной модели, описывающая объект Document.

Microsoft Word Objects (Documents)

See Also

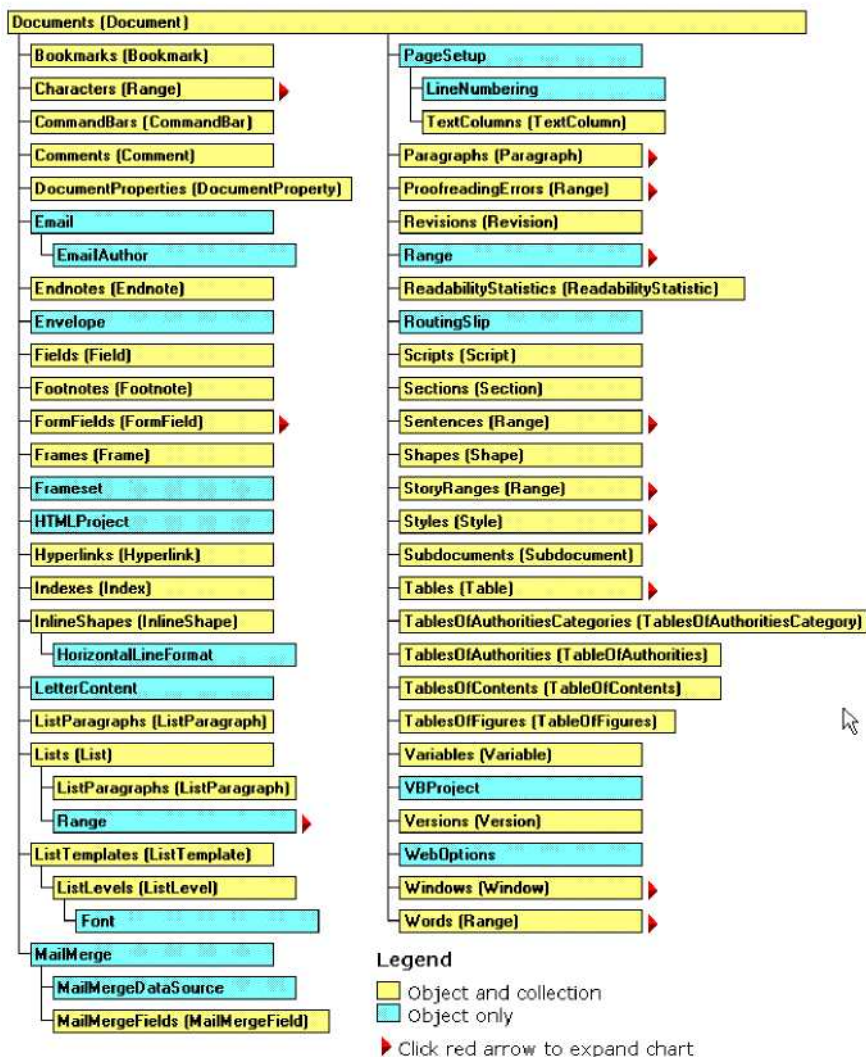


Рисунок 1. Объектная модель Word. Файл справки предлагает глобальное представление структуры Word.

4.2 Переходим к Word

Прежде чем вы сможете работать с Word, вам нужна ссылка на объект сервера Автоматизации Word или на определенный объект в Word. Две функции VFP позволяют получить доступ к Word. `CreateObject()` открывает новый экземпляр Word и возвращает ссылку на него, например:

```
oWord = CreateObject("Word.Application")
```

`GetObject()` немного сложнее. Она принимает любой из двух параметров, которые отражают два подхода к предоставлению ссылки на Word. Если вы передадите ей имя существующего документа, она проверит, открыт ли уже Word. Если нет, открывается Word. В любом случае она открывает указанный документ и возвращает ссылку на документ:

```
oDocument = GetObject("d:\writing\books\automation\chapter4.doc")
```

Если вы опустите первый параметр функции `GetObject()` и укажете сервер Автоматизации Word в качестве второго, она ищет открытый экземпляр Word и возвращает ссылку на этот экземпляр:

```
oWord = GetObject( , "Word.Application")
```

Если Word не открыт, генерируется сообщение об ошибке.

См. главу 3, «Visual FoxPro как клиент Автоматизации», для получения более общей информации о CreateObject() и GetObject() и их использовании с приложениями Office.

4.3 Управляющие документы

Методы создания, открытия, сохранения и закрытия документов довольно просты. Единственная сложность заключается в том, какой объект какой метод предоставляет. Методы создания и открытия документа относятся к коллекции Documents. Методы сохранения и закрытия документа принадлежат объекту Document. Хотя на первый взгляд это кажется запутанным, на самом деле это имеет смысл, поскольку у вас нет объекта Document, с которым можно было бы работать в момент создания или открытия документа. Но когда приходит время сохранить или закрыть его, становится доступна ссылка на документ.

Чтобы открыть существующий документ, используйте метод Open коллекции Documents. Open имеет множество параметров, но в большинстве случаев важен только первый: имя и путь к открываемому файлу. Сервер Word не распознает путь поиска VFP, поэтому обычно вам необходимо указать полный путь к документу, например:

```
oDocument = oWord.Documents.Open("d:\writing\books\automation\chapter4.doc")
```

В случае успеха Open возвращает ссылку на объект вновь открытого документа. Если указанный документ не существует, генерируется ошибка и ничего не возвращается. (В этом примере, как и в других примерах в этой и следующих двух главах, предполагается, что у вас есть работающий экземпляр Word, на который oWord содержит ссылку.)

Чтобы создать новый документ, используйте метод Add, который имеет только два необязательных параметра. Важным является первый, в котором указан путь к шаблону, на котором должен основываться новый документ. Если он опущен, новый документ будет основан на шаблоне Normal. (Шаблоны более подробно обсуждаются в главе 5 «Промежуточный Word».)

Как и Open, Add возвращает ссылку на вновь созданный документ. Эта строка создает новый документ на основе шаблона OfficeFax:

```
oDocument = oWord.Documents.Add( ;  
    "C:\WINNT\Profiles\Tamar\Application  
    Data\Microsoft\OfficeFax.DOT")
```

Как и в случае с именем файла в Open, необходим полный путь к шаблону. См. раздел «Шаблоны документов» в главе 5 для получения информации о том, где Word устанавливает и хранит шаблоны.

Когда вы закончите работу с документом, его можно сохранить двумя способами. Оба метода принадлежат объекту Document. Метод Save сохраняет документ обратно в текущий файл; если он никогда не сохранялся, появится диалоговое окно «Сохранить как». Метод SaveAs позволяет указать имя файла (и многое другое), не видя диалогового окна, что обычно является тем, что вам нужно в коде Автоматизации.

Если текущий активный документ уже был сохранен, эта строка повторно сохраняет документ без вмешательства пользователя:

```
oWord.ActiveDocument.Save()
```


Чтобы сохранить документ в другой файл или сохранить документ в первый раз без указания имени файла пользователем, вызовите SaveAs и передайте имя файла, например:

```
oWord.ActiveDocument.SaveAs ("D:\Documents\ThisIsNew.DOC")
```

Будь осторожны. Когда вы передаете имя файла в SaveAs, он перезаписывает любой существующий файл без запроса. (Конечно, SaveAs, как и другие методы Word, не учитывает настройку SET SAFETY VFP, поскольку он не работает внутри VFP.)

Вы можете проверить, был ли сохранен документ, проверив его свойство «Saved». Если Saved имеет значение .T., документ не изменялся. Это может произойти либо потому, что вы уже сохранили документ и с тех пор не меняли его, либо потому, что это новый документ и еще не был изменен.

Кроме того, свойства Name и FullName предоставляют альтернативный способ проверить, сохранялся ли когда-либо документ. Когда вы создаете новый документ, Word присваивает ему имя в виде «Document n », где n – число. При сохранении документа вы можете дать ему более осмысленное имя, а также указать путь к файлу. Свойство Name документа содержит только основу файла без пути или расширения. Свойство FullName содержит полное имя файла, включая путь и расширение. Однако перед первым сохранением файла и Name и FullName содержат одну и ту же строку – исходное имя документа, присвоенное Word. Вы можете написать такой код, чтобы понять, следует ли использовать Save или SaveAs:

```
WITH oWord.ActiveDocument
  IF .Name = .FullName
    * Предложить пользователю получить имя,
    * затем:
    .SaveAs( cFileName )
  ELSE
    .Save
  ENDIF
ENDWITH
```

4.4 Доступ к частям документа

Большая часть того, что вы хотите сделать с Word, включает в себя добавление, изменение или чтение документа, будь то новый документ, который вы создаете, или существующий документ, который вы изменяете. (На самом деле существует третья возможность. Возможно, вы работаете над новым документом, основанном на шаблоне, содержащем шаблонный текст, который вы настраиваете.) Есть множество способов сделать это, но ключ практически ко всем из них – объект Range и, в меньшей степени, объект Selection.

Объект Selection представляет подсвеченную в данный момент (то есть выделенную) часть документа. Если ничего не выделено, выделение относится к точке вставки. Существует только один объект Selection, доступ к которому осуществляется непосредственно из объекта Word Application. Например, чтобы узнать, сколько абзацев в текущем выделении, вы можете использовать этот код:

```
nParagraphs = oWord.Selection.Paragraphs.Count
```

Объект Range может представлять любую часть документа. Диапазоны – это не то же самое, что область выделения. Вы можете определять или изменять диапазоны,

не затрагивая текущий выбор. Вы даже можете определить несколько диапазонов для документа, тогда как для каждого документа доступен только один объект Selection. Диапазоны очень полезны для многократного обращения к определенным частям документа.

Диапазоны можно получить разными способами. Многие объекты Word, такие как «Абзац» и «Таблица», имеют свойство Range, которое содержит ссылку на объект Range для исходного объекта. Например, чтобы создать диапазон из третьего абзаца активного документа, вы можете использовать следующее:

```
oRange = oWord.ActiveDocument.Paragraphs[3].Range
```

Объект Document имеет метод Range, который позволяет указать диапазон по позиции символа. Например, чтобы получить ссылку на диапазон, содержащий от 100-го до 250-го символа в активном документе (вероятно, это не особенно полезный диапазон), используйте:

```
oRange = oWord.ActiveDocument.Range(100,250)
```

Свойство Content объекта Document содержит ссылку на Range, который состоит из всего основного документа (тела документа без заголовков, нижних колонтитулов, сносок и т. д.). Таким образом, следующие две команды эквивалентны:

```
oRange = oWord.ActiveDocument.Range()  
oRange = oWord.ActiveDocument.Content
```

Будьте осторожны: для большого документа создание такой переменной может занять значительное время.

Объекты Range легко преобразовать в Selection и наоборот. Как и многие другие объекты, Selection имеет свойство Range, которое предоставляет объект Range из Selection. Аналогично, объект Range имеет метод Select, который выделяет содержимое диапазона, превращая его в Selection. Например, чтобы выделить диапазон из предыдущего примера, используйте:

```
oRange.Select()
```

Selection и Range кажутся очень похожими, и они во многом похожи, но есть различия. Самым большим, конечно, является то, что вы можете иметь несколько диапазонов, но только один выбор. Кроме того, работа с диапазоном обычно происходит быстрее, чем с выделением. В целом эксперты Word VBA рекомендуют везде, где это возможно, использовать Range, а не Selection. Основная причина в том, что использование Selection по сути дублирует действия на экране с помощью кода; Range позволяет вам действовать более непосредственно. Средство записи макросов Word обычно использует объект Selection; об этом следует помнить при преобразовании макросов Word в код VFP.

Манипулирование текстом

Свойство Text элементов Range и Selection содержит любой текст, находящийся в указанной области. Чтобы перенести содержимое документа в FoxPro, создайте соответствующий Range и прочитайте его свойство Text, например:

```
oRange = oWord.ActiveDocument.Paragraphs[7].Range  
cParagraph7 = oRange.Text
```

Свойство Text также позволяет добавлять или изменять содержимое документа. Вы можете добавить текст, назначив его свойству Text.

```
oRange.Text = "Это новое предложение."
```

Вы также можете добавить текст к тому, что уже есть. Простая манипуляция

текстом делает свое дело.

```
oRange.Text = oRange.Text + " Добавить текст в конце."
```

Или вы можете сделать это:

```
oRange.Text = "Добавить текст в начале. " + oRange.Text
```

Другая возможность – прочитать текст в VFP, каким-либо образом манипулировать им и записать его обратно:

```
cMakeUpper = oRange.Text
cMakeUpper = UPPER(cMakeUpper)
oRange.Text = cMakeUpper
```

Этот пример можно сократить до одной строки, например:

```
oRange.Text = UPPER(oRange.Text)
```

Используя этот подход, мы можем отправить данные из записи в новый пустой документ:

```
USE _SAMPLES + "TasTrade\Data\Customer"
```

```
LOCAL oDocument, oRange
oDocument = oWord.Documents.Add() && Использовать шаблон Normal
oRange = oDocument.Range()
```

```
oRange.Text = Customer_ID + ": " + Company_Name
oRange.Text = oRange.Text + "Attn: " + TRIM(Contact_Name) + ;
    " - " + Contact_Title
oRange.Text = oRange.Text + Address
oRange.Text = oRange.Text + TRIM(City) + " " + TRIM(Region) + ;
    Postal_Code
oRange.Text = oRange.Text + UPPER(Country)
```

Поскольку Word всегда сохраняет маркер абзаца (CHR(13)) в конце документа, он добавляет маркер абзаца после каждого добавления к oRange.Text при выполнении этого кода. В других ситуациях (включая приведенные ниже примеры) маркер абзаца необходимо добавить явно. Новый документ выглядит так, как показано на рисунке 2.

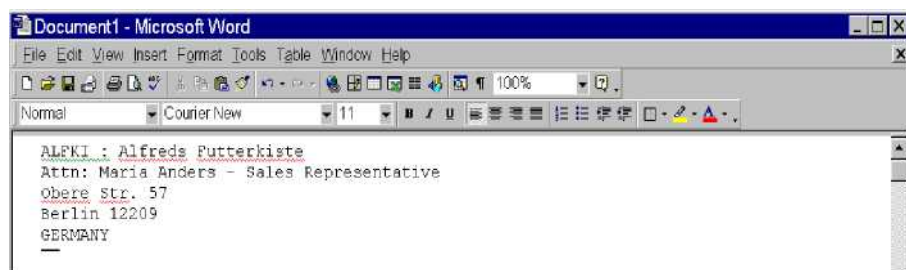


Рисунок 2. Есть несколько способов снять шкуру с кошки. Показанный обратный адрес можно отправить в Word разными способами. В этом случае, без специального форматирования, самый быстрый подход – создать всю строку в VFP, а затем отправить ее в Word.

Конечно, создание документа путем составления одной строки не использует особых возможностей Word. Методы Range InsertAfter и InsertBefore позволяют добавлять текст в конец или начало соответственно, а также расширять диапазон для включения нового текста.

Вот альтернативный, более быстрый подход к созданию документа, показанного на рисунке 2:

```
#DEFINE CR CHR(13)
```

```
USE _SAMPLES + "TasTrade\Data\Customer"
```

```
LOCAL oDocument, oRange
```

```
oDocument = oWord.Documents.Add() && Использовать шаблон Normal
```

```
oRange = oDocument.Range()
```

```
oRange.InsertAfter(Customer_ID + ": " + Company_Name + CR)
```

```
oRange.InsertAfter("Attn: " + TRIM(Contact_Name) + ;
```

```
" - " + Contact_Title + CR)
```

```
oRange.InsertAfter(Address + CR)
```

```
oRange.InsertAfter(TRIM(City) + " " + TRIM(Region) + Postal_Code + CR)
```

```
oRange.InsertAfter(UPPER(Country))
```

В наших тестах версия InsertAfter была в полтора-два раза быстрее метода конкатенации.

Перемещение в диапазоне или выделенном фрагменте

Помимо изменения содержимого диапазона или выделения вам может потребоваться изменить его размер. Ряд методов изменяют область, охватываемую диапазоном или выделением. Одним из самых простых является метод Move, который изменяет границы диапазона или выделения.

Move принимает два параметра. Первый указывает единицу перемещения, которую вы можете перемещать по символам, словам, абзацам, строкам в таблице или по всему документу. Второй параметр сообщает, сколько из указанных единиц необходимо переместить. Положительное число указывает на движение вперед (к концу документа), а отрицательное число означает движение назад (к началу документа).

Во всех случаях диапазон или выделенный фрагмент перед перемещением уменьшаются (или «сворачиваются», в терминах Word VBA) до одной точки. Хотя свертывание диапазона или выделения звучит ужасно, это не так. Текст, содержащийся в диапазоне/выделении, остается в документе, изменяется только размер диапазона или выделения. При движении вперед диапазон или выделение сокращается до конечной точки, а затем перемещается; при движении назад он уменьшается до исходной точки перед перемещением. После этого вам не нужно делать ничего особенного. Для программиста Автоматизации ключевой вопрос - понять, где в диапазоне начинается движение.

Константы из группы wdUnits используются для единиц движения. В таблице 1 показаны значения этой группы, которые можно передать методу Move.

Таблица 1. Словесные единицы. Константы в группе wdUnits представляют части документа.

Константа	Значение	Описание
wdCharacter	1	Один символ.
wdWord	2	Одно слово.
wdSentence	3	Одно предложение.

wdParagraph	4	Один абзац.
wdSection	8	Один раздел документа. (Word позволяет разделить документы на несколько разделов с разным форматированием.)
wdStory	6	Полная длина любой части документа, в которой вы находитесь. Word считает основную часть документа одной «историей», заголовок другой «историей», сноски - третьей и т. д.
wdCell	12	Одна ячейка таблицы.
wdColumn	9	Один столбец таблицы.
wdRow	10	Одна строка таблицы.
wdTable	15	Все пространство таблицы.

Чтобы создать диапазон в конце документа, вы можете использовать следующее:

```
oRange = oWord.ActiveDocument.Range()
oRange.Move( wdStory, 1)
```

Вот еще один способ создания документа, показанного на рисунке 2. Он использует метод Move для перемещения объекта Range.

```
#DEFINE CR          CHR(13)
#DEFINE wdStory 6
```

```
USE _SAMPLES + "TasTrade\Data\Customer"
```

```
LOCAL oDocument, oRange
oDocument = oWord.Documents.Add() && Использовать шаблон Normal
```

```
oRange = oDocument.Range()
```

```
oRange.Text = Customer_ID + ": " + Company_Name + CR
oRange.Move(wdStory)
oRange.Text = "Attn: " + TRIM(Contact_Name) + " - " + ;
    Contact_Title + CR
oRange.Move(wdStory)
oRange.Text = Address + CR
oRange.Move(wdStory)
oRange.Text = TRIM(City) + " " + TRIM(Region) + Postal_Code + CR
oRange.Move(wdStory)
oRange.Text = UPPER(Country)
```

По скорости эта версия занимает место между версиями конкатенации и вставки.

Метод Collapse позволяет явно уменьшить диапазон или выделение до одной точки. Он принимает один параметр, указывающий направление коллапса. Передача константы wdCollapseEnd (со значением 0) сворачивает диапазон или выделение до конечной точки (точки, ближайшей к концу документа). Передача wdCollapseStart (значение которого равно 1) уменьшает диапазон или выделение до начальной точки. (То есть функция «Collapse» перемещает либо начальную точку диапазона в конечную точку, либо конечную точку в начальную точку. Тогда диапазон состоит

только из одной точки, которая ранее была концом или началом диапазона.)

Пример можно переписать еще раз, чтобы использовать Collapse для управления диапазоном:

```
#DEFINE CR CHR(13)
#DEFINE wdCollapseEnd 0

USE _SAMPLES + "TasTrade\Data\Customer"

LOCAL oDocument, oRange
oDocument = oWord.Documents.Add() && Использовать шаблон Normal
oRange = oDocument.Range()

oRange.Text = Customer_ID + ": " + Company_Name + CR
oRange.Collapse(wdCollapseEnd)
oRange.Text = "Attn: " + TRIM(Contact_Name) + " - " + ;
    Contact_Title + CR
oRange.Collapse(wdCollapseEnd)
oRange.Text = Address + CR
oRange.Collapse(wdCollapseEnd)
oRange.Text = TRIM(City) + " " + TRIM(Region) + Postal_Code + CR
oRange.Collapse(wdCollapseEnd)
oRange.Text = UPPER(Country)
```

С точки зрения времени эта версия работает примерно так же, как версия Move.

Ряд других методов позволяют более точно настроить движение. К ним относятся MoveEnd, MoveStart, MoveLeft, MoveRight, EndOf и StartOf. Некоторые методы применимы только к выборке, а не к диапазонам.

Наконец, стоит отметить, что для этой конкретной задачи самый быстрый подход – объединить все строки в VFP, а затем отправить одну строку в документ:

```
USE _SAMPLES + "TasTrade\Data\Customer"

LOCAL oDocument, oRange
oDocument = oWord.Documents.Add() && Использовать шаблон Normal
oRange = oDocument.Range()
LOCAL cText
cText = ""

cText = Customer_ID + ": " + Company_Name + CR
cText = cText + "Attn: " + TRIM(Contact_Name) + " - " + ;
    Contact_Title + CR
cText = cText + Address + CR
cText = cText + TRIM(City) + " " + TRIM(Region) + Postal_Code + CR
cText = cText + UPPER(Country) + CR

oRange.Text = ""
oRange.InsertAfter(cText)
```

Учитывая скорость VFP при построении строк, эта версия занимает всего от одной трети до одной четверти времени, чем другие подходы. Хотя «Свернуть» и «Переместить» – не лучший подход в этом простом случае, они являются важными

методами работы с Word.

Закладки

Закладка – это способ назвать диапазон или местоположение. Word поддерживает коллекцию закладок, которые, естественно, называются «Bookmarks». Чтобы создать закладку, вызовите метод Add коллекции Bookmarks, передав имя новой закладки и диапазон, на который она ссылается. Например, этот код создает закладку «Заголовок» для первого абзаца документа:

```
oDocument.Bookmarks.Add("Заголовок", oDocument.Paragraphs[1].Range)
```

Зачем использовать закладки? Потому что Word выполняет работу по их обслуживанию. Вместо того, чтобы отслеживать переменную в VFP, мы можем просто попросить Word сохранить за нас диапазон. Что еще более важно, Word может запоминать диапазон между сеансами, поэтому, когда мы возвращаемся к документу, закладка все еще доступна и указывает на то же место.

```
? oDocument.Bookmarks.Item(1).Range.Text
```

```
? oDocument.Bookmarks.Item("Заголовок").Range.text
```

4.5 Форматирование

Если бы все, что мы могли сделать, это отправить текст в Word и прочитать уже существующий текст, Автоматизация была бы полезна, но не стоила бы слишком больших хлопот. Однако Автоматизация Word – это гораздо больше, чем просто отправка и получение текста. Одним из больших преимуществ использования Word по сравнению с VFP является возможность применять к документам сложное форматирование.

Word позволяет форматировать документы разными способами, а объекты, доступные для форматирования, отражают способ структурирования команд Word. Например, объект Font содержит свойства для настроек, найденных в диалоговом окне «Шрифт» Word (меню «Формат|Шрифт»). Объект ParagraphFormat управляет настройками диалогового окна «Абзац», такими как отступ, интервал и выравнивание. Аналогично, параметры диалогового окна «Параметры страницы» контролируются объектом PageSetup. Объекты Style представляют предварительно отформатированные и определяемые пользователем стили, доступные в документе. Эти четыре объекта управляют большинством часто используемых настроек. Другие объекты управляют другими аспектами форматирования.

Настройка страниц

Диалоговое окно «Параметры страницы» в меню «Файл» Word сообщает Word, какой тип бумаги ожидать, где ее взять и как расположить документ на этой бумаге, как только он ее найдет. В диалоговом окне элементы управления указывают источник бумаги (лоток для бумаги по умолчанию, альтернативный лоток, ручная подача и т. д.), тип бумаги (letter, legal, A4 и т. д.) и ориентацию страницы (книжная или альбомная). Диалоговое окно также используется для определения размера полей, верхних и нижних колонтитулов, чтобы указать, имеет ли первая страница документа верхний и нижний колонтитул, отличный от остальных, и имеют ли нечетные и четные страницы разные верхние и нижние колонтитулы. На самом деле там тоже гораздо больше вариантов. Повторяя нашу тему, Word обладает удивительной мощностью. На рисунке 3 показана страница «Поля» диалогового окна «Параметры страницы».

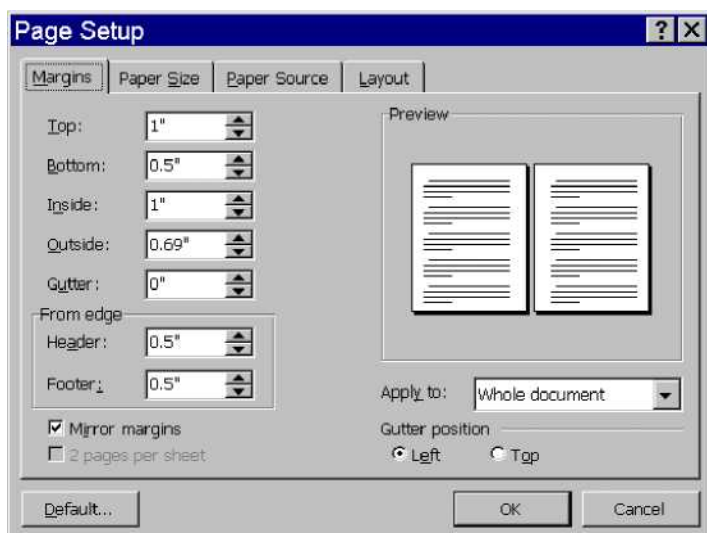


Рисунок 3. Диалоговое окно «Параметры страницы». В этом диалоговом окне можно указать поля, размер и ориентацию бумаги, одинаковые или разные верхние и нижние колонтитулы на первой странице, а также на нечетных и четных страницах и многое другое.

Объект PageSetup – это объект Автоматизации, который работает параллельно с диалогом. Он имеет набор свойств, которые обрабатывают различные доступные параметры. В отличие от многих объектов Word, большинство свойств PageSetup не ссылаются на другие объекты. Это просто ценности, поэтому работать с ними немного проще, чем со многими другими.

(Примечание: хотя в других приложениях Office есть объекты с именем PageSetup, каждый объект приложения Office уникален; их нельзя использовать как взаимозаменяемые.)

Таблица 2 показывает наиболее распространенные свойства объекта PageSetup, а также постоянные значения для часто используемых параметров.

Таблица 2. Свойства PageSetup. Объект PageSetup имитирует диалоговое окно «Параметры страницы» в меню «Файл» Word.

Свойство	Тип	Описание
PaperSize	Numeric	Тип бумаги, используемой для документа, например Legal, Letter, A4 и т. д. wdPaperLetter 2 wdPaperLegal 4 wdPaperA4 7 wdPaper11x17 1
FirstPageTray	Numeric	Источник первого листа бумаги документа. Первый лист обрабатывается отдельно, например, чтобы можно было использовать фирменный бланк для первой страницы письма, а для остальных – обычную бумагу wdPrinterDefaultBin 0 wdPrinterUpperBin 1 wdPrinterLowerBin 2

		wdPrinterManualFeed 4 wdPrinterEnvelopeFeed 5 wdPrinterAutomaticSheetFeed 7
OtherPagesTray	Numeric	Источник листов бумаги отличный от первого. Использует те же константы, что и FirstPageTray.
Orientation	Numeric	Ориентация бумаги. wdOrientPortrait 0 wdOrientLandscape 1
TopMargin, BottomMargin	Numeric	Вертикальные поля страницы в пунктах.
LeftMargin, RightMargin	Numeric	Горизонтальные поля страницы в пунктах.
VerticalAlignment	Numeric	Вертикальное положение текста на странице. wdAlignVerticalTop 0 wdAlignVerticalCenter 1 wdAlignVerticalJustify 2 wdAlignVerticalBottom 3
DifferentFirstPageHeaderFooter	Logical or Numeric	Указывает, отличаются ли верхние и нижние колонтитулы первой страницы документа от остальной части документа. (Числовое, только если не определено.)
OddAndEvenPagesHeaderFooter	Logical or Numeric	Указывает, имеют ли нечетные и четные страницы разные верхние и нижние колонтитулы. (Числовое, только если не определено.)

Как и многие другие свойства, связанные с измерениями, настройки полей предполагают пункты, а не то, что показано в диалоговом окне «Параметры страницы». Чтобы упростить задание измерений, объект приложения Word имеет ряд методов преобразования, включая InchesToPoints, CentimetersToPoints, PointsToInches, PointsToCentimeters и многие другие.



В следующем примере создается документ, центрируется по вертикали на странице, присваиваются разные нечетные и четные верхние и нижние колонтитулы, а также устанавливаются поля в один дюйм везде, кроме нижней части, где используется 0,75 дюйма.

* Настройка документа с настраиваемыми полями, разными нечетными и четными верхними и нижними колонтитулами и выравниванием по центру

```
#DEFINE wdAlignVerticalCenter 1
```



```
oDocument = oWord.Documents.Add()

WITH oDocument.PageSetup
    .VerticalAlignment = wdAlignVerticalCenter
    .OddAndEvenPagesHeaderFooter = .T.

    * Установите нижнее поле 0,75 и 1 дюйм,
    * верхнее, левое и правое поля.
    LOCAL nThreeQuartersInPoints, nInchInPoints
    nThreeQuartersInPoints = oWord.InchesToPoints( .75 )
    nInchInPoints          = oWord.InchesToPoints( 1 )

    .BottomMargin = nThreeQuartersInPoints
    .TopMargin    = nInchInPoints
    .LeftMargin   = nInchInPoints
    .RightMargin  = nInchInPoints
ENDWITH
```

Поскольку количество пунктов в дюймах (или сантиметрах) никогда не меняется, вы можете выполнить эти преобразования самостоятельно. Вы можете определить свои собственные константы и выполнять арифметические действия в собственном коде VFP, а не вызывать для этого методы Автоматизации. Вот альтернативный код для части настройки поля из предыдущего примера:

```
* АЛЬТЕРНАТИВНЫЙ МЕТОД: заменяет семь строк кода выше:
#DEFINE autoInchesToPoints 72

.BottomMargin = 0.75 * autoInchesToPoints
.TopMargin    = 1.00 * autoInchesToPoints
.LeftMargin   = 1.00 * autoInchesToPoints
.RightMargin  = 1.00 * autoInchesToPoints
```

Наши тесты показывают увеличение скорости примерно на два порядка (это примерно в 100 раз быстрее) при самостоятельном выполнении преобразований. Для случайных вычислений это не имеет большого значения, но если вам нужно преобразовать пункты в другие единицы сотни или тысячи раз, определенно стоит использовать собственный код VFP и не забывать везде включать правильные константы.

Объект PageSetup имеет ряд других, более непонятных свойств, не показанных в таблице 2. Хотя описания некоторых свойств, показанных в таблице 2, относятся к документу в целом, каждый раздел документа может иметь собственный объект PageSetup, свойства которого применяются только в этом разделе.

PageSetup имеет только два метода. TogglePortrait переключает ориентацию между книжной и альбомной ориентацией. Это эквивалентно установке для свойства Orientation значения, противоположного его текущему значению. SetAsDefaultTemplate делает текущий PageSetup параметром по умолчанию для этого документа и для всех новых документов на основе активного шаблона. Это похоже на нажатие кнопки «По умолчанию» в диалоговом окне «Параметры страницы». (Осторожно! Название этого метода подразумевает, что он создает шаблон по умолчанию. Это не так. Он создает настройки по умолчанию для текущего шаблона.)

Настройка шрифтов

Шрифты в Word управляются диалоговым окном «Шрифт» в меню «Формат» (показано на рисунке 4). В этом диалоговом окне можно настроить имя шрифта, его размер, стиль (например, жирный, курсив, подчеркивание и т. д.), цвет и эффекты (например, зачеркивание, надстрочные и подстрочные индексы и многое другое). Оно также управляет более экзотическими параметрами, такими как кернинг, анимация текста или вокруг него, вертикальное положение текста относительно базовой линии, расстояние между символами и многое другое. Этими параметрами управляет объект Font, который похож (хотя и не совсем идентичен) объектам Font на других серверах Office.

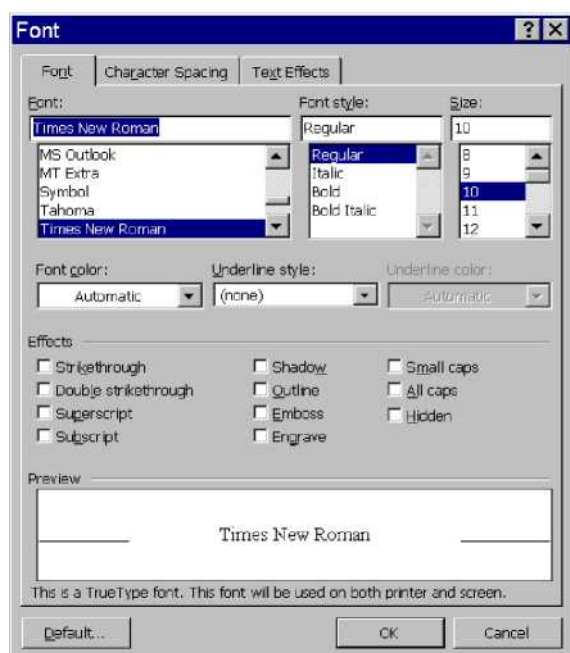


Рисунок 4. Указание шрифтов. Диалоговое окно «Шрифт» управляет шрифтом, размером, стилем и цветом, а также такими необычными параметрами, как кернинг и расстояние между символами. В Автоматизации объект Font управляет всеми этими функциями.

Range, Selection и Style (обсуждаемые в разделе «Работа со стилями» далее в этой главе), а также многие другие объекты имеют свойство Font, указывающее на объект Font. Изменение свойств объекта Font изменяет шрифт этой части документа. Например, чтобы изменить всю информацию о клиенте, показанную на рисунке 2, на Arial с 12 пунктами, вы можете использовать этот код:

```
oRange = oDocument.Range()
oRange.Font.Name = "Arial"
oRange.Font.Size = 12
```

Чтобы упростить задачу, просто установите нужный шрифт перед отправкой текста в документ. Вот еще одна версия программы для отправки адреса клиента. Здесь с самого начала используется 12-пунктовый Arial:

```
#DEFINE CR CHR(13)
```

```
USE _SAMPLES + "TasTrade\Data\Customer"
```

```
LOCAL oDocument, oRange
oDocument = oWord.Documents.Add() && Использовать шаблон Normal
oRange = oDocument.Range()
```

```
oRange.Font.Name = "Arial"
oRange.Font.Size = 12
```

```
LOCAL cText
cText = ""
```

```
cText = Customer_ID + ": " + Company_Name + CR
cText = cText + "Attn: " + TRIM(Contact_Name) + " - " +
Contact_Title + CR
cText = cText + Address + CR
cText = cText + TRIM(City) + " " + TRIM(Region) + Postal_Code + CR
cText = cText + UPPER(Country) + CR
```

```
oRange.Text = ""
oRange.InsertAfter(cText)
```

На самом деле это не лучший способ установить шрифт для всего документа. Лучше использовать шаблон, в котором установлен необходимый шрифт стиля Обычный. (Дополнительную информацию см. в разделе «Работа со стилями» далее в этой главе.)

Таблица 3 перечисляет свойства шрифта, с которыми вы, вероятно, захотите работать, а также константы Word для них, где это необходимо.

Таблица 3. Свойства шрифта. Объект Font управляет внешним видом шрифта: от его начертания до его размера, стиля и многого другого. В этой таблице показаны наиболее распространенные свойства. Проверьте справку, чтобы узнать о более необычных настройках.

Свойство	Тип	Описание
Name	Character	Название шрифта.
Size	Numeric	Размер шрифта, в пунктах.
Bold	Numeric or Logical	Указывает, выделен ли текст жирным шрифтом.
Italic	Numeric or Logical	Указывает, выделен ли текст курсивом.
Underline	Numeric	Тип подчеркивания. wdUnderlineNone 0 wdUnderlineDouble 3 wdUnderlineSingle 1 wdUnderlineDotted 4 wdUnderlineWords 2 wdUnderlineThick 6
Superscript, Subscript	Numeric or Logical	Указывает, является ли текст надстрочным или подстрочным индексом.

Текст в диапазоне (или в любой другой области, которую покрывает объект Font) может иметь более одной настройки шрифта. Когда это происходит, различные числовые свойства получают значение wdUndefine (9999999). (Именно поэтому свойства, которые, как вы ожидаете, будут логическими, перечислены как числовые или логические.) Font.Name в этой ситуации – пустая строка.

Следующий пример демонстрирует связанную с этим сложность работы с объектами VBA в Visual FoxPro. Хотя эти логические свойства (например, жирный шрифт и курсив) можно установить, назначив логические значения VFP .T. и .F., их нельзя сравнивать с логическими значениями. Подобный код завершается с ошибкой «Несоответствие типа оператора/операнда»:

```
IF oFont.Bold
```

Это связано с двойными числовыми и логическими возможностями Bold. Когда вы присваиваете логические значения, Word где-то их переводит, но для сравнения вам придется использовать числовые значения. В следующем примере (SetUserFont.PRG) вместо использования логических значений VFP определяются константы TRUE и FALSE. Таким образом, код становится читабельным, но позволяет избежать проблемы несоответствия типов.

Вы можете разрешить пользователю выбирать шрифт, сначала вызвав функцию GetFont() VFP. Здесь представлена функция, которая позволяет пользователю указать шрифт, запрашивая шрифт, используемый в данный момент, и заменяет его на указанный шрифт. Запустите этот код, используя следующий синтаксис, убедившись, что существует активный документ с указанным диапазоном (например, запустите любой из примеров кода в предыдущих разделах).

```
SetUserFont(oRange.Font)
```



Листинг 1 показывает код SetUserFont (он также включен в файлы загрузки для разработчиков, доступные по адресу www.hentzenwerke.com).

Листинг 1. Эта функция позволяет пользователю выбрать шрифт, запрашивая имя, размер и стиль объекта шрифта, который он получает в качестве параметра. *

```
* SetUserFont.PRG
* © 2000, Tamar E. Granor and Della Martin
* From: Microsoft Office Automation with Visual FoxPro
* Hentzenwerke Publishing. www.hentzenwerke.com
```

```
* Позвольте пользователю указать шрифт, затем установите
* переданный объект шрифта, чтобы использовать его.
```

```
#DEFINE TRUE -1
#DEFINE FALSE 0
```

```
LPARAMETERS oFont
* oFont = Ссылка на объект шрифта
```

```
LOCAL cName, nSize, cStyle
LOCAL cFontString, aFontInfo[3]
```

```
* Получили ли мы объект шрифта для работы?
IF VarType(oFont) <> "0"
    RETURN .F.
ENDIF
```

```
* Получить текущие настройки объекта шрифта.
WITH oFont
    cName = .Name
    nSize = .Size
```

```
cStyle = ""
IF .Bold = TRUE    && Не используйте VFP .T. здесь
    cStyle = cStyle + "B"
ENDIF
IF .Italic = TRUE  && или здесь
    cStyle = cStyle + "I"
ENDIF
ENDWITH

* Запросите у пользователя шрифт
cFontString = GetFont(cName, nSize, cStyle)

IF EMPTY(cFontString)
    * Пользователь отменил ввод
    RETURN .F.
ELSE
    * Разобрать выбранное на компоненты
    cFontString = CHRTRAN(cFontString, ",", CHR(13))
    ALINES(aFontInfo,cFontString)

    * Примените их к объекту шрифта
    WITH oFont
        .Name = aFontInfo[1]
        .Size = VAL(aFontInfo[2])
        IF "B"$aFontInfo[3]
            .Bold = .T.    && .T. здесь работает
        ENDIF
        IF "I"$aFontInfo[3]
            .Italic = .T.
        ENDIF
    ENDWITH
ENDIF
RETURN .T.
```

Форматирование абзацев

Абзацы – ключевое понятие в Word. Большую часть форматирования Word можно рассматривать как хранящуюся «в» маркере, который следует за каждым абзацем. Вот почему при перемещении текста иногда меняется его форматирование. Если вам не удастся взять с собой маркер абзаца, перемещенный текст примет форматирование нового места.

На уровне абзаца вы можете определить выравнивание текста (по левому, правому краю, по центру или полное выравнивание), различные виды отступов (как количество, так и тип), интервалы между строками и абзацами, обработку висячих строк и многое другое. Word допускает регулярные отступы как от левого, так и от правого поля, а также отступы первой строки и выступающие отступы. В интерактивном режиме всем этим управляет диалоговое окно «Абзац» в меню «Формат» (показано на рисунке 5). За кулисами этими параметрами управляет объект ParagraphFormat.

Range, Selection и Style, среди прочего, имеют объект ParagraphFormat, доступ к которому осуществляется через свойство ParagraphFormat. Объект Paragraph имеет свойство Format, которое обращается к объекту ParagraphFormat.

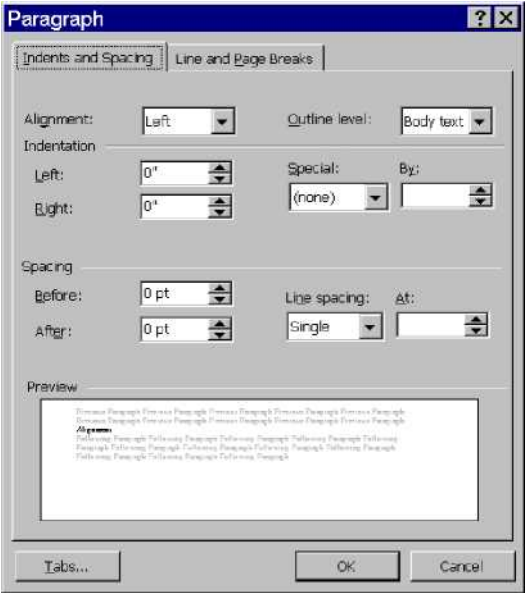


Рисунок 5. Диалог «Абзац». Показанная здесь страница «Отступы и интервалы» позволяет указать выравнивание, отступ и интервал. Страница «Разрывы строк и страниц» управляет висячими и потерянными строками, а также автоматически расставляет переносы. Объект ParagraphFormat управляет этими параметрами Автоматизации.

Таблица 4 показывает некоторые часто используемые свойства объекта ParagraphFormat и часто используемые для них константы. Как и многие объекты Word, ParagraphFormat имеет лишь несколько методов; ни один из них вряд ли будет полезен в большинстве кодов Автоматизации.

В этом примере для первого абзаца в текущем диапазоне задается отступ первой строки 0,5 дюйма, контроль висячих строк, двойной интервал и полное выравнивание:

```
#DEFINE wdAlignParagraphJustify 3
#DEFINE wdLineSpaceDouble 2

WITH oRange.Paragraphs[1].Format
    .FirstLineIndent = oWord.InchesToPoints( .5 )
    .WidowControl = .T.
    .Alignment = wdAlignParagraphJustify
    .LineSpacingRule = wdLineSpaceDouble
ENDWITH
```

В этом примере в диапазоне задается тройной межстрочный интервал:

```
#DEFINE wdLineSpaceMultiple 5

WITH oRange.ParagraphFormat
    .LineSpacingRule = wdLineSpaceMultiple
    .LineSpacing = oWord.LinesToPoints( 3 )
ENDWITH
```

Таблица 4. Свойства формата абзаца.

Свойство	Тип	Описание

Alignment	Numeric	Выравнивание текста в абзаце. wdAlignParagraphLeft 0 wdAlignParagraphCenter 1 wdAlignParagraphRight 2 wdAlignParagraphJustify 3
LeftIndent	Numeric	Отступ левого края данного абзаца от левого поля, в пунктах.
RightIndent	Numeric	Отступ правого края данного абзаца от правого поля, в пунктах.
FirstLineIndent	Numeric	Отступ первой строки абзаца. Это свойство определяет, будет ли в абзаце первая строка с отступом или выступом (обеспечивая висячий отступ). Установите положительное значение для отступа первой строки, 0, чтобы первая строка оставалась на одном уровне с остальной частью абзаца, или отрицательное значение для выступающего отступа. Обратите внимание, что при висячем отступе первая строка не сдвигается влево; последующие строки перемещаются вправо.
SpaceBefore, SpaceAfter	Numeric	Количество пробелов (так называемых <i>ведущих</i>) до и после абзаца в пунктах.
LineSpacingRule	Numeric	Действующий тип межстрочного интервала. Этот параметр может полностью определять межстрочный интервал или задавать основу для свойства LineSpacing. wdLineSpaceSingle 0 wdLineSpaceAtLeast 3 wdLineSpaceDouble 2 wdLineSpace1pt5 1 wdLineSpaceExactly 4 wdLineSpaceMultiple 5
LineSpacing	Numeric	Фактический межстрочный интервал в пунктах, если LineSpacingRule имеет значение wdLineSpaceAtLeast, wdLineSpaceExactly или wdLineSpaceMultiple.
WidowControl	Numeric or Logical	Указывает, сохраняются ли первая и последняя строки абзаца на той же странице, что и остальная часть абзаца.
KeepTogether	Numeric or Logical	Указывает, будет ли весь абзац храниться на одной странице.
KeepWithNext	Numeric or Logical	Указывает, находится ли абзац на той же странице, что и следующий за ним абзац.
Hyphenation	Numeric or Logical	Указывает, расставляются ли в абзаце переносы автоматически.

Работа со стилями

Хотя здесь и там уместно вручную настроить форматирование слова, предложения или абзаца, наиболее эффективный способ использования Word — это воспользоваться стилями, которые представляют собой именованные форматы, которые можно применить к части документа. Когда вы работаете в Word, вы можете увидеть стиль точки вставки в первом раскрывающемся списке на панели инструментов «Форматирование».

В Word есть два типа стилей: стили абзацев и стили символов. Стили символов используются для фрагментов и управляют лишь некоторыми настройками, в основном связанными со шрифтами. Стили абзацев, как следует из названия, применяются ко всем абзацам и включают гораздо больше опций. В стилях абзацев можно указать шрифт и форматирование абзацев, а также настройки вкладок и многое другое. В диалоговом окне «Стиль», доступном в меню «Формат» Word (см. рис. 6), стилям абзаца предшествует маркер абзаца, а стилям символов — подчеркнутая буква «а».

Использование стилей во многом похоже на использование классов в объектно-ориентированном языке. Они упрощают обеспечение единообразия во всех документах и позволяют изменять характеристики разделов текста одним изменением. Стили Word предлагают и другие преимущества. Например, каждый стиль абзаца устанавливает стиль по умолчанию для следующего абзаца. Немного поработав заранее, стили также можно использовать для создания структуры документа.

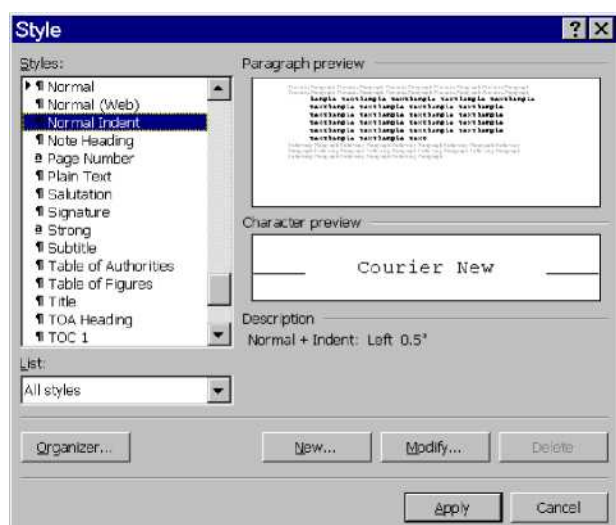


Рисунок 6. Создание и выбор стилей. В диалоговом окне «Стиль» Word перечислены доступные стили. Стили абзацев предваряются символом абзаца, а подчеркнутая буква «а» предшествует стилям символов. Обратите внимание, что в описании стиль описывается в терминах другого «базового стиля».

Объект Document включает коллекцию Styles, содержащую по одному объекту Style для каждого стиля, хранящегося в документе. Вы можете добавить свои собственные стили, используя метод Add коллекции Styles. Свойства стиля различных объектов указывают на объекты стиля.

Вся эта возня со стилями означает, что вместо написания большого количества кода для изменения шрифтов и размеров, установки выравнивания, интерлиньяжа и других подобных вещей, вы можете просто определить несколько пользовательских стилей или изменить встроенные стили, а затем примените их к своим документам по мере необходимости. Например, этот код изменяет стиль «Обычный», который всегда доступен, на использование курсива Garamond с центрированием и

размером в 16 пунктов:

```
#DEFINE wdStyleNormal -1
#DEFINE wdAlignParagraphCenter 1

WITH oWord.ActiveDocument.Styles[ wdStyleNormal ]
    WITH .Font
        .Name = "Garamond"
        .Size = 16
        .Italic = .T.
    ENDWITH
    .ParagraphFormat.Alignment = wdAlignParagraphCenter
ENDWITH
```

Чтобы применить существующий стиль к части документа, задайте для свойства Style диапазона или абзаца встроенный стиль с использованием константы или имя пользовательского стиля. В таблице 5 перечислены константы для некоторых наиболее часто используемых встроенных стилей. В этом примере стиль «Заголовок 1» применяется к диапазону, на который указывает oRange:

```
#DEFINE wdStyleHeading1 -2
oRange.Style = oWord.ActiveDocument.Styles[ wdStyleHeading1 ]
```

Таблица 5. Встроенные стили. Word имеет более 100 встроенных стилей, на каждый из которых ссылается определенная константа. В этой таблице показаны лишь некоторые из наиболее распространенных; используйте Обзорщик объектов, чтобы найти остальное.

Константа	Значение	Константа	Значение
wdStyleNormal	-1	wdStyleHeading1	-2
wdStyleBodyText	-67	wdStyleHeading2	-3
wdStyleDefaultParagraphFont	-66	wdStyleHeading3	-4

Создание пользовательских стилей

Помимо изменения встроенных стилей, вы можете создавать свои собственные стили. Чтобы добавить новый стиль, используйте метод Add коллекции Styles. Add принимает два параметра: имя нового стиля и константу Word, которая указывает, является ли это стилем абзаца (wdStyleTypeParagraph = 1) или стилем символов (wdStyleTypeCharacter = 2).

Каждый стиль основан на существующем стиле. По умолчанию новые стили абзацев основаны на стиле «Обычный», а новые стили символов – на стиле «Шрифт символов по умолчанию». Однако свойство BaseStyle указывает, от какого стиля наследуется другой стиль.

На какой бы стиль ни указывал BaseStyle, все остальные изменения свойств стиля используют BaseStyle в качестве точки отсчета. Если вы посмотрите на диалоговое окно «Стиль», то увидите, что характеристики стиля описаны как «<базовый стиль>» + «<различные другие характеристики>». Свойство «Описание» содержит ту же информацию. Таким образом, как и в случае с классами в ООП, изменения в базовом стиле приводят к изменению любых стилей, основанных на нем.

Таблица 6 перечисляет ключевые свойства объекта Style, а также важные константные значения.

Таблица 6. Стил ь имеет значение. Стили – это версия ООП для Word. Они предлагают способ обеспечить единообразное форматирование внутри и между документами.

Свойство	Тип	Описание
BaseStyle	Character, Numeric, or Object	Имя, постоянное значение или указатель на стиль, на котором основан этот стиль. См. Таблицу 5 и Справку или Обзорщик объектов для получения константных значений для встроенных стилей.
Type	Numeric	Тип абзаца или символа стиля. wdStyleTypeParagraph 1 wdStyleTypeCharacter 2
Builtin	Logical	Указывает, является ли это встроенным стилем.
Description	Character	Описание стиля (как показано в диалоговом окне «Стил ь»).
Font	Object	Указатель на объект Font для стиля.
ParagraphFormat	Object	Указатель на объект ParagraphFormat для стиля.
Borders	Object	Указатель на коллекцию Borders для этого стиля.
Shading	Object	Указатель на объект Shading для стиля.
NextParagraphStyle	Character, Numeric, or Object	Имя, постоянное значение или указатель на стиль абзаца, следующего за этим абзацем, для стилей абзаца.



Листинг 2 берет простой документ с адресом клиента, приведенный ранее в этой главе, и начинает создавать документ, достойный Word. Он создает несколько новых стилей для выполнения этой работы. На практике вы можете использовать для этого документа встроенные стили «Обычн ый» и «Заголовок X» (существует несколько уровней заголовков), переопределяя их по мере необходимости. Но пример показывает, насколько легко создавать новые стили. (Эту программу можно найти как Styles.PRG в файлах для загрузки для разработчиков, доступных на сайте www.hentzenwerke.com.)

Листинг 2. Эта программа создает несколько пользовательских стилей, а затем использует их для создания документа, показанного на рисунке 7.

```
* Styles.PRG
* © 2000, Tamar E. Granor and Della Martin
* From: Microsoft Office Automation with Visual FoxPro
* Hentzenwerke Publishing. www.hentzenwerke.com

* Создайте форматированный документ, отправив данные из одной
* записи.
* Демонстрирует объекты Style, но более вероятно, что потребности
* здесь могут быть удовлетворены существующими стилями.

#DEFINE CR CHR(13)
#DEFINE wdStyleTypeParagraph 1
#DEFINE wdStyleNormal -1
#DEFINE wdAlignParagraphLeft 0
#DEFINE wdAlignParagraphCenter 1
```

[illegible]

```

WITH oMinorHeadingStyle
    .BaseStyle = oBodyStyle
    .Font.Size = 16
ENDWITH

* Теперь создайте отчет о клиенте
* Во-первых, информация о нашей компании сосредоточена вверху
oRange.Style = oMajorHeadingStyle
oRange.InsertAfter("Automation Sample Company" + CR)
oRange.InsertAfter("Factory Blvd." + CR)
oRange.InsertAfter("Robotville, PA 19199" + CR)

* Теперь оставьте пустое место и поместите информацию об этом
клиенте
oRange.Collapse( wdCollapseEnd )
oRange.End = oRange.End + 1 && чтобы разрешить присвоение стиля
oRange.Style = oBodyStyle
oRange.InsertAfter(CR + CR)

* Используйте второстепенный заголовок для идентификатора и имени
клиента.
* Выделите идентификатор клиента жирным шрифтом
oRange.Collapse( wdCollapseEnd )
oRange.End = oRange.End + 1 && чтобы разрешить присвоение стиля
oRange.Style = oMinorHeadingStyle
oRange.InsertAfter(Customer_ID + ": " + TRIM(Company_Name) + CR)
oRange.Words[1].Font.Bold = .t.

* Обычный стиль тела для информации об адресе
oRange.Collapse( wdCollapseEnd )
oRange.End = oRange.End + 1 && чтобы разрешить присвоение стиля
oRange.Style = oBodyStyle
oRange.InsertAfter(TRIM(Contact_Title) + ":" + TRIM(Contact_Name);
                  + CR)
oRange.InsertAfter(TRIM(Address) + CR)
oRange.InsertAfter(TRIM(City) + " " + TRIM(Region) + ;
                  Postal_Code + CR)
oRange.InsertAfter(UPPER(TRIM(Country)) + CR )
* Дополнительная пустая строка
oRange.InsertAfter( CR )

* Вернуться к второстепенному заголовку для номера телефона
oRange.Collapse( wdCollapseEnd )
oRange.End = oRange.End + 1 && чтобы разрешить присвоение стиля
oRange.Style = oMinorHeadingStyle
oRange.InsertAfter( "Phone: " + TRIM(Phone) + CR)

* Номер факса в обычном стиле
oRange.Collapse( wdCollapseEnd )
oRange.End = oRange.End + 1 && чтобы разрешить присвоение стиля
oRange.Style = oBodyStyle
oRange.InsertAfter( "Fax: " + TRIM(Fax) + CR )

```

Обратите внимание на использование коллекции Words для выделения жирным шрифтом только идентификатора клиента, а не всей строки. На рисунке 7 показан получившийся документ в Word.

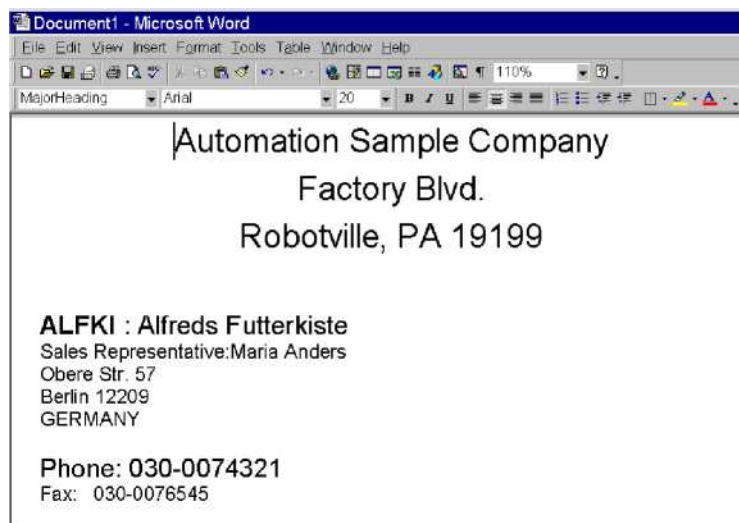


Рисунок 7. Использование стилей. Вместо независимого форматирования каждого элемента стили позволяют определять и называть наборы характеристик форматирования, а затем единообразно применять их внутри и между документами. Стили можно считать ООП для форматирования.

В таком маленьком примере может быть трудно увидеть смысл создания и использования стилей. Стоит отметить, что код в примере почти поровну разделен между форматированием стилей и помещением данных в документ. Для выполнения того же форматирования без использования стилей потребовалось бы гораздо больше строк. Кроме того, подумайте, что потребуется для добавления дополнительной информации для того же клиента. После того как стили определены, их можно использовать снова и снова. Благодаря 42 строкам кода у нас есть три стиля, которые можно применять там, где они нам нужны, независимо от того, сколько еще строк кода мы напишем для отправки текста в документ. В главе 5 «Промежуточный Word» мы рассмотрим создание шаблонов, в которых мы можем сохранять наши собственные стили для использования в нескольких документах.

Границы и штриховка

Word позволяет размещать границы и затенять различные части документа, чтобы отделить их от остальной части документа. Вы можете разместить рамки вокруг или заштриховать практически все: от одного символа до целой страницы. Стили границ варьируются от простых линий до сложных узоров и от половины до шести точек. Граница может быть применена к любой или всем сторонам текста. Затенение может быть любого цвета и может варьироваться от простого цвета до узорчатой заливки. В интерактивном режиме все это контролируется из диалогового окна «Границы и заливка» в меню «Формат».

В Автоматизации всем правят два объекта. Неудивительно, что они называются «Borders» и «Shading». Borders – это коллекция объектов Border.

Создание границ

Интерактивная установка границ может оказаться сложной задачей. Часто бывает трудно провести границу там, где вы действительно этого хотите. Это связано с тем, что коллекция Borders может принадлежать разным объектам, и объект, которому

она принадлежит, определяет, где на самом деле появляется граница. В Таблице 7 перечислены некоторые объекты, которые могут содержать коллекцию Borders, и интерпретация коллекции в этом случае.

Таблица 7. Почти все может иметь границы. Интерпретация и размер коллекции Borders зависят от того, какому объекту она принадлежит.

Объект	Значение Borders
Section	Устанавливает Page Border для этого раздела документа.
Paragraphs collection	Устанавливает границы вокруг группы абзацев в коллекции.
Paragraph	Устанавливает границы для этого абзаца.
Range	Устанавливает границы для этого диапазона. Когда диапазон пересекает границы абзаца или возникает какой-либо разрыв, например жесткий разрыв страницы или разрыв раздела внутри диапазона, диапазон расширяется и включает в себя полные абзацы на каждом конце. Граница применяется к этому расширенному диапазону, который обрабатывается как коллекция Paragraphs.
Table	Устанавливает границы внешней части таблицы и линий сетки внутри таблицы.
Rows collection	Устанавливает границы строк в коллекции. Применяется только к внешним границам коллекции, а не к границам каждой строки в коллекции.
Columns collection	Устанавливает границы столбцов в коллекции. Применяется только к внешним границам коллекции, а не к границам каждого столбца в коллекции.
Row	Устанавливает границы для указанной строки.
Column	Устанавливает границы для указанного столбца.
Cells collection	Устанавливает границы ячеек в указанной коллекции. Этот подход работает намного лучше, чем использование строк или столбцов. Обязательно укажите верхнюю или нижнюю границу при доступе к ячейкам через строку, а также левую или правую границу при доступе к ячейкам через столбец. Пока остальные работают, они не очень продуктивны.
Cell	Устанавливает границы для указанной ячейки, обозначенной как строкой, так и столбцом. Этот подход также эффективен.
Style	Устанавливает границы стиля.

Ряд других объектов также ссылаются на коллекции Borders, но они довольно загадочны.

Коллекция Borders содержит гораздо больше информации, чем большинство других. Кроме того, в отличие от большинства коллекций, вы не можете свободно добавлять участников. Количество содержащихся в нем объектов Border определяется типом объекта, которому он принадлежит. В большинстве случаев имеется четыре члена, доступ к которым осуществляется с помощью этих констант: wdBorderTop (-1), wdBorderLeft (-2), wdBorderBottom (-3) и wdBorderRight (-4). Для некоторых объектов, включая таблицы, коллекция имеет дополнительные члены.

Константы: wdBorderHorizontal (-5), wdBorderVertical (-6), wdBorderDiagonalDown (-7) и wdBorderDiagonalUp (-8).

Помимо отдельных объектов Border, коллекция имеет ряд других свойств. В таблице 8 представлена их выборка.

Таблица 8. Свойства коллекции границ. Хотя большинство коллекций имеют мало свойств, Borders особенно богат ими.

Свойство	Тип	Описание
Count	Numeric	Количество объектов Border в этой коллекции.
InsideLineStyle	Numeric	Тип линии, используемой внутри таблицы. Можно переопределить, установив границы для отдельных строк, столбцов или ячеек. Использует константы, показанные для LineStyle в таблице 9.
InsideLineWidth	Numeric	Размер строк, используемых внутри таблицы. Можно переопределить, установив границы для отдельных строк, столбцов или ячеек. Использует константы, показанные для LineWidth в таблице 9.
OutsideLineStyle	Numeric	Тип линии, используемой для внешней границы указанного объекта. Использует константы, показанные для LineStyle в таблице 9.
OutsideLineWidth	Numeric	Размер линий, используемых для внешней границы указанного объекта. Использует константы, показанные для LineWidth в таблице 9.
Enable	Logical	Указывает, следует ли включать границы для указанного объекта. Установка этого свойства в .F. отключает границы объекта.

Множество других свойств определяют, насколько далеко от указанного объекта появляются границы.

Наконец мы дошли до настоящего объекта Border, который описывает характеристики отдельной границы. В Таблице 9 показаны его ключевые свойства.



Этот код создает документ и размещает на нем рамку страницы в виде воздушных шаров. На рисунке 8 показан результат (существенно уменьшенный в размерах).

```
#DEFINE wdArtBalloonsHotAir 12
#DEFINE wdBorderTop -1
#DEFINE wdBorderBottom -2
#DEFINE wdBorderLeft -3
#DEFINE wdBorderRight -4
```

```
oDocument = oWord.Documents.Add()
WITH oDocument.Sections[1]
    .Borders[ wdBorderTop ].ArtStyle = wdArtBalloonsHotAir
    .Borders[ wdBorderBottom ].ArtStyle = wdArtBalloonsHotAir
    .Borders[ wdBorderLeft ].ArtStyle = wdArtBalloonsHotAir
    .Borders[ wdBorderRight ].ArtStyle = wdArtBalloonsHotAir
```

ENDWITH

Таблица 9. Что внутри границы? Эти свойства описывают одну границу.

Свойство	Тип	Описание																														
LineStyle	Numeric	Тип линии, используемой для границы. Например: <table><tr><td>wdLineStyleNone</td><td>0</td><td>wdLineStyleDashDot</td><td>5</td></tr><tr><td>wdLineStyleSingle</td><td>1</td><td>wdLineStyleDouble</td><td>7</td></tr><tr><td>wdLineStyleDot</td><td>2</td><td>wdLineStyleSingleWavy</td><td>18</td></tr></table>	wdLineStyleNone	0	wdLineStyleDashDot	5	wdLineStyleSingle	1	wdLineStyleDouble	7	wdLineStyleDot	2	wdLineStyleSingleWavy	18																		
wdLineStyleNone	0	wdLineStyleDashDot	5																													
wdLineStyleSingle	1	wdLineStyleDouble	7																													
wdLineStyleDot	2	wdLineStyleSingleWavy	18																													
LineWidth	Numeric	Размер линии, используемой для границы. Вы не можете установить размер в пунктах для линии, а должны использовать постоянное значение для заранее определенного размера. Проверьте обозреватель объектов, чтобы увидеть больше вариантов. <table><tr><td>wdLineWidth025pt</td><td>2</td><td>0.25 points</td></tr><tr><td>wdLineWidth050pt</td><td>4</td><td>0.5 points</td></tr><tr><td>wdLineWidth075pt</td><td>6</td><td>0.75 points</td></tr><tr><td>wdLineWidth100pt</td><td>8</td><td>1 point</td></tr><tr><td>wdLineWidth150pt</td><td>10</td><td>1.5 points</td></tr><tr><td>wdLineWidth225pt</td><td>18</td><td>2.25 points</td></tr><tr><td>wdLineWidth300pt</td><td>24</td><td>3 points</td></tr><tr><td>wdLineWidth450pt</td><td>36</td><td>4.5 points</td></tr><tr><td>wdLineWidth600pt</td><td>48</td><td>6 points</td></tr><tr><td>wdUndefined</td><td>9999999</td><td>не определено</td></tr></table>	wdLineWidth025pt	2	0.25 points	wdLineWidth050pt	4	0.5 points	wdLineWidth075pt	6	0.75 points	wdLineWidth100pt	8	1 point	wdLineWidth150pt	10	1.5 points	wdLineWidth225pt	18	2.25 points	wdLineWidth300pt	24	3 points	wdLineWidth450pt	36	4.5 points	wdLineWidth600pt	48	6 points	wdUndefined	9999999	не определено
wdLineWidth025pt	2	0.25 points																														
wdLineWidth050pt	4	0.5 points																														
wdLineWidth075pt	6	0.75 points																														
wdLineWidth100pt	8	1 point																														
wdLineWidth150pt	10	1.5 points																														
wdLineWidth225pt	18	2.25 points																														
wdLineWidth300pt	24	3 points																														
wdLineWidth450pt	36	4.5 points																														
wdLineWidth600pt	48	6 points																														
wdUndefined	9999999	не определено																														
ColorIndex, Color	Numeric	Цвет линии, используемой для границы. Word предлагает два набора цветов. Используйте ColorIndex, чтобы указать значение из короткого списка, используя константы wdColorIndex, или используйте Color, чтобы получить полный список значений RGB, который можно передать либо с помощью функции RGB() VFP, как фактическое числовое значение, либо с помощью констант wdColor. Любое свойство является приемлемым. ColorIndex – это быстро и просто, но Color дает вам больше возможностей. Обратите внимание, что Color – это новая функция в Word 2000.																														
ArtStyle	Numeric	Изображение, используемое в качестве границы страницы. Доступно более 150. Оттенки PrintShop. Вот образец: <table><tr><td>wdArtApples</td><td>1</td><td>wdArtMusicNotes</td><td>79</td></tr><tr><td>wdArtBalloonsHotAir</td><td>12</td><td>wdArtPaperClips</td><td>82</td></tr><tr><td>wdArtBasicBlackDots</td><td>156</td><td>wdArtPencils</td><td>25</td></tr></table>	wdArtApples	1	wdArtMusicNotes	79	wdArtBalloonsHotAir	12	wdArtPaperClips	82	wdArtBasicBlackDots	156	wdArtPencils	25																		
wdArtApples	1	wdArtMusicNotes	79																													
wdArtBalloonsHotAir	12	wdArtPaperClips	82																													
wdArtBasicBlackDots	156	wdArtPencils	25																													

ArtWidth	Numeric	Ширина границы страницы, в пунктах.
Inside	Logical	Указывает, должен ли объект с рамкой иметь внутреннюю границу, если он ее поддерживает.

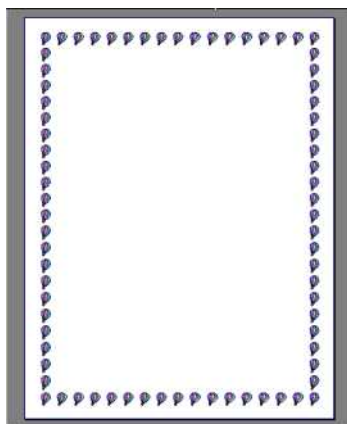


Рисунок 8. Рамка страницы. Коллекция Borders позволяет добавлять границы к различным объектам. Его интерпретация меняется вместе с объектом.

Затенение текста

После сложностей с границами хорошая новость заключается в том, что затенение стало намного проще. Здесь задействован только один объект, и у него всего несколько свойств.

Как и границы, затенение можно применять на нескольких уровнях. Многие объекты имеют свойство Shading, которое ссылается на объект Shading. Все объекты, перечисленные в Таблице 7, поддерживают затенение, за исключением Section, что имеет смысл, поскольку коллекция Borders Section на самом деле касается границ страниц, а это нечто иное, чем остальные.

Когда вы знаете, что хотите заштриховать, в определении затенения участвуют только три компонента: цвет фона, цвет переднего плана и текстура. А еще лучше, вы часто можете опустить последние два из них. Цвет фона – это именно то, что звучит как цвет текста. Для большинства печатных документов вам обычно понадобится какой-то вариант серого цвета. Вы можете указать его с помощью BackgroundPatternColor или BackgroundPatternColorIndex. Разница в том, что BackgroundPatternColor ожидает значение RGB, а свойство с более длинным именем берет свое значение из списка из 18 предопределенных цветов (их можно найти в Обозревателе объектов в разделе wdColorIndex). Чтобы указать светло-серую заливку, попробуйте:

```
oRange.Shading.BackgroundPatternColor = RGB(230, 230, 230)
```

ForegroundColor и ForegroundPatternColorIndex помещают другой цвет поверх цвета фона. Делать это не имеет особого смысла, если вы не укажете значение для Texture. Это просто альтернативный способ задания цвета штриховки (или сложный способ смешивания цветов).



BackgroundPatternColor и oforegroundPatternColor являются новыми в Word 2000. В Word 97 необходимо использовать индексные версии двух свойств и ограничиваться списком предопределенных цветов, которые они предоставляют.

Однако Word может делать гораздо лучшие трюки. Затенение можно приглушить от полной интенсивности до 5 процентов, всего предлагается около 40 шагов. Кроме того, для штриховки доступно около десятка других образцов заливки. В этом случае используются два цвета, как указано в их названиях. В Таблице 10 показаны некоторые константы, доступные для свойства Texture.

Таблица 10. Текстурированная штриховка. Word предлагает множество текстур, которые позволяют сделать штриховку не просто фоном.

Константа	Значение	Константа	Значение
wdTextureNone	0	wdTextureCross	-11
wdTextureSolid	1000	wdTextureDarkCross	-5
wdTexture5Percent	50	wdTextureHorizontal	-7
wdTexture10Percent	100	wdTextureVertical	-8
wdTexture12Pt5Percent	125	wdTextureDiagonalCross	-12
wdTexture30Percent	300	wdTextureDiagonalUp	-10
wdTexture75Percent	750	wdTextureDiagonalDown	-9

В этом примере устанавливается диагональная штриховка оранжевого цвета на бирюзовом фоне:

```
#DEFINE wdTextureDiagonalCross -12
WITH oRange.Shading
    .Texture = wdTextureDiagonalCross
    .BackgroundColor = RGB(0,192,255)
    .ForegroundColor = RGB(255,128,0)
ENDWITH
```

Верхние и нижние колонтитулы

Вы когда-нибудь замечали документ, в котором название главы располагалось вверху или было написано что-то вроде «Страница 3 из 17» внизу каждой страницы? Подобные элементы называются верхними и нижними колонтитулами. Word имеет для них огромную поддержку, позволяя размещать на первой странице каждого раздела документа иные заголовки, чем на других страницах, а также создавать отдельные верхние и нижние колонтитулы для нечетных и четных страниц.

В интерактивном режиме вы получаете доступ к верхнему и нижнему колонтитулу документа через меню «Вид» (см. рис. 9), что делает их доступными для редактирования и открывает для этой цели специальную панель инструментов. Панель инструментов включает инструменты для вставки общих элементов верхнего и нижнего колонтитула, таких как номера страниц, дата и время и т. д. Однако верхние и нижние колонтитулы могут содержать любой текст, а также графику, например логотипы компаний.

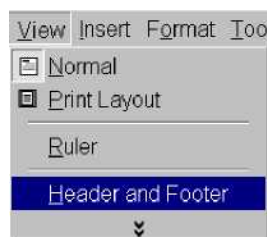


Рисунок 9. Добавление верхних и нижних колонтитулов. Используйте меню «Вид», чтобы сделать верхние и нижние колонтитулы видимыми в интерактивном режиме, чтобы вы могли их добавлять. В объектной модели они представлены объектами HeaderFooter.

Типы верхних и нижних колонтитулов в документе (разные на первой странице, разные на нечетных и четных страницах) контролируются с помощью объекта PageSetup и обсуждаются в разделе «Настройка страниц» ранее в этой главе. Содержимое верхнего и нижнего колонтитула хранится в объектах HeaderFooter, которые являются частью коллекции HeaderFooters, доступ к которой осуществляется через свойства Headers и Footers объекта Section. Любой раздел может иметь до трех таких элементов, доступ к которым осуществляется через константы wdHeaderFooterPrimary (1), wdHeaderFooterFirstPage (2) и wdHeaderFooterEvenPages (3).

Каждый объект HeaderFooter представляет отдельный верхний или нижний колонтитул. Ключевые свойства показаны в Таблице 11.

Таблица 11. Объект HeaderFooter. Несмотря на свое название, каждый экземпляр представляет собой отдельный верхний или нижний колонтитул.

Свойство	Тип	Описание
Exists	Logical	Указывает, существует ли указанный верхний или нижний колонтитул. Каким бы противоречивым ни казалось это свойство, оно имеет смысл при применении к члену коллекции HeadersFooters.
IsHeader	Logical	Указывает, является ли это верхним колонтитулом. Если нет, то, конечно, это нижний колонтитул.
PageNumbers	Object	Указатель на коллекцию PageNumbers, которая представляет информацию о номерах страниц в этом верхнем или нижнем колонтитуле. (См. следующий раздел.)
LinkToPrevious	Logical	Указывает, совпадает ли этот верхний или нижний колонтитул с соответствующим верхним или нижним колонтитулом в предыдущем разделе документа.

Нумерация страниц

Номера страниц могут появляться в любом месте документа, но обычно они размещаются либо в верхних, либо в нижних колонтитулах. Word предлагает различные варианты форматирования номеров страниц: от обычных арабских цифр до букв, римских цифр и даже нумерации глав (например, 1-1, 1-2 и т. д.). У вас также есть выбор положения номера. Он может располагаться в верхнем или нижнем колонтитуле, а также выравниваться по левому, центральному или правому краю, а также по внутреннему или внешнему полю. Последние два варианта применяются, конечно, только в том случае, если документ имеет разные нечетные и четные верхние и нижние колонтитулы. Все эти варианты доступны только в диалоговом окне «Вставка | Номера страниц». Если вы редактируете верхний или нижний колонтитул вручную, вы можете разместить номер страницы в любом месте внутри него.

Коллекция PageNumbers в Word состоит из отдельных объектов PageNumber. Каждый представляет собой одно вхождение номера страницы в верхнем или нижнем колонтитуле. PageNumber довольно прост. Единственное его важное свойство – Alignment, которое указывает горизонтальное положение номера страницы. В таблице 12 показаны возможные значения.

Таблица 12. Выравнивание номеров страниц. Эти константы являются доступными значениями свойства Alignment PageNumber.

Константа	Значение	Константа	Значение
wdAlignPageNumberLeft	0	wdAlignPageNumberInside	3
wdAlignPageNumberCenter	1	wdAlignPageNumberOutside	4
wdAlignPageNumberRight	2		

Коллекция PageNumbers – это место, где происходят все действия. Она имеет ряд свойств, описывающих нумерацию страниц внутри раздела. В Таблице 13 показаны те, которые вы, скорее всего, будете использовать.

Чтобы использовать нумерацию страниц по типу главы, номера глав должны быть идентифицируемыми. Для этого их связывают с уровнем структуры. (См. следующий раздел «Организация текста с помощью списков».)

Таблица 13. Коллекция PageNumbers. В каждом разделе документа можно заново начать нумерацию страниц и изменить внешний вид номеров страниц.

Свойство	Тип	Описание
NumberStyle	Numeric	Стиль, используемый для номера страницы. Проверьте обозреватель объектов на наличие дополнительных вариантов. wdCaptionNumberStyleArabic 0 wdCaptionNumberStyleUppercaseRoman 1 wdCaptionNumberStyleLowercaseRoman 2 wdCaptionNumberStyleUppercaseLetter 3 wdCaptionNumberStyleLowercaseLetter 4 wdCaptionNumberStyleHebrewLetter1 45
RestartNumberingAtSection	Logical	Указывает, начинаются ли номера страниц заново в начале этого раздела.
StartingNumber	Numeric	Номер первой страницы этого раздела, если RestartNumberingAtSection имеет значение .T.
IncludeChapterNumber	Logical	Указывает, используется ли нумерация типов глав. То есть номера страниц должны быть в формате «глава разделитель страница», например, 13-4.
HeadingLevelForChapter	Numeric	Схема, которая будет использоваться в качестве номера главы в номере страницы.
ChapterPageSeparator	Numeric	Символ, используемый между номером главы и номером страницы для нумерации типов глав.

		wdSeparatorHyphen 0 wdSeparatorEmDash 3 wdSeparatorPeriod 1 wdSeparatorEnDash 4 wdSeparatorColon 2
ShowFirstPageNumber	Logical	Указывает, отображается ли номер страницы на первой странице раздела.

Этот код центрирует номер страницы в нижнем колонтитуле, используя строчные римские цифры.

```
#DEFINE wdCaptionNumberStyleLowercaseRoman 2
#DEFINE wdAlignPageNumberCenter 1

WITH oWord.ActiveDocument.Sections[1].Footers[1]
    WITH .PageNumbers
        .Add()
        .NumberStyle = wdCaptionNumberStyleLowercaseRoman
    ENDWITH
    .PageNumbers[1].Alignment = wdAlignPageNumberCenter
ENDWITH
```

Организация текста со списками

Word предоставляет несколько альтернатив для структурирования текста в списки. Они объединяются в диалоговом окне «Маркеры и нумерация (Список)» меню «Формат». Маркеры позволяют отмечать элементы, которые не имеют определенного порядка, а нумерованные списки предназначены для элементов, которые должны располагаться в определенном порядке, например инструкций. Третья страница этого диалогового окна, вкладка «Структурная нумерация», предлагает параметры, которые еще более структурированы и связаны со способностью Word связывать стили со структурой документа. Как отмечалось в предыдущем разделе, вам необходимо использовать структурную нумерацию, чтобы пронумеровать страницы и главы.

В создании списков участвуют два набора объектов. Одна иерархия (ListGalleries, ListTemplates и ListLevels и их элементы) предоставляет набор вариантов внешнего вида списка, а другая (Lists и ListParagraphs и их элементы) содержит фактические элементы списка. Эта структура отражает способ применения стилей к содержимому документа. Доступны различные объекты Style, которые можно применять к содержимому, но любой диапазон имеет только один стиль. Аналогично, существует множество шаблонов списков, но к каждому списку применяется только один шаблон.

Определение структуры списка

Три коллекции и их содержимое предоставляют широкий выбор вариантов форматирования списков. На рисунке 10 показана задействованная иерархия. Коллекция ListGalleries, доступ к которой осуществляется через свойство ListGalleries объекта Application, начинает работу. Каждый объект ListGallery представляет одну страницу в диалоговом окне «Маркеры и нумерация». Для доступа к ним используются константы wdBulletGallery (1), wdNumberGallery (2) и wdOutlineNumberGallery (3).



Рисунок 10. Определение структуры списка. Эти три коллекции содержат определения для всех маркированных, нумерованных и структурированных списков, которые вы можете создать.

Каждый объект ListTemplate описывает способ расположения отдельного списка. Однако, как и ListGallery, ListTemplate сам по себе имеет мало свойств и методов. Единственное свойство, заслуживающее внимания, – это OutlineNumbered, которое указывает, указывает ли шаблон многоуровневый список структуры. В противном случае он ограничивается одним уровнем (что справедливо для всех списков, отображаемых на первых двух страницах диалогового окна «Маркеры и нумерация»). На рисунке 11 показана первая страница диалогового окна «Маркеры и нумерация», которая не имеет контурной нумерации, а на рисунке 12 показана последняя страница, которая имеет нумерацию.

Коллекция ListLevels состоит из объектов ListLevel, из которых собственно и состоят списки. В таблице 14 показаны интересные свойства ListLevel. Если вам не нравятся предоставленные структуры списков, это место, где можно что-то изменить. Например, в первом структурном нумерованном списке (показанном во второй позиции первой строки на рисунке 12), чтобы изменить список на использование квадратных скобок, а не круглых, вы можете использовать такой код:

```

#DEFINE wdOutlineNumberedGallery 3

LOCAL oLevel
WITH oWord.ListGalleries[ wdOutlineNumberedGallery ].ListTemplates[ 1 ]
  FOR EACH oLevel IN .ListLevels
    WITH oLevel
      * Изменим ")" на "]"
      .NumberFormat = STRTRAN(.NumberFormat, ")", "]")
      .NumberFormat = STRTRAN(.NumberFormat, "(", "[")
    ENDWITH
  ENDFOR
ENDWITH
  
```

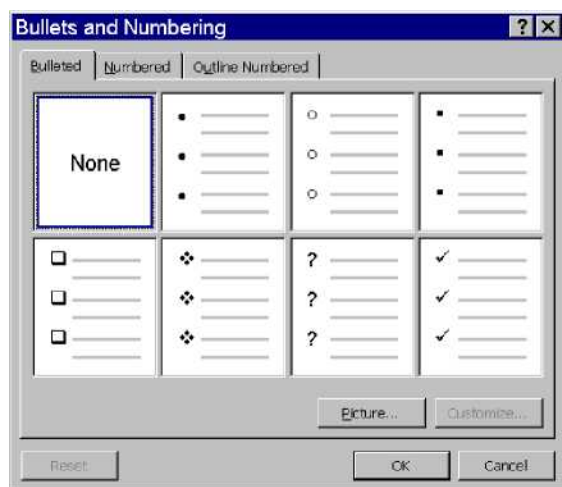


Рисунок 11. Список галерей без контуров. Первые две страницы диалогового окна «Маркеры и нумерация» содержат одноуровневые списки. Коллекция ListGalleries

содержит по одному элементу для каждой страницы диалогового окна. Каждый показанный вариант (кроме параметра «Нет») представлен одним объектом ListTemplate.

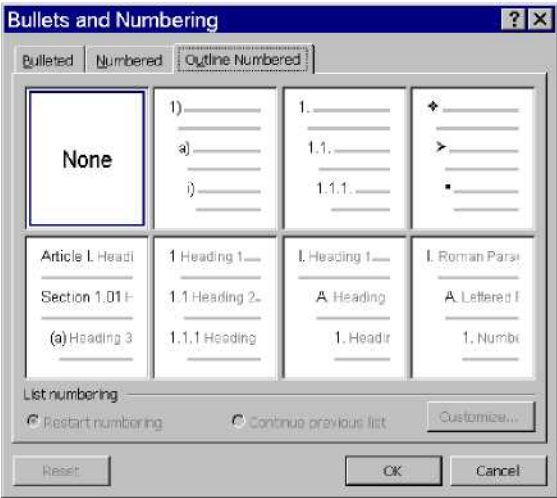


Рисунок 12. Многоуровневые списки. Третья страница диалогового окна предлагает структурную нумерацию и предоставляет третий член коллекции ListGalleries.

Каждый уровень можно связать со стилем с помощью свойства LinkedStyle, чтобы использование этого стиля автоматически указывало уровень нумерации и наоборот. Это дает несколько полезных способностей. Во-первых, как отмечалось ранее, включена нумерация внутри глав. Во-вторых, полезными инструментами становятся «Просмотр структуры» и «Карта документа» Word. Хотя ни один из них строго не требует нумерации, оба основаны на идее, что документ организован иерархически и что выбранные стили отражают эту иерархию. (Подробнее по этой теме см. «Организация документа с помощью стилей» в главе 6.) По умолчанию последние четыре шаблона списков на нумерованной странице «Структура» связаны со встроенными стилями заголовков.

Таблица 14. Определение структуры списка. ListLevel определяет, как расположен один уровень в списке.

Свойство	Тип	Описание
NumberStyle	Numeric	Стиль номера для этого уровня. wdListNumberStyleBullet 23 wdListNumberStyleArabic 0 wdListNumberStyleUppercaseRoman 1 wdListNumberStyleLowercaseRoman 2 wdListNumberStyleUppercaseLetter 3 wdListNumberStyleLowercaseLetter 4
NumberFormat	Character	Строка, показывающая, как вставить номер уровня в правильном стиле. Используйте %, за которым следует цифра, чтобы представить номер уровня, за исключением случаев, когда NumberStyle имеет значение wdListNumberStyleBullet. (В этом случае NumberFormat ограничен одним символом.)
TrailingCharacter	Numeric	Символ, следующий за номером. Используйте одну

		из ЭТИХ КОНСТАНТ: wdTrailingTab 0 wdTrailingSpace 1 wdTrailingNone 2
StartAt	Numeric	Начальный номер списка.
Alignment	Numeric	Выравнивание номера уровня или маркера. wdListLevelAlignLeft 0 wdListLevelAlignCenter 1 wdListLevelAlignRight 2
NumberPosition	Numeric	Отступ (в пунктах) номера уровня или маркера.
TabPosition	Numeric	Отступ (в пунктах) табуляции после номера уровня или маркера, то есть того места, где в этой строке должен начинаться текст.
TextPosition	Numeric	Отступ (в пунктах) текста во второй и последующих строках нумерованного или маркированного текста.
Font	Object	Шрифт, используемый для номера уровня.
LinkedStyle	Character	Название стиля, связанного с этим уровнем. (Пояснения смотрите в тексте.)

Создание списков

Коллекция Lists объекта Document ссылается на все списки в документе. Каждый объект List ссылается на один список. Ключевым свойством List является ListParagraphs, коллекция абзацев в списке. Если список существует, вы можете изменить его внешний вид, работая со свойствами этих объектов.

Процесс добавления нового списка в документ не прямой, особенно если вам нужен многоуровневый список. Вы не можете просто установить нужный формат и начать добавлять элементы, как это можно сделать в интерактивном режиме. Вам необходимо добавить текст в документ, а затем применить соответствующий шаблон списка. Затем для многоуровневого списка нужно вернуться назад и сделать соответствующие отступы.

Подготовив текстовый диапазон, готовый к форматированию, используйте свойство ListFormat Range для доступа к методу ApplyListTemplate. Этот метод принимает диапазон и превращает его в список, применяя к нему указанный объект ListTemplate. Необязательный четвертый параметр (здесь не показан) позволяет форматировать список более подходящим для веб-страниц способом.

```
oRange.ListFormat.ApplyListTemplate( oListTemplate [,
                                     lContinuePreviousList [, nApplyTo ] ] )
```

oListTemplate Object Ссылка на объект ListTemplate.

lContinuePreviousList Logical Указывает, следует ли продолжить нумерацию из предыдущего списка или начать заново с 1.

nApplyTo Numeric Часть списка, к которой следует применить указанный шаблон. Использует следующие константы Word:
wdListApplyToWholeList (0),

wdListApplyToThisPointForward (1),
wdListApplyToSelection (2).
Этот параметр позволяет изменить
форматирование части списка, что не обязательно
является наиболее эстетически приятным
вариантом.



Другими важными методами являются ListIndent и ListOutdent, которые позволяют понижать и повышать элементы по одному уровню в структурном списке.

Вот краткий пример создания многоуровневого списка:

```
#DEFINE CR CHR(13)
#DEFINE wdListApplyToWholeList 0

* Начните с пустого диапазона
oWord = CreateObject("Word.Application")
oWord.Visible = .T.
oDocument = oWord.Documents.Add()
oRange = oDocument.Range()

* Теперь создайте строку, содержащую элементы для вывода в список
cItems = "First item" + CR + ;
        "Second item " + CR + ;
        "first sub-item for item 2 " + CR + ;
        "second sub-item for item 2" + CR + ;
        "Third item" + CR

* Отправьте строку в Word
oRange.InsertAfter(cItems)

* Применение шаблона списка
oRange.ListFormat.ApplyListTemplate( ;
    oWord.ListGalleries[3].ListTemplates[1], ;
    .F., wdListApplyToWholeList )

* Теперь обрабатываем подэлементы
oRange.ListParagraphs[3].Range().ListFormat.ListIndent()
oRange.ListParagraphs[4].Range().ListFormat.ListIndent()
```

Вот результаты:

- 1) First item
- 2) Second item
 - a) first sub-item for item 2
 - b) second sub-item for item 2
- 3) Third item



Существует альтернатива использованию ListIndent. Помните возможность связывать стили с уровнями списка с помощью LinkedStyle? После этого, если вы примените указанные стили к абзацу и примените ApplyListTemplate, этот абзац

будет автоматически пронумерован. Вот пример, демонстрирующий этот подход. Результирующий список показан на рисунке 13 .

* Начните с пустого диапазона

```
#DEFINE CR CHR(13)
#DEFINE wdCollapseEnd 0
#DEFINE wdParagraph 4
#DEFINE wdListApplyToWholeList 0
```

* Начните с пустого диапазона

```
oWord = CreateObject("Word.Application")
oWord.Visible = .T.
oDocument = oWord.Documents.Add()
oRange = oDocument.Range()
```

* Теперь создайте строку, содержащую элементы для вывода в список

```
oRange.Style = "Заголовок 1" && "Heading 1"
oRange.InsertAfter("First item" + CR + ;
    "Second item " + CR )
oRange.Collapse( wdCollapseEnd )

oRange.Style = "Заголовок 2" && "Heading 2"
oRange.InsertAfter("first sub-item for item 2 " + CR + ;
    "second sub-item for item 2 " + CR )
oRange.Collapse( wdCollapseEnd )
```

```
oRange.Style = "Заголовок 1" && "Heading 1"
```

```
oRange.InsertAfter("Third item")
```

* Теперь сделайте диапазон всего этого

```
oRange.MoveStart( wdParagraph, -4 )
```

* Применение шаблона списка со связанными стилями

```
oRange.ListFormat.ApplyListTemplate( ;
    oWord.ListGalleries[3].ListTemplates[6], ;
    .F., wdListApplyToWholeList )
```

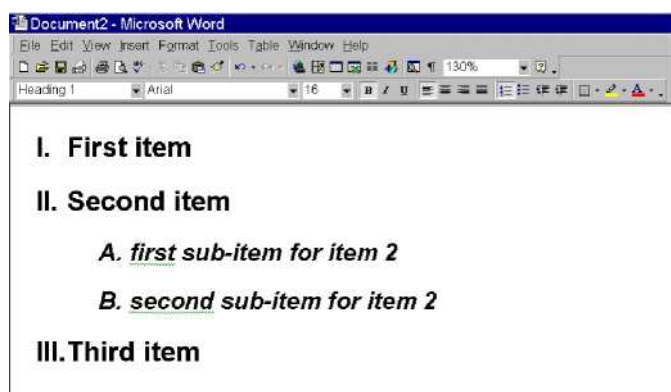


Рисунок 13. Использование стилей для создания списков. Связав стили с уровнями списка, вы можете автоматически выравнивать элементы при применении шаблона списка.

Объект ListFormat имеет свойства, описывающие внешний вид и структуру номера

уровня. В Таблице 15 описаны те, которые вы, вероятно, захотите использовать. Диапазон может содержать множество разных списков, но имеет только один объект ListFormat. Свойства ListFormat относятся только к первому списку в диапазоне. Чтобы просмотреть все различные списки в сложном диапазоне, разбейте диапазон на более мелкие диапазоны.

ListFormat в первую очередь занимается значением и форматированием номера абзаца, но также предоставляет свойства для доступа к самому списку и шаблону списка, на котором он основан.

Таблица 15. Свойства ListFormat. Объект ListFormat, на который ссылается Range, управляет внешним видом списков.

Свойство	Тип	Описание
ListType	Numeric	Тип списка. wdListNoNumbering 0 wdListSimpleNumbering 3 wdListNumOnly 1 wdListOutlineNumbering 4 wdListBullet 2 wdListMixedNumbering 5
ListLevelNumber	Numeric	Уровень отступа для первого абзаца этого объекта.
ListValue	Numeric	Номер абзаца без форматирования.
ListString	Character	Номер абзаца с указанным форматированием.
ListTemplate	Object	Ссылка на ListTemplate, с помощью которого был отформатирован список.
List	Object	Ссылка на первый список, содержащийся в объекте ListFormat.

4.6 Вывод данных

Создание привлекательных документов – хорошее начало, но пользователям обычно тоже нужен какой-то результат. В Word имеется несколько методов вывода данных. В интерактивном продукте они собраны в меню «Файл» в меню «Предварительный просмотр» и «Печать».

Печать

Метод PrintOut объекта Document автоматизирует печать. Он принимает огромный массив параметров. (Это одна из ситуаций, когда именованные параметры начинают выглядеть довольно хорошо.)

К счастью, все параметры являются необязательными, и большинство из них вам не понадобятся для нормальной обработки. К сожалению, те, которые вам, скорее всего, понадобятся, находятся в середине списка, а не в начале. Основные параметры показаны здесь:

```
oDocument.PrintOut( lBackground, , nPrintRangeType, cFileName,
                    cFrom, cTo, , cCopies, cPageRange, , lPrintToFile)
```

lBackground **Logical** Указывает, следует ли печатать в фоновом режиме, продолжая выполнять другой код во время печати.

nPrintRangeType **Numeric** Какая часть документа будет распечатана. (См. текст.)

cFileName	Character	При печати в файл имя файла, включая путь, — файл назначения.
cFrom, cTo	Character	Если nPrintRangeType имеет значение wdPrintFromTo, это начало и конец диапазона печати. (Смотрите текст.)
cCopies	Character	Количество копий для печати.
cPageRange	Character	Если nPrintRangeType имеет значение wdPrintRangeOfPages, это диапазон печати. (Смотрите текст.)
lPrintToFile	Logical	Указывает, следует ли печатать в файл.

Невероятно, но после этих параметров есть еще восемь. Показанный синтаксис отражает два разных подхода к указанию страниц для печати. Параметр nPrintRangeType определяет, какой из них будет использоваться. Допустимые значения констант: wdPrintAllDocument (0), wdPrintSelection (1), wdPrintCurrentPage (2), wdPrintFromTo (3) и wdPrintRangeOfPages (4). Когда передается wdPrintFromTo, значения cFrom и cTo передаются в качестве пятого и шестого параметров соответственно. Обратите внимание, что по причинам, известным только Microsoft, они передаются как символы. Например, чтобы распечатать три копии страниц 4-10 документа, на который ссылается oDocument, используйте:

```
oDocument.PrintOut( , , wdPrintFromTo, , "4", "10", , "3")
```

Выбор wdPrintRangeOfPages позволяет указать одну строку и обеспечивает большую гибкость. В этом случае передайте значение cPageRange в качестве восьмого параметра. Оно может включать несколько значений, разделенных запятыми, каждое из которых может включать диапазон. Например, вы можете передать что-то вроде «3, 4-6, 12».

Для печати в файл вы указываете как имя файла, так и флаг, который вы печатаете в файл, например:

```
oDocument.PrintOut( , , , "fileoutput", , , , , .t.)
```

Полученный файл готов к печати вместе с кодами принтера. Имейте в виду, что файл по умолчанию хранится в текущем каталоге Word, поэтому рекомендуется указать полный путь.

Имейте в виду, что при печати в файл устанавливается флажок «Печать в файл» в диалоговом окне «Печать» и он остается установленным. Если этот параметр опущен при последующем вызове PrintOut, по умолчанию используется значение .T.; вам нужно явно передать .F., чтобы распечатать на принтере.

Создание конвертов

Word также может автоматически создавать и печатать конверты. Для этого используйте объект Envelope документа. В этом примере предполагается, что oRange — это диапазон, содержащий имя и адрес клиента в почтовом формате. Он добавляет адрес в закладки (см. раздел «Закладки» ранее в этой главе, где представлено введение в этот метод идентификации части документа) и просит Word создать и распечатать конверт.

```
oDocument.Bookmarks.Add("EnvelopeAddress", oRange)
oDocument.Envelope.PrintOut(.T., , , .F., oWord.UserAddress)
```

Сначала мы создаем закладку под названием EnvelopeAddress на основе того, что

находится в `oRange`. Вызов `PrintOut` сообщает Word использовать в качестве адреса все, что находится в закладке `EnvelopeAddress`, и включить сохраненный адрес пользователя (со страницы информации о пользователе диалогового окна «Инструменты | Параметры») в качестве обратного адреса.

Параметры метода `PrintOut` объекта `Envelope` отличаются от параметров объекта `Document`. Вот сокращенная версия синтаксиса:

```
oDocument.Envelope.PrintOut( lExtractAddress, cAddress, ,  
                             lOmitReturnAddress, cReturnAddress )
```

<code>lExtractAddress</code>	Logical	Указывает, следует ли извлечь адрес из закладки <code>EnvelopeAddress</code> .
<code>cAddress</code>	Character	Адрес, который будет указан на конверте. Игнорируется, если <code>lExtractAddress</code> имеет значение <code>.T.</code>
<code>lOmitReturnAddress</code>	Logical	Указывает, следует ли опускать обратный адрес из конверта.
<code>cReturnAddress</code>	Character	Обратный адрес, который будет указан на конверте.

Дополнительные параметры позволяют задать тип и размер конверта, а также ориентацию печати. При исключении этих параметров используются текущие настройки.

Предварительный просмотр печати

Возможно, все, что хотят ваши пользователи, — это увидеть, как документ будет выглядеть при печати. Это легко. Просто вызовите метод `PrintPreview` объекта `Document`. Это переключит Word в режим `PrintPreview`.

Конечно, это полезно только в том случае, если Word виден. Если Word скрыт, это не принесет никакой пользы. Сделать Word видимым так же просто, как установить для свойства `Visible` приложения значение `.T.` А еще лучше, если вы все делали в фоновом режиме и теперь готовы показать пользователю, чем вы занимались, вызовите `PrintPreview`, сделайте Word видимым, а затем вызовите метод `Word Activate`. Это выведет Word на передний план. Попробуйте это так:

```
oDocument.PrintPreview && Подготовьте документ для просмотра  
                        && пользователем  
oWord.Visible = .t.    && Показать Word  
oWord.Activate()      && Выведите его на передний план
```

Независимо от того, скрываете ли вы Word большую часть времени или показываете его все время, когда вы закончите работу с предварительным просмотром, вы отключите его, вызвав метод `ClosePrintPreview` объекта `Document`.

4.7 Собираем все вместе



Листинг 3. показывает программу (`WordSample1.PRG` в файлах для загрузки разработчика, доступных на сайте www.hentzenwerke.com), которая создает документ, в котором перечислены все клиенты *Tasmanian Traders*, сгруппированные по странам. Он демонстрирует верхние и нижние колонтитулы, номера страниц, заливку и границы. На рисунке 14 показан предварительный просмотр полученного

документа.

Листинг 3. Отчет клиента Tasmanian Traders.

```
* WordSample1.PRG
* © 2000, Tamar E. Granor and Della Martin
* From: Microsoft Office Automation with Visual FoxPro
* Hentzenwerke Publishing. www.hentzenwerke.com

* Это пример «Собираем все вместе» из главы 4.

#DEFINE wdHeaderFooterPrimary      1
#DEFINE wdGray25                   16
#DEFINE wdStyleTypeParagraph       1
#DEFINE wdCaptionNumberStyleArabic 0
#DEFINE wdAlignPageNumberCenter    1
#DEFINE wdAlignParagraphCenter     1
#DEFINE wdCollapseEnd              0
#DEFINE wdLineStyleSingle          1
#DEFINE wdLineStyleNone            0
#DEFINE wdLineWidth050Point        4
#DEFINE CR CHR(13)

RELEASE ALL LIKE o*
PUBLIC oWord

LOCAL oDoc, oRange, cText, oHeaderStyle

oWord = CreateObject("Word.Application")
oWord.Visible = .T.

oDoc = oWord.Documents.Add()

* Создайте стиль для текста заголовка.
oHeaderStyle = oDoc.Styles.Add("CenteredHeader",
wdStyleTypeParagraph)
WITH oHeaderStyle
    .BaseStyle = oDoc.Styles["Заголовков 1"]    && ["Heading 1"]
    .ParagraphFormat.Alignment = wdAlignParagraphCenter
ENDWITH

* Добавить заголовок.
WITH oDoc.Sections[1].Headers[ wdHeaderFooterPrimary ]
    oRange = .Range()
    WITH oRange
        .Text = "Tasmanian Traders"
        .Style = oHeaderStyle
        * .Shading.BackgroundPatternColor = RGB(230, 230, 230)
        .Shading.BackgroundPatternColorIndex = wdGray25
    ENDWITH
ENDWITH

* Добавьте центрированный номер страницы в нижний колонтитул.
WITH oDoc.Sections[1].Footers[ wdHeaderFooterPrimary ].PageNumbers
```

```
.Add()
.NumberStyle = wdCaptionNumberStyleArabic
ENDWITH
oDoc.Sections[1].Footers[ wdHeaderFooterPrimary ;
    ].PageNumbers[1].Alignment = wdAlignPageNumberCenter

* Теперь создайте контент для документа.
* Получите список клиентов, упорядоченный по странам.
* Сортировка по странам в алфавитном порядке.
SELECT Company_Name, Country ;
    FROM _SAMPLES+"\\TasTrade\\Data\\Customer" ;
    ORDER BY Country, Company_Name ;
    INTO CURSOR CustomerByCountry

* Заголовок документа.
oRange = oDoc.Range()
oRange.InsertAfter("Customers By Country" + CR)
oRange.Style = oDoc.Styles("Заголовок 1")    && ("Heading 1")
oRange.Collapse( wdCollapseEnd )

* Циклическое перемещение курсора.
* Поместите каждое название страны в заголовок 2.
* Затем перечислите каждого клиента в этой стране.
LOCAL cCurrentCountry
cCurrentCountry = ""
cText = ""

SCAN
    IF NOT (Country==cCurrentCountry)
        * Новая страна.
        oRange.InsertAfter( cText + CR )
        oRange.Style = oDoc.Styles("Обычный")    && ("Normal")
        cText = ""
        oRange.Collapse( wdCollapseEnd )

        WITH oRange
            .InsertAfter( Country + CR )
            .Style = oDoc.Styles("Заголовок 2")&& ("Heading 2")
            .Borders.OutsideLineStyle = wdLineStyleSingle
            .Borders.OutsideLineWidth = wdLineWidth050Point
        ENDWITH
        oRange.Collapse( wdCollapseEnd )
        oRange.InsertAfter( CR )
        oRange.Borders.OutsideLineStyle = wdLineStyleNone
    ENDIF

    cText = cText + Company_Name + CR
    cCurrentCountry = Country
ENDSCAN

oRange.InsertAfter( cText )
oRange.Style = oDoc.Styles("Обычный")    && ("Normal")
```

```
USE IN CustomerByCountry
USE IN Customer
```

```
oDoc.PrintPreview()
oWord.Activate()           && Выведите его на передний план
```

В этой главе охвачено невероятное количество материала, и мы только разогреваемся. Word может предложить гораздо больше. Следите за обновлениями.

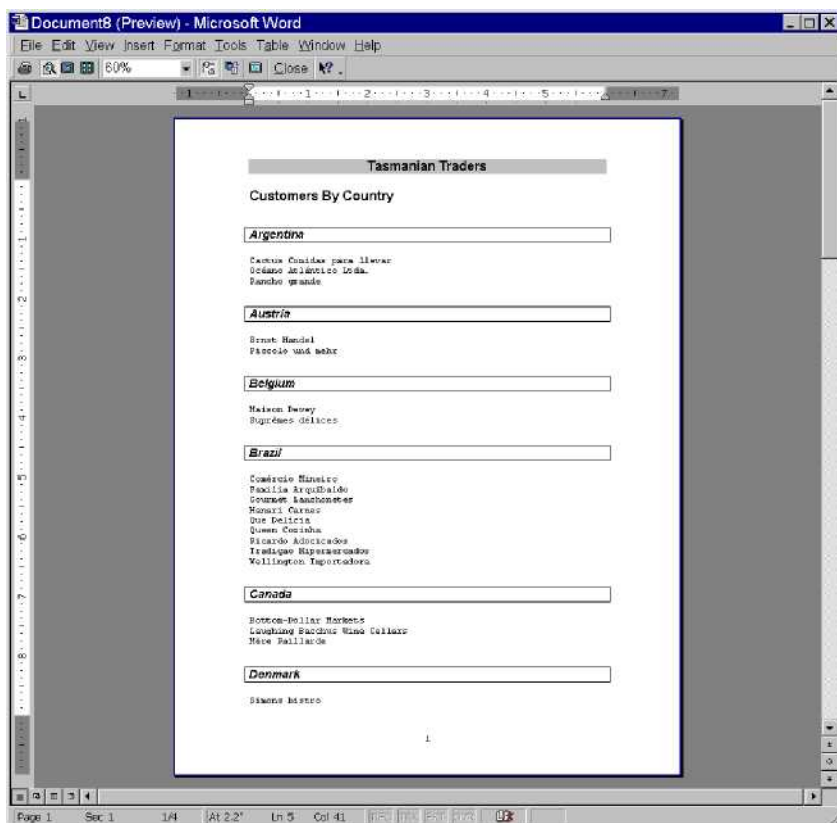


Рисунок 14. Отчет клиента. В списке клиентов Tasmanian Traders показаны границы, заливка, верхние и нижние колонтитулы, номера страниц и многое другое.

Глава 5. Промежуточный Word

Word предлагает множество возможностей сделать документы более привлекательными и единообразными, а также работать с существующими документами.

Как только вы освоите основы Автоматизации Word, перед вами появится множество возможностей. В этой главе рассматриваются различные способы, с помощью которых можно усложнить работу Word, чтобы вам не приходилось это делать, включая создание и использование шаблонов документов, работу с таблицами, создание документов с несколькими столбцами, добавление сносок и концевых сносок, добавление графики в документы, Автоматизацию макросов, использование поиска и замены, а также аудит документов.

5.1 Шаблоны документов

Шаблоны Word позволяют обеспечить стандартное форматирование и стили для документов. Подобно классу формы в Visual FoxPro, шаблон позволяет вам указывать общие элементы, такие как настройки страницы, верхние и нижние колонтитулы и даже шаблонный текст, чтобы каждый новый документ, который вы создаете, содержал одни и те же элементы. Но в остальном шаблоны больше похожи на библиотеки классов, поскольку содержат стили, которые можно использовать в документе.

Каждый документ, который вы создаете в Word, основан на шаблоне – файле с расширением DOT. Если вы не укажете, какой шаблон использовать, используется обычный шаблон. По умолчанию «Обычный» – это пустой документ, содержащий стандартный набор стилей. Он устанавливается вместе с Word вместе с рядом других шаблонов. Чтобы просмотреть доступные шаблоны, выберите в меню «Файл|Новый». Появится диалоговое окно «Новый» (показанное на рисунке 1). Обычный шаблон отображается как «Пустой документ».

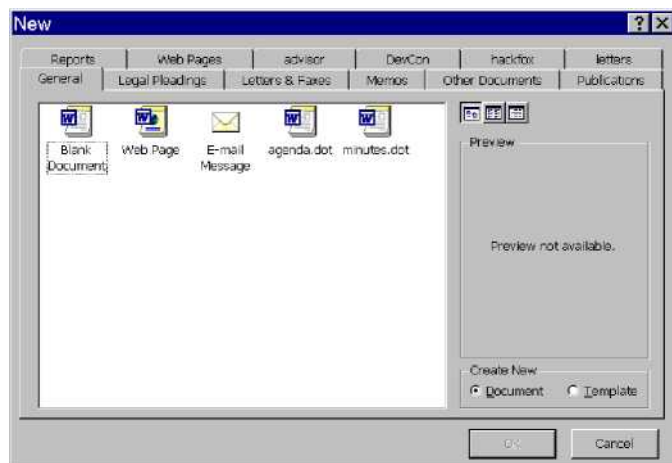


Рисунок 1. Доступные шаблоны. Выбор «Файл|Новый» в меню Word позволяет выбрать один из всех сохраненных шаблонов.

Что входит в шаблон?

Шаблон может содержать все, что есть в документе, от простого пустого документа стандартного шаблона до сложного юридического документа, полного, за исключением имен участвующих сторон. Большинство шаблонов находятся где-то посередине и содержат некоторый текст и форматирование, а также некоторые пользовательские стили.

Шаблоны также могут содержать макросы, в том числе тот, который запускается

автоматически при создании документа на основе шаблона. Шаблоны также могут включать закладки, которые можно заполнить либо по запросу пользователя, либо с помощью Автоматизации. Короче говоря, шаблонов может быть столько, сколько вы захотите.

Поиск шаблонов

Шаблоны хранятся в нескольких разных местах. Шаблоны (и мастера), установленные Word, помещаются в подкаталог «Templates» установки Office. В Word 2000 они затем опускаются на один уровень ниже в подкаталог, названный числовым кодом используемого вами языка (1033 для американского английского). В Word 97 в разделе «Templates» имеется набор подкаталогов, в которых содержатся различные шаблоны и мастера. (В обоих случаях мастера имеют расширение WIZ.)

Кроме того, каждый пользователь может установить каталог шаблонов пользователя и каталог шаблонов рабочей группы. Каталог пользовательских шаблонов предназначен для индивидуальных шаблонов, а каталог шаблонов рабочей группы предназначен для шаблонов, совместно используемых всей командой пользователя. Местоположение пользовательских шаблонов по умолчанию зависит от версии Word и Windows, но оно не всегда находится в дереве каталогов Office. В некоторых случаях шаблоны хранятся как часть профилей пользователей в дереве каталогов Windows. Независимо от того, где находится каталог пользовательских шаблонов, шаблоны из этого каталога отображаются на странице «Общие» диалогового окна «Создать». Шаблоны в подкаталогах этого каталога размещаются на дополнительных страницах диалога, страницы помечаются именами подкаталогов. Создание новых подкаталогов создает дополнительные страницы, как вы можете видеть на **рисунке 1**.

По умолчанию параметр «Шаблоны рабочих групп» пуст. Когда он установлен, применяется та же двухуровневая структура. Шаблоны в самом каталоге отображаются на странице «Общие», а подкаталоги имеют собственные страницы в диалоговом окне «Создать».

Оба параметра можно установить на странице «Местоположение файлов» диалогового окна «Инструменты (Сервис) | Параметры» Word (см. **рис. 2**).

С точки зрения Автоматизации вы можете определить (или указать) расположение шаблонов пользователей и рабочих групп, проверив свойство DefaultFilePath объекта «Options» (который является членом объекта «Application» Word). Передайте wdUserTemplatesPath (2), чтобы найти собственные шаблоны пользователя, и wdWorkgroupTemplatesPath (3), чтобы найти шаблоны рабочей группы. (Метод NewDocument класса cusWord в файлах загрузки для разработчиков, доступных по адресу www.henzenwerke.com принимает шаблон с путем или без него и пытается найти файл шаблона перед созданием нового документа. Класс cusWord обсуждается в главе 15 «Подведение итогов по серверам».)

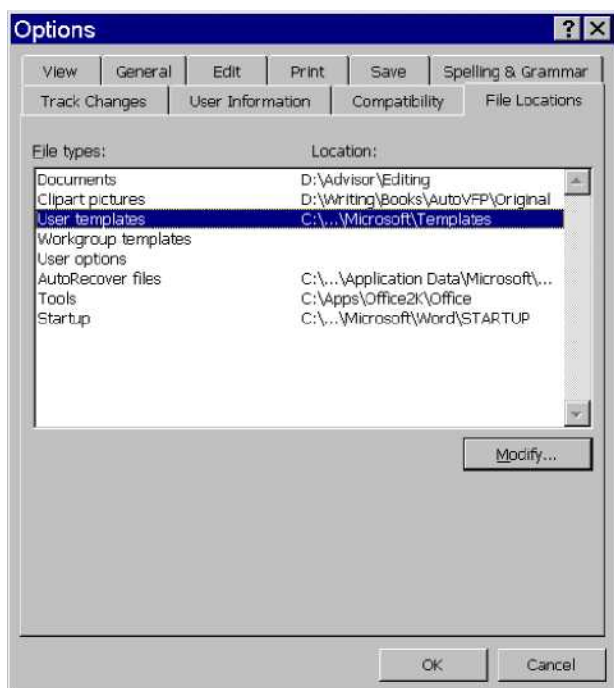


Рисунок 2. Указание на шаблоны. Диалоговое окно «Параметры Word» позволяет указать, где хранятся шаблоны пользователей и рабочих групп.

Использование шаблонов

При Автоматизации Word можно использовать шаблоны несколькими способами. Самый простой – создавать новые документы на основе существующих шаблонов. Для этого укажите шаблон, включая путь, в качестве первого параметра метода Documents.Add. Например, чтобы создать новый документ на основе шаблона agenda, показанного на рисунке 1, вы должны использовать следующий код. (В отличие от других примеров в этой книге, вы не можете работать с этим, потому что на вашей машине нет agenda.DOT. Мы вернемся к этому примеру позже и рассмотрим тот, который вы действительно можете попробовать.)

```
#DEFINE wdUserTemplatesPath 2
cTemplatePath = oWord.Options.DefaultFilePath(wdUserTemplatesPath)
* AddBS() добавляет обратную косую черту в конце
* — до версии VFP 6 вам необходимо загрузить FoxTools.
oDocument = oWord.Documents.Add( AddBS(cTemplatePath) + ;
    "agenda.DOT" )
```

Создав новый документ на основе шаблона, вы можете обращаться с ним так же, как с любым другим новым документом. Однако у вас есть то преимущество, что он содержит весь специальный текст, форматирование и стили, хранящиеся в шаблоне.

Создание шаблонов

Вы также можете создавать шаблоны с помощью Автоматизации. Любой документ можно сохранить как шаблон, передав соответствующий параметр методу SaveAs. Чтобы создать новый шаблон, создайте документ, отформатируйте его по желанию, создайте любые стили, которые вы хотите, чтобы шаблон имел, затем вызовите SaveAs, например:

```
#DEFINE wdFormatTemplate 1
#DEFINE wdUserTemplatesPath 2
cTemplatePath = oWord.Options.DefaultFilePath(wdUserTemplatesPath)
```

```
oDocument.SaveAs( AddBS(cTemplatePath) + "MyNewTemplate.DOT", ;  
    wdFormatTemplate )
```

Теперь, когда мы создали собственный шаблон, мы можем создать на его основе новый документ. Вот пример из предыдущего раздела, но с использованием нового шаблона, а не agenda.DOT:

```
#DEFINE wdUserTemplatesPath 2  
cTemplatePath = oWord.Options.DefaultFilePath(wdUserTemplatesPath)  
* AddBS() добавляет обратную косую черту в конце  
* — до версии VFP 6 вам необходимо загрузить FoxTools.  
oDocument = oWord.Documents.Add( AddBS(cTemplatePath) + ;  
    "MyNewTemplate.DOT" )
```

Как и в интерактивном Word, вы можете сохранить шаблон в подкаталоге, чтобы он отображался на другой странице в диалоговом окне «Файл | Создать». Конечно, если вы работаете с ним через Автоматизацию, вам все равно, где он появится. Фактически, при использовании Автоматизации не имеет значения, где вы храните шаблоны, поскольку вы можете указать, где Word должен их искать. Однако сохранение их вместе с другими шаблонами означает, что интерактивные пользователи также смогут их найти. С другой стороны, для автоматизированного процесса вы можете предпочесть хорошо скрыть свои шаблоны, чтобы пользователи не могли их найти, удалить или использовать по незнанию.

Вы также можете указать, что новые документы, которые вы создаете, должны быть шаблонами. Передайте .T. в качестве второго параметра метода Add коллекции Documents, чтобы указать, что новый документ является шаблоном, а не обычным документом. Затем, когда вы сохраняете его, он получает расширение DOT вместо DOC, и местоположением по умолчанию для него является каталог пользовательских шаблонов. В этом случае вам не нужно передавать второй параметр в SaveAs.

Использование шаблонов в работе

Мы обе широко используем шаблоны в своей работе, чтобы обеспечить единообразный внешний вид документов и сэкономить время и усилия. Мы не использовали Word без шаблонов так же, как не использовали бы VFP без классов. Шаблоны позволяют упростить создание документов, особенно в сочетании с возможностью определения пользовательских стилей.

5.2 Таблицы

Таблицы Word кажутся естественными для представления данных VFP. Строка может представлять запись, а каждый столбец — поле. Что Word дает вам, так это возможность форматировать данные и саму таблицу сложными способами, выходящими далеко за рамки возможностей дизайнера отчетов FoxPro, а также позволяет пользователям манипулировать результатами или создавать выходные данные в альтернативных форматах. Таблицу можно форматировать целиком, но отдельные ячейки также можно форматировать отдельно. Границы таблиц и ячеек могут быть видимыми или невидимыми, а также могут иметь различные размеры и стили. (Подробную информацию о форматировании границ см. в разделе «Границы и заливка» в главе 4.) Как столбцы, так и строки могут иметь индивидуальный размер.

Иерархия объектов таблиц немного сбивает с толку. Каждый документ имеет коллекцию Tables, которая, в свою очередь, содержит отдельные объекты Table. Объект Table содержит коллекции Rows и Columns, которые содержат объекты Row

и Column соответственно. Каждый из этих объектов имеет коллекцию Cells, которая ссылается на отдельные ячейки в строке или столбце, каждая из которых представлена объектом Cell. Хотя объект Table не имеет коллекции Cells, доступ к отдельным объектам Cell можно получить с помощью метода Cell, который принимает параметры номера строки и столбца. Вот несколько способов обращения к ячейке в третьей строке и четвертом столбце первой таблицы активного документа:

```
oWord.ActiveDocument.Tables[1].Rows[3].Cells[4]
oWord.ActiveDocument.Tables[1].Columns[4].Cells[3]
oWord.ActiveDocument.Tables[1].Cell[3,4]
```

В интерактивном режиме вы можете добавить таблицу, выбрав в меню «Таблица|Вставка|Таблица», после чего появится диалоговое окно, показанное на рисунке 3.

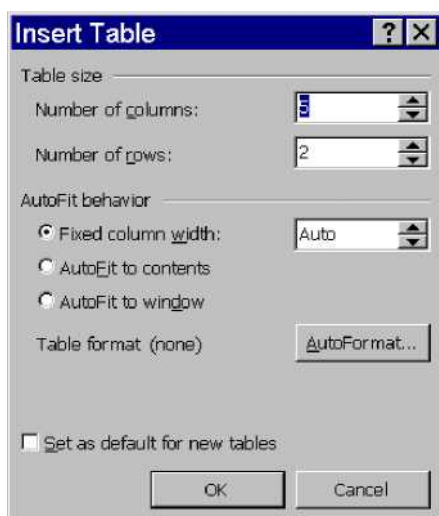


Рисунок 3. Добавление таблицы. Чтобы добавить таблицу в интерактивном режиме, вы указываете количество столбцов и строк и, если хотите, должны ли столбцы автоматически изменять размер в соответствии с их содержимым. Вы также можете указать predetermined формат таблицы.

Поскольку в случае Автоматизации Tables представляет собой коллекцию, неудивительно, что способ добавить таблицу в документ – это вызвать метод Add этой коллекции. Он принимает несколько параметров:

```
oDocument.Tables.Add( oRange, nRows, nColumns, nTableBehavior,
nAutoFit)
```

oRange	Object	Ссылка на диапазон, указывающий, куда вставить новую таблицу.
nRows	Numeric	Число строк в новой таблице.
nColumns	Numeric	Число столбцов в новой таблице.
nTableBehavior	Numeric	Константа, указывающая, изменяется ли размер таблицы автоматически в соответствии с ее содержимым.
	wdWord8TableBehavior 0	размер автоматически не изменяется.
	wdWord9TableBehavior 1	Автоматическое

изменение размера.

nAutoFit	Numeric	Если nTableBehavior имеет значение wdWord9TableBehavior, то указывает, какой алгоритм используется для изменения размера ячеек.	
wdAutoFitFixed	0	Фиксированная ширина столбца.	
wdAutoFitContent	1	Размер ячеек в соответствии с содержимым.	
wdAutoFitWindow	2	Размер таблицы на всю ширину окна	

Последние два параметра являются необязательными. Если вы их опустите, вы получите старое поведение таблицы Word 97, размер которой не изменяется по мере ее заполнения. Однако вы можете изменить это поведение. Свойство AllowAutoFit и метод AutoFitBehavior управляют этой возможностью изменения размера.

Даже если вы не разрешаете автоматическое изменение размера столбцов, метод AutoFit объекта Column позволяет изменять размеры отдельных столбцов в зависимости от их содержимого. Вместо того, чтобы изменять размер ячеек при вводе данных, вы применяете изменения после того, как что-то поместили в таблицу. Метод SetWidth объекта Column позволяет установить для столбца определенную ширину в пунктах.

После интерактивного добавления таблицы в диалоговом окне «Свойства таблицы» меню «Таблица» можно настроить различные характеристики таблицы в целом, а также отдельных строк, столбцов и ячеек. На рисунке 4 показана страница «Строка» этого диалогового окна.

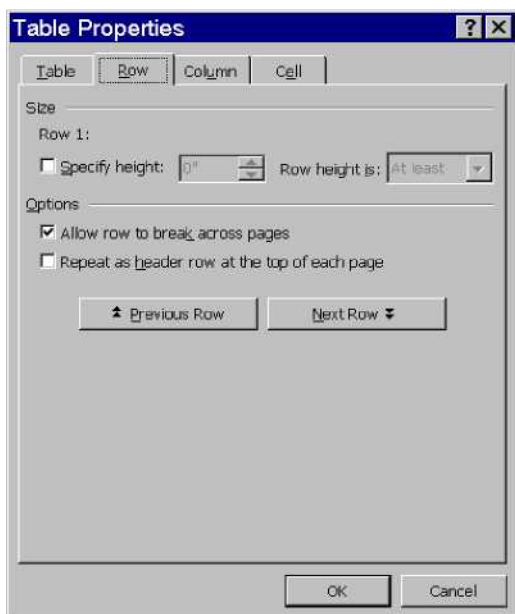


Рисунок 4. Форматирование таблицы. Диалоговое окно «Свойства таблицы» позволяет изменять размеры строк и столбцов, а также указывать другие характеристики таблиц и их компонентов.

Таблица 1 показывает наиболее часто используемые свойства объекта Table.

Таблица 1. Определение таблиц. С этими свойствами объекта Table вам, скорее всего, придется работать.

Свойство	Тип	Описание
Rows	Object	Указатель на коллекцию Rows для таблицы.
Columns	Object	Указатель на коллекцию Columns для таблицы.
Uniform	Logical	Указывает, имеет ли каждая строка одинаковое количество столбцов. (Только чтение)
Borders	Object	Указатель на коллекцию Borders для таблицы.
Shading	Object	Указатель на объект Shading для таблицы.
AllowAutoFit	Logical	Указывает, изменяется ли размер столбцов автоматически при добавлении данных в таблицу. Соответствует параметру nTableBehavior метода Tables.Add.
AllowPageBreaks	Logical	Указывает, можно ли разделить таблицу на несколько страниц документа.
AutoFormatType	Numeric	Константа, указывающая, какой из наборов predefined форматов был применен к таблице (если таковой имеется). Автоформаты применяются с помощью метода AutoFormat. Вот выборка: wdTableFormatNone 0 wdTableFormatSimple1 1 wdTableFormatClassic1 4 wdTableFormatColorful1 8 wdTableFormatContemporary 35 wdTableFormatElegant 36 wdTableFormatGrid1 16
Spacing	Numeric	Указывает расстояние между ячейками в пунктах. Это пространство, которое фактически разделяет ячейки, а не граница между границей ячейки и текстом.

Неудивительно, что Row и Column имеют ряд общих свойств, включая Cells для указания на коллекцию Cells и Shading для ссылки на объект Shading. У Row также есть свойство Borders, которое ссылается на коллекцию Borders, а у Column – нет. Оба объекта имеют логические свойства IsFirst и IsLast, которые, как следует из их названий, указывают, является ли конкретная строка или столбец первой или последней в коллекции.

На этом этапе два объекта расходятся, хотя сходство все еще существует. Размер строки определяется HeightRule и Height, как указано в таблице 2. Ширина столбца также использует два свойства PreferredWidth и PreferredWidthType, показанные в таблице 3.

У строки есть еще одно свойство, связанное с размером, SpaceBetweenColumns.

Оно указывает расстояние между границами ячейки и текстом. Значение свойства вдвое меньше того, которое вы установили в самом Word, поскольку оно измеряется от текста в одной ячейке до текста в следующей ячейке.

Свойство Row AllowBreakAcrossRows определяет, что происходит, когда содержимое строки не помещается на текущей странице. Если это .T., строку можно разделить на две страницы; если .F., перед строкой происходит разрыв страницы.

Когда таблица разделена на несколько страниц, строки, для свойства HeadingFormat которых установлено значение .T. повторяются.

Одно большое различие между строкой и столбцом заключается в том, что строка может быть диапазоном, а столбец – нет.

Таблица 2. Определение размера строки. Два свойства Row объединяются, чтобы указать высоту строки.

Свойство	Тип	Описание
HeightRule	Numeric	Указывает логику, используемую для определения высоты этой строки. Используйте одну из следующих констант: wdRowHeightAuto 0 wdRowHeightAtLeast 1 wdRowHeightExactly 2
Height	Numeric	Высота строки, если HeightRule имеет значение wdRowHeightExactly. Минимальная высота строки, если HeightRule имеет значение wdRowHeightAtLeast. Игнорируется (и неинформативно при запросе), если HeightRule имеет значение wdRowHeightAuto; в этом случае высота строки зависит от ее содержимого.

Таблица 3. Указание ширины столбца. Как и в случае со строками, два свойства объединяются для указания размера столбца.

Свойство	Тип	Описание
PreferredWidth	Numeric	Желаемая ширина этого столбца в пунктах или в процентах от общей ширины окна. Интерпретация определяется PreferredWidthType.
PreferredWidthType	Numeric	Указывает, измеряется ли PreferredWidth в пунктах или процентах или игнорируется. wdPreferredWidthAuto 0 Размер столбца по содержимому. wdPreferredWidthPoints 1 Размер столбца в пунктах. wdPreferredWidthPercent 2 Размер столбца в процентах от общего окна.

Cell имеет ряд общих свойств с Table, Row и Column, включая Borders, Shading,

HeightRule, Height, PreferredWidth и PreferredWidthType. В таблице 4 показаны некоторые другие свойства, уникальные для Cell.

Таблица 4. Характеристики клетки. В конце перечня характеристик таблицы ячейки имеют довольно много свойств. Вот некоторые из них, с которыми вам, вероятно, придется иметь дело.

Свойство	Тип	Описание
Width	Numeric	Ширина ячейки в пунктах.
WordWrap	Logical	Указывает, переносится ли текст на несколько строк, а высота ячейки увеличивается, чтобы вместить все содержимое.
FitText	Logical	Указывает, уменьшается ли отображаемый размер текста (но не фактический размер шрифта), чтобы все содержимое ячейки помещалось в одну строку
VerticalAlignment	Numeric	Указывает вертикальное положение текста в ячейке. wdAlignVerticalTop 0 wdAlignVerticalCenter 1 wdAlignVerticalBottom 3

Таблица, строка и ячейка имеют свойства диапазона, поэтому всю таблицу, строку или ячейку можно легко преобразовать в диапазон. Это означает, что для вставки текста в таблицу используются те же методы, что и для других частей документа. Однако диапазон, созданный из ячейки, содержит специальный маркер конца ячейки. Чтобы получить доступ только к тексту в ячейке, переместите конец диапазона на один символ назад. Любое из следующих действий поможет:

```
oRange.End = oRange.End - 1
oRange.MoveEnd( wdCharacter, -1 )
```



Программа, показанная в листинге 1, открывает представление истории заказов TasTrade и создает таблицу Word, показывающую историю заказов текущего клиента. Она демонстрирует различные функции, включая границы, затенение и автоматическое изменение размера столбцов. Программа называется OrderTblFormat.PRG в файлах для загрузки разработчика, доступных по адресу www.hentzenwerke.com.

Листинг 1. Создание таблицы. Эта программа генерирует таблицу, содержащую историю заказов клиента.

```
* OrderTblFormat.PRG
* © 2000, Tamar E. Granor and Della Martin
* From: Microsoft Office Automation with Visual FoxPro
* Hentzenwerke Publishing. www.hentzenwerke.com
```

```
* Создайте таблицу Word с информацией о заказе для одного клиента.
* Настройте таблицу с двумя строками, отформатировав вторую строку
* для данных. Затем добавьте необходимые строки для каждой записи.
```

```
#DEFINE wdStory
```

```

#DEFINE wdCollapseEnd          0
#DEFINE CR                      CHR(13)
#DEFINE wdBorderTop            -1
#DEFINE wdLineStyleDouble      7
#DEFINE wdAlignParagraphLeft   0
#DEFINE wdAlignParagraphCenter 1
#DEFINE wdAlignParagraphRight  2

RELEASE ALL LIKE o*
PUBLIC oWord

LOCAL oRange, oTable, nRecCount, nTotalOrders
LOCAL nRow

oWord = CreateObject("Word.Application")
oWord.Documents.Add()
* oWord.Visible = .T.    && отладка

OPEN DATABASE (_SAMPLES + "Tastrade\Data\Tastrade")
USE CUSTOMER
GO RAND()*RECCOUNT()    && выбрать клиента случайным образом

* Откройте представление «История заказов», которое
* содержит сводку заказов для одного клиента.
SELECT 0
USE "Order History" ALIAS OrderHistory

* Узнайте, сколько записей.
nRecCount = _TALLY

oRange = oWord.ActiveDocument.Range()

* Настроить шрифт для таблицы.
oRange.Font.Name = "Arial"
oRange.Font.Size = 12

oRange.InsertAfter( 'Customer '+TRIM(Customer.Company_Name) + ;
    ' ('+TRIM(Customer.Customer_Id)+') ' )

* Перейти в конец документа.
* Вставьте две пустые строки.
oRange.MoveEnd( wdStory )
oRange.Collapse( wdCollapseEnd )
oRange.InsertAfter( CR + CR )
oRange.Collapse( wdCollapseEnd )

* Добавить таблицу с двумя строками.
oTable = oWord.ActiveDocument.Tables.Add( oRange, 2, 4)

WITH oTable
    * Настройте границы и затенение.
    * Сначала удалите все границы.
    .Borders.InsideLineStyle = .F.
    .Borders.OutsideLineStyle = .F.

    * Заштрихуйте первый ряд заголовков.
    .Rows[1].Shading.Texture = 100

    * Вставьте текст заголовка и настройте выравнивание.
    .Cell[1,1].Range.ParagraphFormat.Alignment = wdAlignParagraphRight
    .Cell[1,1].Range.InsertAfter("Order Number")

    .Cell[1,2].Range.ParagraphFormat.Alignment = wdAlignParagraphLeft
    .Cell[1,2].Range.InsertAfter("Date")

```

```

.Cell[1,3].Range.ParagraphFormat.Alignment = wdAlignParagraphRight
.Cell[1,3].Range.InsertAfter("Total")

.Cell[1,4].Range.ParagraphFormat.Alignment = wdAlignParagraphCenter
.Cell[1,4].Range.InsertAfter("Paid?")

* Форматирование ячеек данных.
.Cell[2,1].Range.ParagraphFormat.Alignment = wdAlignParagraphRight
.Cell[2,3].Range.ParagraphFormat.Alignment = wdAlignParagraphRight
.Cell[2,4].Range.ParagraphFormat.Alignment = wdAlignParagraphCenter

* Добавьте данные и отформатируйте.
* Подсчитайте общую сумму по пути.
nTotalOrders = 0
FOR nRow = 1 TO nRecCount
    WITH .Rows[nRow + 1]
        .Cells[1].Range.InsertAfter( Order_Id )
        .Cells[2].Range.InsertAfter( TRANSFORM(Order_Date, "@D") )
        .Cells[3].Range.InsertAfter( TRANSFORM(Ord_Total, "$$$$$$$$9.99") )
        * Поставьте "X" в четвертом столбце, если выплачено;
        * в противном случае пусто.
        IF Paid
            .Cells[4].Range.InsertAfter("X")
        ENDIF
    ENDWITH

    * Добавить новую строку.
    .Rows.Add()

    * Текущая сумма.
    nTotalOrders = nTotalOrders + Ord_Total
    SKIP
ENDFOR

* Добавьте двойную линию перед итоговыми суммами.
.Rows[nRecCount + 2].Borders[ wdBorderTop ].LineStyle = wdLineStyleDouble

* Поместите итоговую строку.
WITH .Rows[ nRecCount + 2]
    .Cells[1].Range.InsertAfter("Total")
    .Cells[3].Range.InsertAfter(TRANSFORM(nTotalOrders, "$$$$$$$$9.99"))
ENDWITH

* Размер столбцов.
* Для простоты позвольте Word сделать всю работу.
.Columns.Autofit
ENDWITH

* Покажем результат.
oWord.Visible = .T.

USE IN OrderHistory
USE IN Order_line_items
USE IN Orders
USE IN Customer

RETURN

```

Результаты показаны на рисунке 5.

Customer The Cracker Box (THECR)

Order Number	Date	Total	Paid?
1004	28.02.95	\$1152.97	X
776	05.11.94	\$585.33	X
625	01.07.94	\$22870.25	X
58	13.08.92	\$34.68	X
Total		\$24643.24	

Рисунок 5. Использование таблиц для данных. История заказов клиента выглядит хорошо, если ее занести в таблицу Word.

Код создает таблицу из двух строк, вставляет заголовки, а затем форматирует ячейки во второй строке. Затем цикл вставляет данные и добавляет новую строку. Каждая новая строка сохраняет форматирование предыдущей, поэтому форматы нужно применить только один раз.

Вы можете объединить код (удалив часть, которая выбирает запись о клиенте) со Styles.PRG из главы 4 «Основы Word» (результат показан на рисунке 7 в этой главе), и вы получите достаточно привлекательный отчет об истории заказов для клиента. Оберните это в цикл с помощью еще нескольких команд (например, InsertBreak для добавления разрывов страниц), и вы сможете создавать истории заказов для всех клиентов или выбранного набора.

Нестандартные таблицы

Таблицы не обязательно должны состоять из простых сеток. Не каждая строка должна иметь одинаковое количество столбцов. Методы Merge и Split Cell и Cells позволяют объединять и разбивать на части группы ячеек для создания таблиц неправильной формы. Свойство Uniform таблицы указывает, является ли таблица регулярной или нет; обязательно проверьте это перед использованием вложенных циклов FOR EACH, чтобы попытаться обработать каждую строку и столбец в таблице.

Метод Merge работает двумя способами. Вы можете либо вызвать его для одной ячейки и передать его другой, чтобы объединить эти две, либо вы можете вызвать его с диапазоном ячеек, чтобы объединить эти ячейки. Вот два формата синтаксиса:

```
oFirstCell.Merge( oSecondCell )
oCells.Merge()
```

Например, чтобы объединить вторую и третью ячейки в строке 1 таблицы oTable, вы можете использовать этот код (все нижеследующее предполагает, что oTable является ссылкой на таблицу, с которой вы работаете):

```
oTable.Cell(1,2).Merge( oTable.Cell(1,3) )
```

Чтобы превратить четвертую строку таблицы в одну ячейку, используйте такой код:

```
oRange = oTable.Rows[4].Range()
oRange.Cells.Merge()
```

Рисунок 6 показывает таблицу (которая начиналась с семи строк и пяти столбцов) после внесения этих двух изменений.

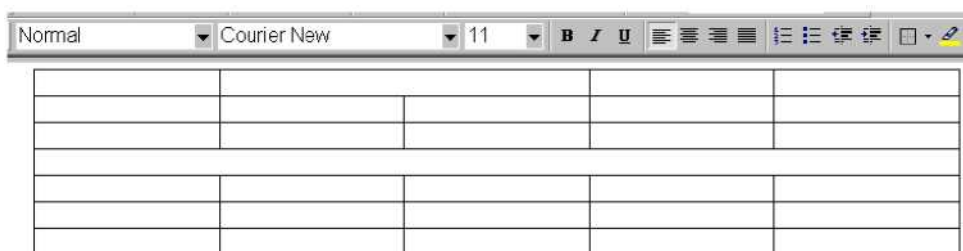


Рисунок 6. Нестандартная таблица. Таблицы не обязательно должны иметь одинаковое количество столбцов в каждой строке. Методы Merge и Split позволяют создавать таблицы неправильной формы.

Функция Split берет ячейку или коллекцию ячеек и делит ее на одну или несколько ячеек. При желании она может объединить ячейки перед их разделением. Опять же, существует две разные формы метода, в зависимости от того, вызываете ли вы его из одной ячейки или из коллекции ячеек:

```
oCell.Split(nRows, nColumns)
oCells.Split(nRows, nColumns [, lMergeFirst])
```

Например, чтобы разделить первую ячейку в строке 3 на две ячейки в одной строке, используйте следующую команду:

```
oTable.Cell(3,1).Split(1,2)
```

Следующий код берет ячейки второй строки таблицы, объединяет их, а затем разбивает на три, в результате чего в этой строке остается всего три ячейки:

```
oTable.Rows[2].Cells.Split( 1, 3, .t.)
```

Если вы опустите третий параметр этого вызова (.t.), каждая ячейка в строке будет разделена на три. Если вы передадите в качестве первого параметра значение, отличное от 1, одна строка станет несколькими строками в таблице.

Используя Merge и Split, вы можете создавать чрезвычайно сложные таблицы. Хотя это обеспечивает привлекательный способ отображения данных, имейте в виду, что это усложняет обработку документа. Простые циклы FOR EACH по коллекциям Rows и Columns не работают, если Uniform имеет значение .F.

5.3 Создание документов с несколькими колонками

Возможно, таблицы излишни для того, что вам нужно. Возможно, вам просто нужно разложить текст в несколько столбцов. В интерактивном режиме это можно сделать через меню «Формат», выбрав «Колонки». Откроется диалоговое окно «Колонки», показанное на рисунке 7. Вы можете указать количество колонок, ширину каждой и расстояние между ними.

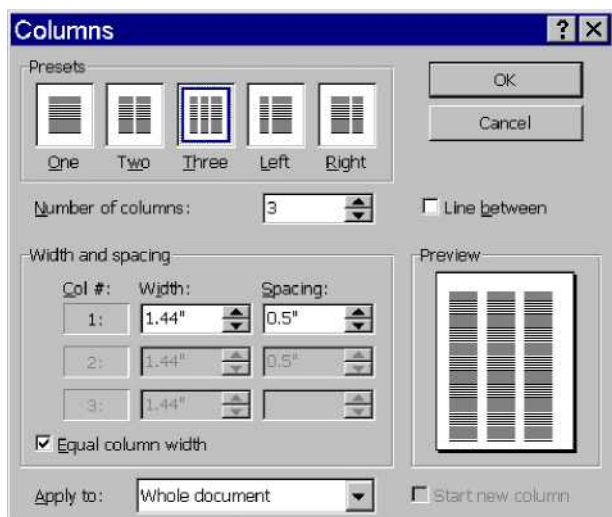


Рисунок 7. Указание колонок. В этом диалоговом окне Word вы можете указать, сколько колонок вы хотите, их ширину и расстояние между ними.

В Автоматизации колонки определяются коллекцией TextColumns, которая является членом объекта PageSetup. (Подробнее об объекте PageSetup см. раздел «Настройка страниц» в главе 4.) Помимо обычных свойств коллекции, TextColumns имеет несколько свойств, специфичных для её роли, показанных в таблице 5.

Таблица 5. Настройка столбцов. Коллекция TextColumns содержит объекты TextColumn, но у нее есть несколько свойств, применимых к коллекции в целом.

Свойство	Тип	Описание
EvenlySpaced	Logical or Numeric	Указывает, равномерно ли расположены столбцы.
Width	Numeric	Ширина столбцов в пунктах, если они расположены равномерно.
Spacing	Numeric	Расстояние между столбцами в пунктах, если они расположены равномерно.
LineBetween	Logical	Указывает, отображается ли вертикальная линия между столбцами. Не отображается в обычном режиме; для их отображения используйте представления «Макет печати» или «Предварительный просмотр». Кроме того, строки отображаются только в том случае, если в столбцах содержится текст.

Как и в других коллекциях, метод Add добавляет новые столбцы. Но у TextColumns есть специальный метод SetCount, который можно использовать для установки количества столбцов. Думайте о SetCount как о методе, требующем наименьших затрат, позволяющем избежать повторного вызова Add.

Объект TextColumn немного проще коллекции TextColumns. Стоит отметить только два свойства: Width и SpaceAfter. Ширина здесь играет ту же роль, что и для коллекции, за исключением того, что она влияет только на один столбец. SpaceAfter соответствует свойству Spacing коллекции. Внимание: установка Width или SpaceAfter для любого столбца приводит к изменению значения Width и Spacing для коллекции на wdUndefined (9999999), а для свойства EvenlySpaced коллекции устанавливается значение .F., даже если назначенное вами значение совпадает со

значением уровня коллекции. Чтобы восстановить значения уровня коллекции, задайте для EvenlySpaced значение .T. для коллекции.

Вот код для создания нового документа с тремя столбцами: первые два расположены на равном расстоянии друг от друга, третий – шире, с вертикальной линией между ними:

```
oDocument = oWord.Documents.Add()
WITH oDocument.PageSetup
    .TextColumns.SetCount(3)
    .TextColumns[1].Width = oWord.InchesToPoints(1.5)
    .TextColumns[1].SpaceAfter = oWord.InchesToPoints(.25)
    .TextColumns[2].Width = oWord.InchesToPoints(1.5)
    .TextColumns[2].SpaceAfter = oWord.InchesToPoints(.25)
    .TextColumns[3].Width = oWord.InchesToPoints(2.5)
    .TextColumns.LineBetween = .T.
    && Введите текст, чтобы увидеть строки.
ENDWITH
```

При отправке кода для заполнения столбцов вы перемещаетесь от одного столбца к другому с помощью метода InsertBreak. Этот метод принимает один параметр, указывающий тип разрыва. Существует набор констант для разрывов столбца, это wdColumnBreak со значением 8. Итак, чтобы перейти к следующему столбцу, используйте это:

```
#DEFINE wdColumnBreak 8
oRange.InsertBreak( wdColumnBreak )
```

Обязательно сверните диапазон (см. раздел «Перемещение в диапазоне или выделение» в главе 4) перед вставкой разрыва столбца; в противном случае разрыв столбца заменяет диапазон.

5.4 Сноски и концевые сноски

Сноски и концевые сноски были проклятием студентов на протяжении нескольких поколений, но Word позволяет достаточно легко ими управлять. Чтобы добавить сноску или примечание в интерактивном режиме, выберите в меню «Вставка|Примечание». Появится диалоговое окно, показанное на рисунке 8. Как указано в диалоговом окне, по умолчанию сноски размещаются внизу текущей страницы (или, если места недостаточно, внизу следующей страницы). По умолчанию концевые сноски появляются в конце документа, но вместо этого они могут появляться в конце каждого раздела.

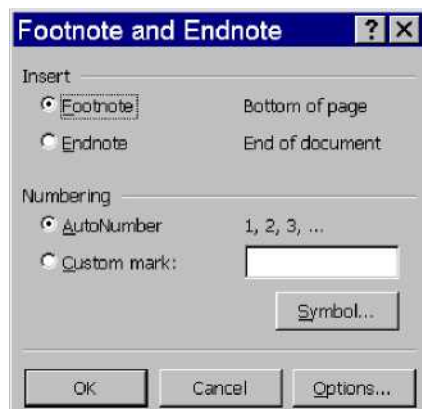


Рисунок 8. Добавление сносок. С помощью этого диалогового окна добавляются как сноски, так и концевые сноски. Оно также позволяет указать нумерацию и

размещение.

В рамках Автоматизации вы работаете с двумя коллекциями: Footnotes и Endnotes, а также содержащимися в них объектами Footnote и Endnote. Document и Range имеют свойства Footnotes и Endnotes, которые обеспечивают доступ к двум коллекциям. Хотя доступ к коллекциям через диапазон позволяет просматривать только сноски или концевые сноски в этом диапазоне, будьте осторожны. Изменение свойств коллекции влияет на все сноски или концевые сноски в документе, а не только на те, которые находятся в этом диапазоне. Диапазон просто включает в себя подмножество заметок коллекции.

За одним исключением, набор свойств и методов в объектах Footnote и Endnote идентичен, как и набор в коллекциях Footnotes и Endnotes. Вы можете думать о каждом случае как о подклассе одного и того же родительского класса.

Единственная разница между ними заключается в способе их рендеринга. Фактически, можно преобразовать сноски в концевые сноски, а концевые сноски в сноски. Несколько методов поддерживают этот переключатель.

Единственное свойство Footnote и Endnote, о котором стоит упомянуть, — это Reference. Оно предоставляет ссылку на объект, являющийся диапазоном, который содержит число, букву или другой символ в основном документе, указывающий на примечание. Вы можете использовать его для внесения изменений в этот символ, например увеличения размера шрифта или подчеркивания. (Являются ли такие изменения хорошей идеей, это отдельный вопрос.)

Таблица 6 показывает важные свойства коллекций Footnotes и Endnotes, а их важные методы перечислены в таблице 7.

Таблица 6. Указание сносок и концевых сносок. Обе коллекции структурно идентичны, только их визуализация различна.

Свойство	Тип	Описание
Location	Numeric	Указывает, где появляются примечания. Для сносок используйте одну из этих констант: <code>wdBottomOfPage 0 wdBeneathText 1</code> Для концевых сносок используйте одну из этих констант: <code>wdEndOfDocument 1 wdEndOfSection 0</code>
NumberingRule	Numeric	Указывает, как нумеруются примечания относительно разрывов страниц и разделов. <code>wdRestartContinuous 0</code> Номер непрерывно. <code>wdRestartSection 1</code> Перезапуск номера при каждом разрыве раздела. <code>wdRestartPage 2</code> Перезапуск номера на каждой новой странице.
NumberStyle	Numeric	Тип нумерации, используемый для примечаний. В Word 2000 предусмотрено около 20 стилей, но лишь некоторые из них

		отображаются в диалоговом окне «Сноски». Другие можно настроить с помощью кода, но не все они отображаются правильно во всех версиях Word. <table><tr><td>wdNoteNumberStyleArabic</td><td>0</td></tr><tr><td>wdNoteNumberStyleLowercaseLetter</td><td>4</td></tr><tr><td>wdNoteNumberStyleUppercaseLetter</td><td>3</td></tr><tr><td>wdNoteNumberStyleLowercaseRoman</td><td>2</td></tr><tr><td>wdNoteNumberStyleUppercaseRoman</td><td>1</td></tr><tr><td>wdNoteNumberStyleSymbol</td><td>9</td></tr><tr><td>wdNoteNumberStyleHebrewLetter1</td><td>45</td></tr></table>	wdNoteNumberStyleArabic	0	wdNoteNumberStyleLowercaseLetter	4	wdNoteNumberStyleUppercaseLetter	3	wdNoteNumberStyleLowercaseRoman	2	wdNoteNumberStyleUppercaseRoman	1	wdNoteNumberStyleSymbol	9	wdNoteNumberStyleHebrewLetter1	45
wdNoteNumberStyleArabic	0															
wdNoteNumberStyleLowercaseLetter	4															
wdNoteNumberStyleUppercaseLetter	3															
wdNoteNumberStyleLowercaseRoman	2															
wdNoteNumberStyleUppercaseRoman	1															
wdNoteNumberStyleSymbol	9															
wdNoteNumberStyleHebrewLetter1	45															
StartingNumber	Numeric	Номер первого примечания.														
Separator	Object	Указатель на диапазон, содержащий разделитель между основной частью документа и примечаниями. Разделителем обычно является короткая линия, но вы можете добавить к ней текст, удалить линию, добавить границы и даже добавить графику.														
ContinuationSeparatorText	Object	Указатель на диапазон, содержащий разделитель между основной частью документа и продолжением примечаний. Появляется, когда примечания переходят на другую страницу. Обычно это просто короткая линия, но, как и в случае с разделителем, вы можете удалить линию, добавить текст, границы и графику.														
ContinuationNotice	Object	Указатель на диапазон, содержащий сообщение, которое отображается, когда примечания переходят на другую страницу. Данное сообщение отображается в конце заполненной страницы. Обычно это пустое поле, но это не обязательно. Как и другие, вы можете добавлять текст, границы, графику и т. д.														

Таблица 7. Работа со сносками и примечаниями. Методы уровня коллекции позволяют управлять свойствами, отделяющими примечания от основной части документа, и изменять типы примечаний.

Метод	Описание
ResetSeparator	Восстанавливает разделитель по умолчанию.
ResetContinuationSeparator	Восстанавливает разделитель продолжения по умолчанию.
ResetContinuationNotice	Восстанавливает уведомление о продолжении по умолчанию.

Convert	Преобразует все примечания в указанном диапазоне или документе в другой тип. То есть, если метод вызывается из коллекции Footnotes, он преобразует сноски в концевые сноски и добавляет их в коллекцию Endnotes документа; если он вызывается из коллекции Endnotes, он преобразует их в сноски и добавляет в коллекцию Footnotes. В любом случае они корректно перемежаются существующими заметками другого типа.
SwapWithEndnotes	Преобразует все сноски в концевые сноски и концевые сноски в сноски. Этот метод необходимо вызывать из коллекции Footnotes.

Как и в других коллекциях, ключом к добавлению новых членов является метод Add. Для сносок и концевых сносок требуются три параметра:

```
oNotes.Add( oRange [, oReferenceMark [, cText ] ] )
```

oRange	Object	Диапазон, в который вставляется номер сноски или символ.
oReferenceMark	Object	Диапазон, содержащий пользовательский символ, который будет использоваться для обозначения сноски или концевые сноски. Если этот параметр опущен, заметка нумеруется в соответствии с правилом, установленным NumberingRule и StartingNumber.
cText	Character	Текст сноски или концевой сноски.

Этот код создает диапазон из первого предложения существующего диапазона и добавляет к нему сноску.

```
#DEFINE wdCharacter 1
```

```
oNoteRange = oRange.Sentences[1]
```

```
* Создайте резервную копию одного символа, чтобы исключить  
* завершающий пробел
```

```
oNoteRange.MoveEnd( wdCharacter, -1 )
```

```
* Разрешить автоматическую нумерацию заметок
```

```
oRange.Endnotes.Add( oNoteRange, , "Это новая концевая сноска")
```

В следующем примере добавляется сноска, которой присваивается специальный маркер звездочка (*) вместо нумерации по умолчанию. Заметка прикрепляется к только что вставленному тексту.

```
#DEFINE wdCollapseEnd 0
```

```
oRange.InsertAfter("Обязательно ознакомьтесь с WhirlyGig 2000!")
```

```
oRange.Collapse( wdCollapseEnd )
```

```
oDocument.Footnotes.Add( oRange, "*", ;
```

```
    "WhirlyGig 2000 является торговой маркой MonsterMega "+;
```

```
    "Corporation, Ltd.")
```

5.5 Добавление графики в документы

Говорят, картинка стоит тысячи слов. Конечно, документ с картинками гораздо приятнее смотреть и легче читать, чем документ без изображений. Word предоставляет несколько способов добавления графических объектов в документы. Все они начинаются с выбора в меню «Вставка|Рисунок». На этом этапе следующий шаг зависит от типа графического объекта, который вы хотите добавить.

Давайте начнем с графики, которая хранится где-то в файле, например BMP, JPG или GIF. В этом случае вы выбираете «Из файла», и появляется диалоговое окно «Вставить изображение», показанное на рисунке 9. Это диалоговое окно представляет собой специализированную версию обычного диалогового окна «Открыть файл». Кнопка «Открыть» называется «Вставить», и если вы отпустите ее, появится несколько дополнительных вариантов. Эти параметры возвращаются к традиционному значению *связывания и внедрения объектов OLE*. При выборе «Вставить» изображение в документе копируется без связи с исходным файлом; то есть он встраивает изображение в документ. Ссылка на файл создает связь между файлом изображения и документом; то есть он связывает изображение с документом. Изменения изображения отражаются в документе. Команды «Вставка» и «Ссылка» одновременно встраивают изображение в документ, но также сохраняют ссылку на файл, так что документ обновляется за счет изменений в файле изображения. Каждый из этих параметров имеет свое место. Выберите тот, который подходит для вашей ситуации, исходя из таких факторов, как ожидаемое ли вами изменение изображения, хотите ли вы, чтобы изменения в изображении отражались в вашем документе, размер файла изображения, можно ли доверять пользователям, чтобы они не удалили картинку и так далее.

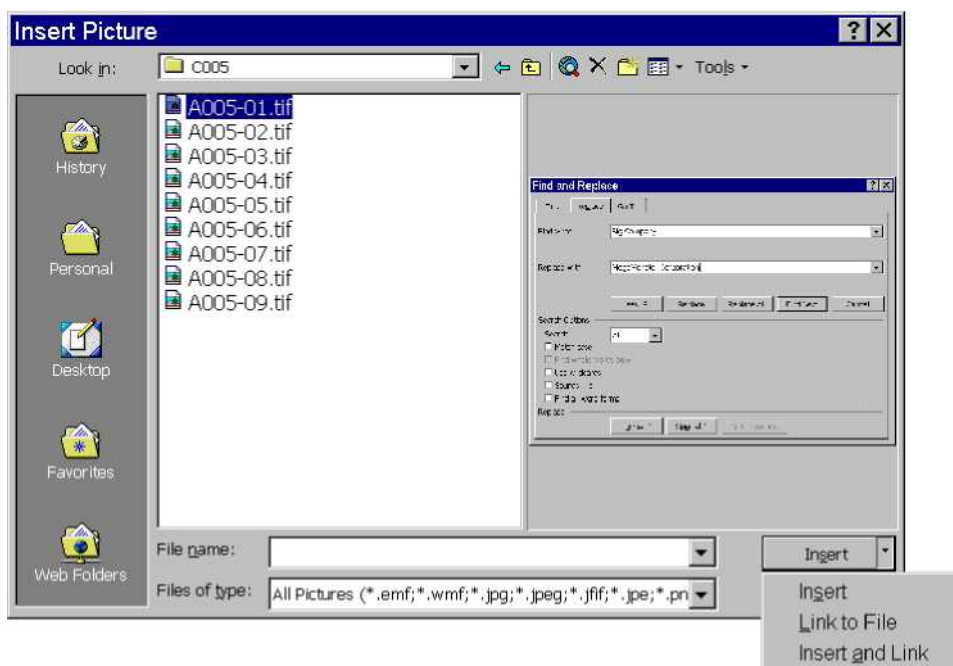


Рисунок 9. Добавление картинок. Графика может быть связана, встроена или и то, и другое. Когда изображение связано, изменения в графическом файле отражаются в документе.

Список графических объектов, которые вы можете добавить, также включает в себя различные виды рисунков, такие как автофигуры и WordArt. Чтобы добавить автофигуру в документ, выберите в меню «Вставка|Рисунок|Автофигуры». Откроется панель инструментов «Автофигуры» (показана на рисунке 10). При нажатии на одну из кнопок появляется меню фигур данного типа (см. рис. 11).

Выберите одну, и Word при необходимости переключится в режим макета страницы и поместит выбранную фигуру туда, куда вы щелкнете. Панель инструментов «Автофигуры» остается открытой, поэтому вы можете добавить столько фигур, сколько захотите.



Рисунок 10. Панель инструментов «Автофигуры». Эта панель инструментов появляется всякий раз, когда вы выбираете в меню «Вставка|Рисунок|Автофигуры».

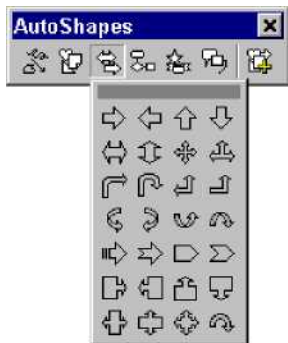


Рисунок 11. Выбор автофигуры. Элементы на панели инструментов «Автофигуры» выглядят как кнопки, но каждый из них открывает меню элементов фигур.

К каждому из остальных видов графических объектов свой подход. Ни один из них не является родным для Word; все они обращаются к другим приложениям (таким как Microsoft Clip Gallery и Microsoft Graph) для предоставления отображаемых данных.

Два лица графики

Графику в Word можно разместить на двух разных слоях документа. Изображения, полученные из файлов, обычно располагаются на том же слое, что и текст документа. Автофигуры и другие объекты рисования располагаются как свободно плавающие объекты на отдельном слое над текстом. Это означает, что они могут покрывать текст и графику основного слоя. Для управления двумя типами графических объектов используются две разные коллекции. Объекты текстового слоя хранятся в коллекции InlineShapes, а объекты, плавающие над текстовым слоем, контролируются коллекцией Shapes. Каждый из них содержит соответствующий тип объекта: InlineShape и Shape соответственно.

Работа с InlineShapes

Вместо одного метода Add в коллекции InlineShapes имеется несколько различных методов для добавления объектов разных типов.

Вероятно, наиболее важным из методов Add для коллекции InlineShapes является AddPicture, который позволяет добавлять изображение из файла. Это программный эквивалент диалогового окна «Вставка изображения», показанного на рисунке 9.

Синтаксис AddPicture следующий:

```
oDocument.InlineShapes.AddPicture( cPictureFile, lLinkToFile,
                                     lSaveWithDocument, oRange )
```

cPictureFile	Character	Имя, включая путь, файла, содержащего изображение, добавляемое в документ.
lLinkToFile	Logical	Указывает, должна ли ссылка на исходное поле сохраняться в терминах OLE, связано ли

изображение.

ISaveWithDocument	Logical	Указывает, сохранено ли изображение в документе или связано только в терминах OLE, внедрено ли изображение.
oRange	Object	Указатель на место, куда следует вставить картинку.

Например, чтобы добавить растровое изображение TasTrade в начало документа, используйте такой код:

```
oRange = oDocument.Paragraphs[1].Range()
* Добавьте изображение, не связывайте его, вставьте
oDocument.InlineShapes.AddPicture( ;
    _SAMPLES + "TASTRADE\BITMAPS\TTRADESM.BMP ", ;
    .F., .T., oRange )
```

Результаты для нового документа показаны на рисунке 12.

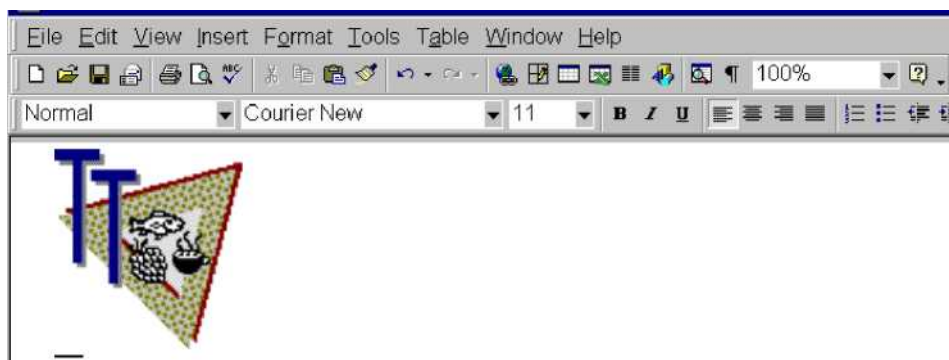


Рисунок 12. Добавление графики. Чтобы добавить изображение с помощью Автоматизации, вы указываете имя файла, следует ли связать его, встроить ли и куда его поместить.

Другие методы добавляют в коллекцию другие типы объектов, например горизонтальные линии, маркеры, созданные из изображений, и объекты OLE. Подробности смотрите в файле справки.

Сам объект InlineShape является контейнером для изображения или другого объекта. Он не содержит фактической графики или объекта OLE, но, как и элемент управления OLE-контейнер VFP, он предоставляет ему место, где можно повесить свою шляпу. InlineShape содержит информацию о формате объекта, который он представляет, а также о типе этого объекта. Доступная информация различается в зависимости от типа объекта и от того, связан ли он или встроен. В таблице 8 перечислены ключевые свойства InlineShape и значения констант, необходимые для добавления графики.

Таблица 8. Добавление графики и других объектов на текстовый слой. Объект InlineShape содержит формат и другую информацию об объектах, которые перемещаются вместе с текстом документа.

Свойство	Тип	Описание
Type	Numeric	Вид графического или другого объекта.
		wdInlineShapePicture 3 Не связано
		wdInlineShapeLinkedPicture 4 Связано

Height, Width	Numeric	Размер встроенной фигуры в пунктах. Это фактическое пространство, используемое для фигуры в документе, оно не связано с размером базовой графики. LockAspectRatio определяет, можно ли их изменить независимо.
ScaleHeight, ScaleWidth	Numeric	Процент исходной высоты и ширины, на котором должен отображаться объект. Когда они изменяются, высота и ширина также изменяются. LockAspectRatio определяет, могут ли они изменяться независимо.
LockAspectRatio	Logical	Указывает, должны ли высота и ширина оставаться в одинаковых относительных пропорциях. Когда .T., изменение высоты также изменяет ширину, а изменение ScaleHeight также изменяет ScaleWidth.
PictureFormat	Object	Для объектов изображения ссылка на объект PictureFormat, который управляет элементами, управляемыми в интерактивном режиме с помощью пункта «Формат изображения» в меню Word (либо «Формат Изображение» из главного меню, либо «Формат изображения» из контекстного меню для изображения). К ним относятся обрезка, яркость и контрастность. Дополнительную информацию см. в справке.
LinkFormat	Object	Для связанных объектов ссылка на объект LinkFormat, который содержит информацию о связанном объекте, включая имя файла и тип объекта. Дополнительную информацию см. в справке.
Borders	Object	Ссылка на коллекцию Borders для InlineShape.

Существует ряд других свойств, многие из которых специфичны для определенных типов объектов или ситуаций.

При обработке коллекции InlineShapes обязательно проверьте Type, прежде чем предполагать, что такие свойства, как PictureFormat или LinkFormat, доступны. Разные типы объектов используют разные подмножества свойств, и доступ к свойствам, которые не используются, является хорошим способом вызвать ошибку.

Работа с фигурами

Как и InlineShapes, коллекция Shapes не имеет метода, называемого просто Add. Она имеет ряд специфических методов для добавления различных типов объектов. У Shapes есть собственная версия AddPicture с несколько другим списком параметров:

```
oDocument.Shapes.AddPicture( cPictureFile, lLink,
lSaveWithDocument, nLeft, nTop, nWidth, nHeight, oAnchor)
```

nLeft, nTop Numeric Положение изображения относительно объекта, на который ссылается oAnchor. Оба значения по умолчанию равны 0. (Необязательно)

nWidth, nHeight	Numeric	Размер изображения. По умолчанию соответствует реальному размеру изображения. (Необязательно)
oAnchor	Object	Объект Range, указывающий, где изображение должно быть «прикреплено» к документу.

Остальные параметры здесь имеют то же значение, что и для метода AddPicture коллекции InlineShapes.



Параметр привязки для Shapes.AddPicture – это позиция в документе, из которой изображение может перемещаться. Это дает Word возможность перемещать изображение и размещать его там, где для него есть место. Кроме того, изображения, добавленные с помощью фигур, могут быть окружены текстом различными способами. Изображения InlineShapes не могут этого сделать, поскольку они всегда занимают всю ширину страницы или столбца, даже если при этом остается значительное свободное пространство.

У использования фигур для добавления графики есть несколько недостатков. Первое – незначительное. Графика не отображается в обычном режиме Word; вам нужно переключиться на режим печати или веб-макет (в Word 97 используйте режим «Макет страницы»), чтобы их увидеть. Это может ввести в заблуждение при редактировании документа, но для Автоматизации это не проблема.

Другая проблема более серьезна. Поскольку Word *может перемещать эту графику, он это делает. Контролировать, где появляется изображение, добавленное через коллекцию фигур, сложно. При изменении документа изображения могут перемещаться по документу.*

Выбор правильного способа добавления графики требует оценки плюсов и минусов каждого подхода. Разные документы могут требовать разных вариантов.

Метод AddShape позволяет добавлять автофигуры. Вот синтаксис:

```
oDocument.Shapes.AddShape( nShape, nLeft, nTop, nWidth, nHeight,
oAnchor)
```

Параметры такие же, как и для метода AddPicture коллекции Shapes, за исключением дополнительного параметра nShape. Это числовое значение, указывающее, какую автофигуру следует добавить. Как и следовало ожидать, для них доступен набор констант. Чтобы просмотреть список, найдите AutoShapeType в справке или, чтобы получить более полезный список, просмотрите константы msoAutoShapeType в Обзорщике объектов. Список соответствует фигурам, доступным на панели инструментов «Автофигуры».

Shapes поддерживает ряд других объектов и имеет для них соответствующие методы Add. Подробности смотрите в справке.

Объект Shape является аналогом InlineShape и содержит информацию об изображении или другом объекте после его добавления в документ. Shape имеет некоторые из тех же свойств, что и InlineShape, но также имеет дополнительные свойства, которые решают конкретные проблемы плавающей графики.

Как и везде, Type определяет тип элемента, который содержит Shape. Однако он ссылается на другой набор констант. В данном случае они исходят из самого Office. В Таблице 9 перечислены некоторые из разрешенных значений.

Таблица 9. Какая у меня фигура? Свойство Type объекта Shape принимает значения из набора констант Office msoShapeType.

Константа	Значение	Константа	Значение
msoPicture	13	msoAutoShape	1
msoLinkedPicture	11	msoChart	3

Height, Width, PictureFormat и LinkFormat для Shape такие же, как и для InlineShape. ScaleHeight и ScaleWidth здесь являются методами, а не свойствами (параметры см. в справке).

В этом примере растровое изображение TasTrade добавляется в виде плавающего изображения, прикрепленного к первому абзацу документа:

```
oRange = oDocument.Paragraphs[1].Range()
* Добавьте изображение, не ссылайтесь, вставьте.
* Если размер не указан, по умолчанию используется размер
изображения.
oDocument.Shapes.AddPicture( ;
    _SAMPLES + "TASTRADE\BITMAPS\TTRADESM.BMP ", ;
    .F., .T., 100, 100, , , oRange )
```

Чтобы увидеть это изображение, вы должны находиться в режиме просмотра, отличном от «Обычного». Установите для свойства Type объекта View ActiveWindow документа значение wdPrintView (3) или wdWebView (6). Чтобы вернуться к обычному виду, установите для свойства значение wdNormalView (1).

```
#DEFINE wdPrintView 3
oDocument.ActiveWindow.View.Type = wdPrintView
```

С помощью объектов Shape вы можете решить, будет ли текст обтекать изображение, и если да, то как будет работать обтекание. Эту возможность контролируют несколько свойств и методов. Свойство WrapFormat ссылается на объект WrapFormat. Свойство Type этого объекта предлагает первый уровень настроек текста, заверстанного вокруг иллюстрации. Установите для него значение wdWrapNone (3), чтобы предотвратить обтекание текста вокруг изображения; в этом случае изображение появляется перед текстом. По умолчанию используется wdWrapSquare (0), который рисует воображаемый квадрат вокруг изображения (независимо от его фактической формы) и помещает текст до краев этого квадрата. Есть также несколько других вариантов.

Свойство Side WrapFormat определяет, отображается ли текст на обеих сторонах изображения, если это возможно. Возможные значения: wdWrapBoth (0), wdWrapLargest (3), wdWrapLeft (1), wdWrapRight (2). Самый сложный вариант здесь — разрешить тексту и графике занимать одно и то же пространство, но сделать их видимыми. На рисунке 13 показана автофигура с наложенным текстом.



Рисунок 13. Объединение графики и текста. Управляя методом ZOrder объекта Shape, вы можете поместить текст сверху.

Секрет создания такого эффекта заключается в вызове метода `ZOrder` класса `Shape` и передаче `msoSendBehindText` (5) в качестве параметра. `ZOrder` имеет больше возможностей для формы, чем в Visual FoxPro. Помимо перемещения объекта на передний план (`msoBringToFront` или 0) или на задний план (`msoSendToBack` или 1), вы можете отправить его за текстом, как показано на рисунке 13, или поместить его перед текстом (`msoBringInFrontOfText` или 4). и еще пара вариантов.

Вот код для создания рисунка 13 в начале документа:

```
#DEFINE msoShape16PointStar 94
#DEFINE msoSendBehindText 5
#DEFINE wdWrapNone 3
#DEFINE wdWrapBoth 0
#DEFINE CR CHR(13)

LOCAL oDocument, oRange, oShape

oDocument = oWord.Documents.Add()
oRange = oDocument.Range()

* Добавьте текст
oRange.Font.Name = "Calligrapher" && или выберите свой любимый
oRange.Font.Size = 18
oRange.InsertAfter( CR + "    Gold" + CR + "    Medal")

* Теперь добавим форму
oShape = oDocument.Shapes.AddShape( msoShape16PointStar, ;
                                     0, 0, 81, 81, oRange )

oShape.WrapFormat.Type = wdWrapNone
oShape.WrapFormat.Side = wdWrapBoth
oShape.ZOrder( msoSendBehindText )
```

5.6 Работа с существующими документами.

После того как вы создали документы, у вас может возникнуть желание просмотреть их, чтобы внести изменения или просто просмотреть их содержимое, по разным причинам. Word содержит ряд инструментов, которые позволяют легко просмотреть документ и заглянуть внутрь. Можно изучить как содержимое документа, так и его структуру.

Поиск и замена

Поиск и замена – одна из убийственных функций, которые обеспечили успех текстовым процессорам с момента их первого появления. Умение просматривать документ и менять каждое вхождение «Большая компания» на «Корпорация МегаМонстр» всего несколькими нажатиями клавиш подружило секретарей всего цивилизованного мира. Word развивает эту способность гораздо дальше.

Вы можете искать либо текст, либо форматирование, либо их комбинацию. Поиск может осуществляться с учетом или без учета регистра. Вы можете выполнить поиск только для того, чтобы найти искомую фразу, или найти ее и заменить другой фразой. При замене вы можете заменить все вхождения или проверить каждое вхождение по мере его обнаружения и заменить его или нет. На рисунке 14 показано диалоговое окно «Найти и заменить», доступное через меню «Правка» в Word.

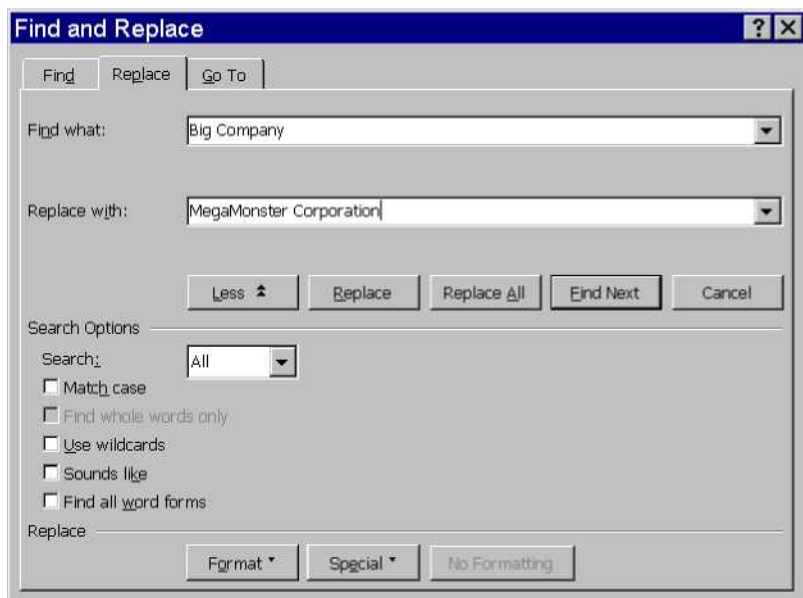


Рисунок 14. Найти и заменить. Версия поиска и замены Word чрезвычайно эффективна. Вы можете искать только текст, только форматирование или текст и форматирование. Вы можете заменить все экземпляры или проверить их по одному.

Поиск обрабатывается объектом Find, доступ к которому осуществляется через свойство Find объекта Range. Он имеет набор свойств, таких как Text, MatchCase и MatchWholeWord, которые определяют, что он ищет. Некоторые свойства, например Font и ParagraphFormat, являются ссылками на другие объекты. Свойство Format определяет, учитывается ли форматирование при поиске. Установите его на .F. для поиска строки, указанной в Text, независимо от формата. В таблице 10 перечислены некоторые часто используемые свойства Find.

Метод Execute фактически выполняет поиск. В своей простейшей форме он использует текущие настройки объекта Find. Он также может принимать ряд параметров, большинство из которых дублируют свойства объекта Find. Подробнее об этом позже.

Таблица 10. Настройка поиска. Свойства объекта Найти определяют, что искать.

Свойство	Тип	Описание
Text	Character	Строка для поиска. Если оно пусто и формат равен .T., поиск основан только на форматировании.
MatchCase	Logical	Указывает, чувствителен ли поиск к регистру. По умолчанию .F. (без учета регистра).
MatchWholeWord	Logical	Указывает, ищет ли поиск только целые совпадения слов. По умолчанию .F. (совпадения внутри слов).
MatchSoundsLike	Logical	Указывает, ищет ли поиск совпадения, похожие на строку поиска. По умолчанию .F. (только реальные совпадения).
MatchWildcards	Logical	Указывает, содержит ли строка поиска подстановочные знаки, которые должны сопоставляться соответствующим образом. Это невероятно мощный параметр, который, по

		нашему мнению, недостаточно используется пользователями Word (включая нас самих). По умолчанию .F. (без подстановочных знаков).
MatchAllWordForms	Logical	Указывает, следует ли заменять другие формы строки поиска соответствующими формами строки замены. Например, если строки поиска и замены представляют собой существительные в единственном числе, а этот параметр равен .T., версии строки поиска во множественном числе будут заменены версиями строки замены во множественном числе. По умолчанию .F. (сопоставлять строку только так, как указано).
Format	Logical	Указывает, включать ли форматирование в поиск. Если .F., игнорируйте при поиске свойства этого объекта, связанные с форматом.
Font	Object	Ссылка на объект Font, указывающий соответствующий шрифт, если Format – .T.
ParagraphFormat	Object	Ссылка на объект ParagraphFormat, указывающий форматирование абзаца, которое соответствует формату .T.
Style	Object	Ссылка на объект Style, указывающий соответствующий стиль, если Format – .T.
Forward	Logical	Указывает направление поиска. Установить на .T. для поиска ближе к концу документа, .F. искать ближе к началу.
Wrap	Numeric	Указывает, что происходит, когда поиск начинается в середине документа и достигает конца. wdFindStop 0 Прекратите поиск. wdFindContinue 1 Продолжайте поиск на другом конце документа. wdFindAsk 2 Спросите пользователя, что делать.
Found	Logical	Указывает результат последнего поиска.
Replacement	Object	Ссылка на замещающий объект с информацией для замены. (Смотрите текст.)

Как отмечалось ранее, Find является членом Range (и Selection), но не Document. Поиск начинается с начала диапазона, если только Forward не равен .F., в этом случае поиск начинается с конца. Свойство Wrap определяет, что произойдет, когда будет достигнут конец (или начало при поиске назад) диапазона.

Для поиска по всему документу создайте диапазон в начале документа. Когда Execute находит совпадение, границы диапазона изменяются, чтобы охватывать только соответствующий элемент.

Например, для поиска первого вхождения строки «Visual FoxPro» независимо от регистра в текущем документе используйте следующий код:

```
oRange = oWord.ActiveDocument.Range(0,0) && Начало документа
WITH oRange.Find
    .Text      = "Visual FoxPro"
    .MatchCase = .F.
    .Format    = .F.
    lFound     = .Execute()
ENDWITH
```

Когда этот код будет выполнен, если lFound равен .T., совпадение было найдено, а oRange содержит соответствующий текст.

Все становится немного сложнее, когда вы хотите заменить найденный текст. Есть два способа сделать это. Последние два параметра метода Find's Execute позволяют указать строку замены и количество выполняемых замен (ни одной, одной или всех). Вот синтаксис выполнения:

```
oRange.Execute( cText, lMatchCase, lMatchWholeWord,
                lMatchWildcards, lMatchSoundsLike,
                lMatchAllWordForms, lForward, nWrap,
                lFormat, cReplaceWith, nReplace)
```

cText	Character	Соответствует свойству Text.
lMatchCase	Logical	Соответствует свойству MatchCase.
lMatchWholeWord	Logical	Соответствует свойству MatchWholeWord.
lMatchWildcards	Logical	Соответствует свойству MatchWildcards.
lMatchSoundsLike	Logical	Соответствует свойству MatchSoundsLike.
lMatchAllWordForms	Logical	Соответствует свойству MatchAllWordForms.
lForward	Logical	Соответствует свойству Forward.
nWrap	Numeric	Соответствует свойству Wrap.
lFormat	Logical	Соответствует свойству Format.
cReplaceWith	Character	Строка замены.
nReplace	Numeric	Указывает, заменять ли строку поиска на cReplaceWith каждый раз, когда она найдена, только один раз или не заменять вообще.

Используйте эти константы:

```
wdReplaceNone  0
wdReplaceOne   1
wdReplaceAll   2
```

Технически cReplaceWith и nReplace на самом деле не являются двумя последними параметрами. Есть еще несколько вариантов, которые могут быть приняты, в зависимости от того, какой язык вы используете. Они касаются особенностей языка. Проверьте справку, если вы работаете на языке, отличном от английского.

Возвращаясь к предыдущему примеру, мы можем заменить каждый экземпляр «Visual FoxPro» на «Visual FoxPro!» изменив строку «Выполнить» на:

```
#DEFINE wdReplaceAll 2
lFound = .Execute( , , , , , , , , , "Visual FoxPro!", wdReplaceAll )
```

Альтернативный подход использует объект Replacement, на который ссылается свойство Replacement объекта Find. Replacement аналогична более простой версии

Find: она может форматировать свойство Text, среди прочего, с помощью свойств Font, ParagraphFormat и Style. Вы можете заполнить свойства Replacement, чтобы точно указать, что должно заменить найденный элемент, например:

```
#DEFINE wdReplaceAll 2
WITH oRange.Find
    * что искать
    .Text      = "Visual FoxPro"
    .MatchCase = .F.
    .Format    = .F.

    * на что заменить
    .Replacement.Text = "Visual FoxPro!"
    .Replacement.Font.Bold = .T.
    .Replacement.ParagraphFormat.LeftIndent = 12

    * выполнить!
    lFound = .Execute( , , , , , , , , , , wdReplaceAll )
ENDWITH
```

Также возможен поиск и замена только форматирования. И Find, и Replacement имеют метод ClearFormatting, который сбрасывает все свойства, связанные с форматом, к значениям по умолчанию, поэтому вы можете начать с нуля.

Чтобы изменить каждое вхождение Arial размером 12 пунктов на Times New Roman размером 16 пунктов, используйте такой код:

```
#DEFINE wdReplaceAll 2
oRange = oWord.ActiveDocument.Range(0,0)
WITH oRange.Find
    * обязательно очистите последний поиск
    .ClearFormatting

    * что искать
    .Text      = ""
    .Format    = .T.
    .Font.Name = "Arial"
    .Font.Size = 12

    * на что заменить
    WITH .Replacement
        .ClearFormatting
        .Text = ""
        .Font.Name = "Times New Roman"
        .Font.Size = 16
    ENDWITH

    * выполнить!
    lFound = .Execute( , , , , , , , , , , wdReplaceAll )
ENDWITH
```

Конечно, со стилями вам не придется часто менять шрифты, но искать и заменять стили тоже легко. Просто используйте свойство Style объектов Find и Replacement.

Проявив немного творчества, можно найти и заменить в документе практически все, что захотите. Вы также можете объединить возможности VFP по обработке данных

с VBA для более эффективного поиска. Представьте себе, что вы помещаете в таблицу набор строк поиска и замены, а затем используете Автоматизацию, чтобы внести все изменения без вмешательства.

Изучение структуры документа

Некоторые свойства и методы позволяют узнать о таких вещах, как количество слов и страниц в документе или диапазоне, автор документа, его заголовок, шаблон, на котором он основан, и т. д. Вы также можете предоставить часть этой информации и другие данные о документе.

Статистические данные, такие как количество слов и страниц, доступны в интерактивном режиме через пункт «Количество слов» в меню «Инструменты». При выборе этой опции отображается диалоговое окно, показанное на **рисунке 15**.

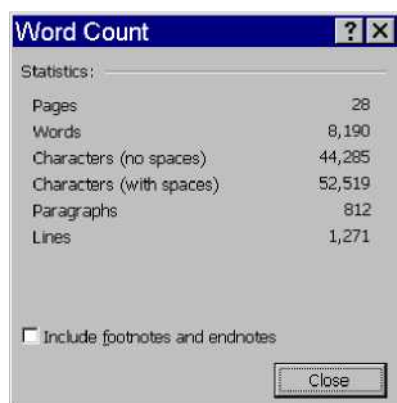


Рисунок 15. Это здесь. Диалоговое окно «Подсчет слов» фактически показывает различную статистику о документе или диапазоне. Та же информация доступна через метод ComputeStatistics.

С помощью Автоматизации вы получаете доступ к той же информации, используя метод ComputeStatistics объектов Document и Range. Он принимает параметр, указывающий, какую статистику вычислять, и возвращает соответствующее значение. В **таблице 11** показаны константы для элементов, которые вам, скорее всего, понадобится вычислить.

Таблица 11. Что вы хотите посчитать сегодня? Метод ComputeStatistics принимает одну из этих констант и сообщает, сколько указанных объектов находится в документе или диапазоне.

Константа	Значение	Константа	Значение
wdStatisticWords	0	wdStatisticCharacters	3
wdStatisticLines	1	wdStatisticParagraphs	4
wdStatisticPages	2	wdStatisticCharactersWithSpaces	5

Например, чтобы увидеть сколько слов в текущем документе, используйте:

```
#DEFINE wdStatisticWords 0
nWords = oWord.ActiveDocument.ComputeStatistics(wdStatisticWords)
```

Обратите внимание, что некоторые из этих значений также доступны, если проверить свойство Count соответствующей коллекции. Например, вы можете определить количество абзацев в диапазоне следующим образом:

```
nParagraphs = oRange.Paragraphs.Count
```

Будьте предупреждены. Вызов `ComputeStatistics` присваивает свойству `Saved` документа значение `.F.` Это верно, даже если вы не вносите никаких других изменений в документ, подсчитав количество слов, страниц или чего-то еще, достаточного, чтобы документ считал, что он был изменен.

Дополнительную информацию о документе можно получить в диалоговом окне «Свойства» меню «Файл». Это многостраничное диалоговое окно (показанное на рисунке 16) включает в себя часть той же информации, что и диалоговое окно «Подсчет слов», но оно также содержит такие свойства документа, как автор, заголовок, тема и многое другое. Помимо свойств, встроенных в диалоговое окно, вы можете указать пользовательские свойства и указать для них значения (с помощью страницы «Пользовательские»). Свойства файла могут отображаться в диалоге `Файл|Открыть` и использоваться для поиска документов.

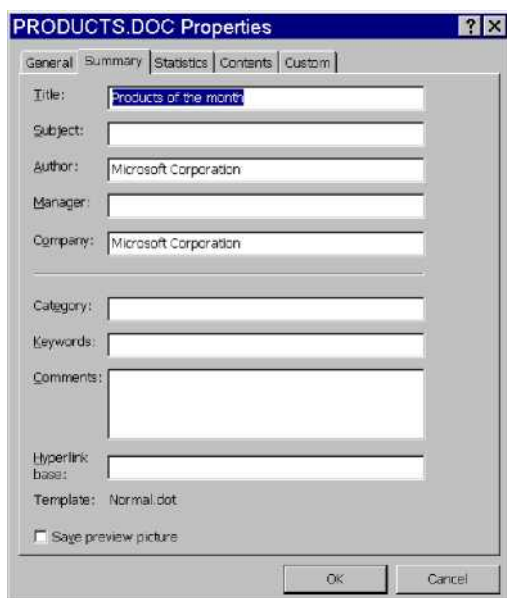


Рисунок 16. Свойства документа. В этом диалоговом окне отображается информация, хранящаяся в документе. В дополнение к предопределенным свойствам вы можете добавить свои собственные свойства.

Две коллекции `DocumentProperties` содержат все свойства. Доступ к ним осуществляется через два свойства `Document`: `BuiltinDocumentProperties` и `CustomDocumentProperties`. `DocumentProperties` и `DocumentProperty` – это объекты Office, а не объекты Word.

Чтобы проверить значение определенного свойства документа, вы можете найти его по имени, а затем проверить его свойство `Value`. Например, чтобы увидеть автора активного документа:

```
? oWord.ActiveDocument.BuiltinDocumentProperties[ "Author" ].Value
```

Список встроенных свойств см. в разделе справки об объекте `DocumentProperty`.

Чтобы добавить пользовательские свойства, используйте метод `Add` коллекции `DocumentProperties`. Вот синтаксис:

```
oDocument.CustomDocumentProperties.Add( cPropertyName, lLinked,
                                         nPropertyType, uValue )
```

<code>cPropertyName</code>	<code>Character</code>	Имя нового настраиваемого свойства.
<code>lLinked</code>	<code>Logical</code>	Указывает, связано ли новое настраиваемое свойство с содержимым документа. Передайте <code>.F.</code> , чтобы просто указать значение. Если <code>.T.</code> ,

необходимо передать необязательный пятый параметр (здесь не обсуждается).

nPropertyType	Numeric	Тип данных для нового настраиваемого свойства. Используйте одну из этих констант:
		msoPropertyTypeNumber 1
		msoPropertyTypeBoolean 2
		msoPropertyTypeDate 3
		msoPropertyTypeString 4
		msoPropertyTypeFloat 5
uValue	Зависит от nPropertyType	Значение для назначения нового настраиваемого свойства.

Этот код добавляет к документу свойство VFPVersion и устанавливает для него версию Visual FoxPro, использованную для создания документа:

```
#DEFINE msoPropertyTypeString 4
oDocument.CustomDocumentProperties.Add( "VFPVersion", .F., ;
    msoPropertyTypeString, VERSION() )
```

Прежде чем добавлять пользовательское свойство, внимательно проверьте список встроенных свойств. Их довольно много.

Обход документа с коллекциями

По документу можно «пройтись» разными способами, используя коллекции объекта Document вместе со структурой цикла FOR EACH VFP. Хотя это может быть медленно, для некоторых задач обход коллекции является лучшим подходом.

Мы не рекомендуем использовать коллекции в качестве замены Find. Поиск текста или форматирования выполняется намного быстрее благодаря встроенному механизму поиска Word. Не лучше также измерять размер документа, вручную подсчитывая его строки, абзацы или слова (хотя свойство Count коллекций может быть весьма информативным). Однако цикл по такой коллекции, как InlineShapes или Tables, не должен снижать эффективность.



Процедура в листинге 2 проверяет каждое изображение в документе и проверяет, является ли оно связанным, но не встроенным. Она создает курсор, в котором перечислены те, кто нарушает правило. Вы найдете его как FindGraphics.PRG в файлах загрузки для разработчиков, доступных по адресу www.hentzenwerke.com.

Листинг 2. Находим графику. Эта программа проходит через InlineShapes и Shapes, чтобы найти каждое изображение в активном документе и проверить, связано ли оно или встроено.

```
* FindGraphics.PRG
* © 2000, Tamar E. Granor and Della Martin
* From: Microsoft Office Automation with Visual FoxPro
* Hentzenwerke Publishing. www.hentzenwerke.com
```

```
#DEFINE wdInlineShapePicture 3
#DEFINE wdInlineShapeLinkedPicture 4
#DEFINE msoPicture 13
```



```

#DEFINE msoLinkedPicture          11

* Предполагается, что у нас есть Word и документ.

* Создайте файл журнала.
IF NOT USED("Graphics")
    CREATE CURSOR Graphics (cDocument C(100), cGraphic C(100), ;
        lLinked L, lEmbedded L, tWhen T)
ENDIF

* Теперь найдите графику.
WITH oWord.ActiveDocument
    * Сначала создайте графику в текстовом слое.
    FOR EACH oShape IN .InlineShapes
        DO CASE
            CASE oShape.Type = wdInlineShapeLinkedPicture
                * Связано, может быть встроено или не быть встроенным
                IF oShape.LinkFormat.SavePictureWithDocument
                    * Он встроен, поэтому зарегистрируйте его
                    INSERT INTO Graphics VALUES ;
                        (.FullName, oShape.LinkFormat.SourceFullName, ;
                            .T. , .T. , DATETIME())
                ENDIF

            CASE oShape.Type = wdInlineShapePicture
                * Встроенный!
                INSERT INTO Graphics VALUES (.FullName, "", .F. , .T., DATETIME())

            OTHERWISE
                && Не интересуюсь этими
            ENDCASE
        ENDFOR

    * Теперь свободно плавающая графика
    FOR EACH oShape in .Shapes

        DO CASE
            CASE oShape.Type = msoLinkedPicture
                * Связано, может быть встроено или не быть встроенным
                IF oShape.LinkFormat.SavePictureWithDocument
                    * Он встроен, поэтому зарегистрируйте его
                    INSERT INTO Graphics VALUES ;
                        (.FullName, oShape.LinkFormat.SourceFullName, ;
                            .T. , .T., DATETIME())
                ENDIF

            CASE oShape.Type = msoPicture
                * Встроенный!
                INSERT INTO Graphics VALUES (.FullName, "", .F. , .T., DATETIME())

            OTHERWISE
                && Не интересуюсь этими
            ENDCASE
        ENDFOR

    ENDWITH

RETURN

```

5.7 Автоматизация макросов

Хотя вы можете выполнять практически любые действия Word с помощью Автоматизации, если у вас уже есть макрос Word для выполнения определенной

задачи, возможно, имеет смысл использовать существующий макрос, чем переписывать его в VFP. Это особенно верно, если макрос не предполагает передачу информации между двумя приложениями или если он является частью предварительно упакованного набора макросов.

Метод Run объекта приложения позволяет выполнять макросы Word. Вы передаете имя макроса и до 30 параметров. Например, AcceptAllRevisions – это пользовательский макрос, который принимает все версии и отключает отслеживание версий. Эта строка выполняет это:

```
oWord.Run("AcceptAllRevisions")
```

В этом случае не составит труда создать ту же функциональность с помощью Автоматизации, поскольку макрос состоит всего из двух строк, но бывают ситуации, когда переписывание существующего макроса в виде кода Автоматизации может представлять проблему. В таких случаях Run предоставляет простое решение.

5.8 Собираем все вместе



В листинге 3 (WordSample2.PRG в файлах для загрузки, доступных на сайте www.hentzenwerke.com) показана программа, которая создает список сотрудников для тасманских трейдеров. В списке используется таблица, организованная по странам, с заголовком для каждой страны в центре таблицы. Для каждого сотрудника он включает имя, дату рождения, дату приема на работу и фотографию. Концевые сноски, сохраненные для каждого сотрудника, отображаются в виде сносок к документу. На рисунках 17 и 18 показаны две страницы полученного документа.

Листинг 3. Создание полного документа. Эта программа создает список сотрудников, используя таблицу, сноски и графику. Результаты показаны на рисунках 17 и 18.

```
* WordSample2.PRG
* © 2000, Tamar E. Granor and Della Martin
* From: Microsoft Office Automation with Visual FoxPro
* Hentzenwerke Publishing. www.hentzenwerke.com

* Это пример «Собираем все вместе» из главы 5.
* Он демонстрирует использование таблиц, графиков и сносок.
* Программа создает документ, в котором показаны все
* сотрудники TasTrade в таблице, организованной по странам.
```

```
#DEFINE CR CHR(13)
#DEFINE wdHeaderFooterPrimary 1
#DEFINE wdGray25 16
#DEFINE wdAlignParagraphCenter 1
#DEFINE wdCollapseEnd 0
#DEFINE wdWord9TableBehavior 1
#DEFINE wdAutoFitContent 1
#DEFINE wdTableFormatList7 30
#DEFINE wdCellAlignVerticalCenter 1
```

```
RELEASE ALL LIKE o*
PUBLIC oWord
```

```

LOCAL oDocument, oRange, oTable, oRow, nRow
LOCAL nCountries, nEmployees, cCountry

oWord = CreateObject("Word.Application")
oWord.Visible = .t.

oDocument = oWord.Documents.Add()

* Установить шрифт для обычного стиля.
* WITH oDocument.Styles["Normal"].Font
WITH oDocument.Styles["Обычный"].Font
    .Name = "Arial"
    .Size = 12
ENDWITH

* Добавить заголовок
WITH oDocument.Sections[1].Headers[ wdHeaderFooterPrimary ]
    oRange = .Range()
    WITH oRange
        .Text = "Tasmanian Traders"
        * .Style = oDocument.Styles["Heading 1"]
        .Style = oDocument.Styles["Заголовок 1"]
        .ParagraphFormat.Alignment = wdAlignParagraphCenter
        .Shading.BackgroundPatternColorIndex = wdGray25
    ENDWITH
ENDWITH

* Открыть данные.
OPEN DATA _SAMPLES + "TasTrade\Data\TasTrade"

* Запустите запрос, чтобы собрать нужные данные в правильном
* порядке.
SELECT Country, Title, First_Name, Last_Name, Birth_Date, ;
        Hire_Date, Notes, Photo_File ;
FROM Employee ;
ORDER BY Country, Title, Last_Name, First_Name ;
INTO CURSOR Emps
nEmployees = _TALLY

* Добавить заголовков.
oRange = oDocument.Range()
WITH oRange
    * .Style = oDocument.Styles["Heading 1"]
    .Style = oDocument.Styles["Заголовок 1"]
    .ParagraphFormat.Alignment = wdAlignParagraphCenter
    .InsertAfter("Employees" + CR + CR)
    .Collapse( wdCollapseEnd )
    * .Style = oDocument.Styles["Normal"]
    .Style = oDocument.Styles["Обычный"]
ENDWITH

* Посчитайте количество разных стран.
SELECT Country, COUNT( * ) ;

```

```

FROM Emps ;
GROUP BY Country ;
INTO CURSOR Countries

```

```
nCountries = _TALLY
```

```

* Теперь добавьте таблицу с одной строкой для каждого сотрудника
* плюс по одной для каждой страны.
* Настройте таблицу на изменение размера в зависимости от
* содержимого ячеек.
oTable = oDocument.Tables.Add( oRange, nEmployees + nCountries + ;
    1, 6, wdWord9TableBehavior, wdAutoFitContent )
* Примените один из встроенных форматов.
oTable.AutoFormat( wdTableFormatList7, .T., .T., .T., .T., .T., ;
    .F., .F., .F., .F.)

* Настройте строку заголовка таблицы.
WITH oTable.Rows[1]
    * .Range.Style = oDocument.Styles[ "Heading 2" ]
    .Range.Style = oDocument.Styles[ "Заголовок 2" ]
    .Cells[1].Range.InsertAfter("Title")
    .Cells[2].Range.InsertAfter("First Name")
    .Cells[3].Range.InsertAfter("Last Name")
    .Cells[4].Range.InsertAfter("Birthdate")
    .Cells[5].Range.InsertAfter("Hired")
    * Настройте повтор этой строки на последующих страницах.
    .HeadingFormat = .T.
ENDWITH

```

```

* Теперь перебираем данные о сотрудниках, отправляя их в таблицу.
* Следите за текущей страной. Для каждой новой страны
* создайте строку "заголовок" в таблице.

```

```

cCountry = ""
nRow = 1

```

```

SELECT Emps
SCAN
    * Захватить текущую строку.
    nRow = nRow + 1
    oRow = oTable.Rows[ nRow ].Range()

    IF NOT (Country == cCountry)
        * Изменение страны
        oRow.Cells.Merge()
        *
        oRange = oRow.Range()
        * oRow.Style = oDocument.Styles[ "Heading 3" ]
        oRow.Style = oDocument.Styles[ "Заголовок 3" ]
        oRow.ParagraphFormat.Alignment = wdAlignParagraphCenter
        oRow.InsertAfter( Country )
        cCountry = Country
        nRow = nRow + 1
        oRow = oTable.Rows[ nRow ].Range()
    
```

```
ENDIF
```

```
* Теперь вставьте данные для этого сотрудника.
```

```
WITH oRow
```

```
    .Cells.VerticalAlignment = wdCellAlignVerticalCenter
```

```
    .Cells[1].Range.InsertAfter( Title )
```

```
    .Cells[2].Range.InsertAfter( First_Name )
```

```
    .Cells[3].Range.InsertAfter( TRIM( Last_Name ) )
```

```
* Добавьте сноску для информации в заметке.
```

```
oDocument.EndNotes.Add( .Cells[3].Range(), , Notes )
```

```
    .Cells[4].Range.InsertAfter( Birth_Date )
```

```
    .Cells[5].Range.InsertAfter( Hire_Date )
```

```
* Добавьте картинку в последнюю ячейку.
```

```
oPicture = oDocument.InlineShapes.AddPicture ( ;
```

```
    _SAMPLES + "TasTrade\" + Photo_File, ;
```

```
    .t., .f., .Cells[6].Range() )
```

```
ENDWITH
```

```
WITH oPicture
```

```
    .ScaleHeight = 50
```

```
    .ScaleWidth = 50
```

```
ENDWITH
```

```
ENDSCAN
```

```
* Закроем таблицы
```

```
USE IN Countries
```

```
USE IN Emps
```

```
USE IN Employee
```

```
* Покажем результат
```

```
oDocument.PrintPreview()
```

```
RETURN
```

Tasmanian Traders				
Employees				
<i>Title</i>	<i>First Name</i>	<i>Last Name</i>	<i>Birthdate</i>	<i>Hired</i>
France				
Advertising Specialist	Laurent	Pereira ^I	12/9/65	2/1/94
Marketing Associate	Xavier	Martin ^{II}	11/30/60	1/15/94
Marketing Director	Justin	Brid ^{III}	10/8/62	1/1/94
UK				
Inside Sales Coordinator	Laura	Callahan ^{IV}	1/9/58	1/30/93
Sales Manager	Steven	Buchanan ^V	3/4/55	9/13/92

Рисунок 17. Отчет сотрудника Tasmanian Traders. Первая страница включает заголовок, а затем сразу переходит в таблицу. Вы можете поместить в таблицу не только текст, но и графику, и не обязательно, чтобы каждая строка имела одинаковое количество столбцов.

Tasmanian Traders				
Title	First Name	Last Name	Birthdate	Hired
Entry Clerk	Andrew	Fuller ¹	2/19/42	7/12/91
Mali Clerk	Tim	Smith ²	6/6/73	1/15/93
Receptionist	Caroline	Patterson ³	9/11/72	5/15/93
Sales Manager	Margaret	Peacock ⁴	9/19/37	3/30/92



¹ Laurent Pereira graduated from the Ecole Supérieure des Sciences Economiques et Commerciales in 1989. He plans to continue his studies in 1994 at the University of Bochum.

² Xavier Martin is a graduate of the University of Chicago (BA) and the ESC Bordeaux (Ecole Supérieure de Commerce de Bordeaux). Mr. Martin has travelled widely in Eastern Europe, most recently in the Czech Republic, Poland, and Hungary for purposes of establishing a network of contacts for future sales activities. Mr. Martin is completely fluent in German and French (he is a native speaker of both languages); he also speaks English and understands Polish.

³ Justin Brid graduated from HEC Paris (Hautes Etudes Commerciales de Paris) in 1986 with high honors. Mr. Brid attended the American Graduate School of International Management (Thunderbird) in Glendale, Arizona. Prior to his employment with Tasmanian Traders, he held positions with several Import/Export companies (Culinary Specialties) in Germany, Switzerland, and Austria. Mr. Brid is equally fluent in French and German; he also speaks English and reads Spanish.

⁴ Laura received a BA in psychology from the University of Washington. She has also completed a course in business French. She reads and writes French.

⁵ Steven Buchanan graduated from St. Andrews University, Scotland, with a BSC degree in 1976. Upon joining the company as a sales representative in 1992, he spent 6 months in an orientation program at the Seattle office and then returned to his permanent post in London. He was promoted to sales manager in March 1993. Mr. Buchanan has completed courses in Successful Telemarketing and International Sales Management. He is fluent in French.

Рисунок 18. Использование ссылок для получения дополнительной информации. Хотя информацию из поля заметок сотрудника можно было бы поместить в таблицу, размещение ее в концевых сносках придает таблице лаконичный вид.

С помощью материала этой и предыдущей глав вы сможете справиться с большинством повседневных задач по обработке текста. Теперь мы переходим к более необычным задачам — вещам, которые вы не видите каждый день, но которые все равно могут попросить заставить Word выполнить их.

Глава 6. Продвинутый Word

Word может создавать очень сложные документы. Несколько функций помогают упорядочить более длинные документы: структуры, оглавления и указатели. Наш обзор Word заканчивается прототипом сочетания текстового процессора и баз данных: слиянием почты.

Некоторые из более сложных возможностей Word могут внести последние штрихи в документы, создаваемые с помощью Автоматизации. В этой главе рассматриваются схемы, оглавления и указатели. Кроме того, она исследует, возможно, самую востребованную из возможностей Word с точки зрения базы данных: слияние почты.

6.1 Организация документа с помощью стилей

Как схемы, так и оглавления основаны на стилях. В каждом случае вы связываете стиль с каждым уровнем заголовка, а Word делает все остальное. (На самом деле, Word может использовать этот подход и для многоуровневых списков. См. «Организация текста с помощью списков» в главе 4.)

Первым шагом в создании структуры или оглавления является использование стилей для каждого уровня заголовка в документе. Вы можете использовать встроенные стили или создать свой собственный набор стилей. В качестве дополнительного преимущества это позволяет легко гарантировать, что заголовки на одном уровне будут выглядеть одинаково повсюду.

Word распознает девять уровней заголовков. Не случайно шаблон «Обычный» включает девять стилей, названных «Заголовок 1» - «Заголовок 9», которые связаны с этими девятью уровнями заголовков. Самый простой способ подготовиться к созданию структуры и оглавления – использовать встроенные стили заголовков в собственных документах. На рисунке 1 они показаны в общих чертах.

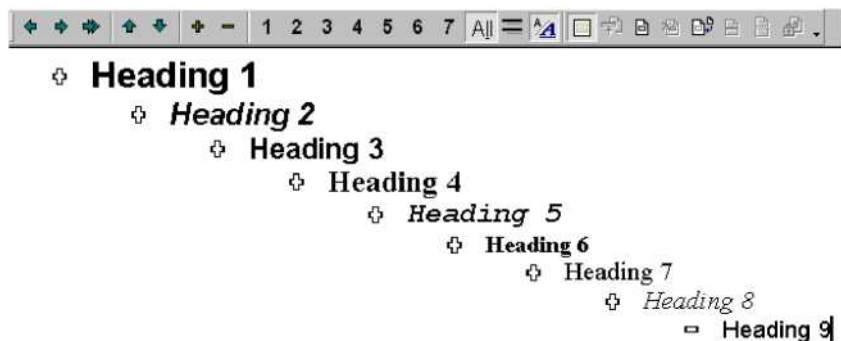


Рисунок 1. Встроенные стили заголовков. Вы можете использовать эти стили для заголовков, чтобы упростить создание структур и оглавлений, или создать свои собственные.

Если хотите, вы можете использовать другие стили. Установите уровень заголовка для стиля с помощью свойства OutlineLevel объекта ParagraphFormat объекта Style. Свойство OutlineLevel использует набор констант с такими именами, как wdOutlineLevel1, значение каждой из которых соответствует уровню структуры.



Чтобы продемонстрировать такое использование стилей, в листинге 1 представлена программа, которая создает отчет о продажах TasTrade, организованных по странам. В каждой стране клиенты перечислены в алфавитном порядке. В заголовке отчета используется стиль «Заголовок 1». В названиях стран используется «Заголовок 2», а в названиях компаний – «Заголовок 3». Объем продаж по годам для каждой компании (округленный до ближайшего доллара)

составляет основную часть отчета и использует стиль основного текста. Эта программа включена как MultiLevel.PRG в файлы загрузки для разработчиков, доступные по адресу www.hentzenwerke.com.

Листинг 1. Использование стилей заголовков. Эта программа создает документ, который упрощает создание структуры и оглавления.

```
* MultiLevel.PRG
* © 2000, Tamar E. Granor and Della Martin
* From: Microsoft Office Automation with Visual FoxPro
* Hentzenwerke Publishing. www.hentzenwerke.com

* Создайте многоуровневый документ с помощью встроенных
* стилей заголовков.

#DEFINE wdCollapseEnd 0
#DEFINE CR CHR(13)

LOCAL oDocument, oRange
LOCAL cCountry, cCompany, nYear

* Создайте экземпляр Word.
* Сделайте его общедоступным для демонстрационных целей и так,
* что вы можете использовать oWord в следующих примерах.

RELEASE ALL LIKE o*
PUBLIC oWord
oWord = CreateObject("Word.Application")

* Сделайте Word видимым.
oWord.Visible = .T.

* Создайте новый документ
oDocument = oWord.Documents.Add()

OPEN DATA _SAMPLES + "TasTrade\Data\TasTrade"

* Соберите информацию о клиенте/заказе. Результат содержит одну
* запись на компанию в год, сортируйте по стране, затем по компании,
* затем год в порядке убывания.

SET ENGINEBEHAVIOR 70
SELECT Company_Name, Country, YEAR(Order_Date) AS Order_Year, ;
       ROUND(SUM(Order_Line_Items.Unit_Price*Order_Line_Items.Quantity - ;
(0.01*Orders.Discount*Order_Line_Items.Unit_Price*Order_Line_Items.Quantity)) +;
       Orders.Freight, 0) AS OrderTotal ;
FROM Customer ;
JOIN Orders ;
ON Customer.Customer_Id = Orders.Customer_Id ;
JOIN Order_Line_Items ;
ON Orders.Order_Id = Order_Line_Items.Order_Id ;
GROUP BY 3, 2, 1 ;
ORDER BY 2, 1, 3 DESC ;
INTO CURSOR CompanyInfo

* Отправьте данные в документ следующим образом:
* Страна - Заголовок 1
* Компания - Заголовок 2
* Итого с некоторым текстом - это основной текст.

oRange = oDocument.Range()
WITH oRange
  * .Style = oDocument.Styles[ "Heading 1" ]
  .Style = oDocument.Styles[ "Заголовок 1" ]
```

```

        .InsertAfter("Tasmanian Traders Sales by Country and Company" + CR)
        .Collapse( wdCollapseEnd )
    ENDWITH

cCountry = ""
cCompany = ""
nYear = 0

SCAN
    WITH oRange
        .Collapse( wdCollapseEnd )
        IF NOT (Country == cCountry)
            * .Style = oDocument.Styles[ "Heading 2" ]
            .Style = oDocument.Styles[ "Заголовок 2" ]
            .InsertAfter( Country + CR )
            .Collapse( wdCollapseEnd )
            cCountry = Country
            cCompany = ""
            nYear = 0
        ENDIF

        IF NOT (Company_Name == cCompany)
            * .Style = oDocument.Styles[ "Heading 3" ]
            .Style = oDocument.Styles[ "Заголовок 3" ]
            .InsertAfter( Company_Name + CR )
            .Collapse( wdCollapseEnd )
            cCompany = Company_Name
            nYear = 0
        ENDIF

        * .Style = oDocument.Styles[ "Body Text" ]
        .Style = oDocument.Styles[ "Обычный" ]
        .InsertAfter( "Total Sales for " + ;
            TRANSFORM(Order_Year,"9999") + " = " + ;
            TRANSFORM(OrderTotal,"@B, $$ 99,999,999") + CR)
        .Collapse( wdCollapseEnd )
    ENDWITH
ENDSCAN

```

Рисунок 2 показывает часть полученного документа.

Tasmanian Traders Sales by Country and Company

Argentina

Cactus Comidas para llevar

Total Sales for 1995 = \$1,012

Total Sales for 1994 = \$899

Total Sales for 1992 = \$17,738

Océano Atlántico Ltda.

Total Sales for 1995 = \$2,948

Total Sales for 1994 = \$112

Total Sales for 1993 = \$2,133

Total Sales for 1992 = \$367

Rancho grande

Total Sales for 1995 = \$697

Total Sales for 1994 = \$1,988

Austria

Ernst Handel

Total Sales for 1995 = \$31,221

Total Sales for 1994 = \$47,050

Total Sales for 1993 = \$24,476

Total Sales for 1992 = \$5,372

Piccolo und mehr

Total Sales for 1995 = \$3,345

Total Sales for 1994 = \$8,816

Total Sales for 1993 = \$12,072

Рисунок 2. Многоуровневый документ. Заголовки в этом документе, созданные программой из листинга 1, используют встроенные стили заголовков.

Работа со структурой

Структура в Word – это скорее состояние ума, чем отдельная сущность. Она позволяет скрыть детали документа, сохраняя при этом его структуру. Поскольку разные части структуры могут быть расширены в разной степени, структуры Word также предоставляют эффект подробного отчета.

Используя соответствующие стили для создания заголовков в документе, вы обрисовали документ. Чтобы увидеть схему, все, что вам нужно сделать, это переключиться на режим «Структура», выбрав «Вид|Структура» в меню. Как показано на рисунке 1, Outline View включает специальную панель инструментов для работы со структурами. Она позволяет интерактивным пользователям разворачивать и сворачивать структуру, определять, какие уровни заголовков являются видимыми, решать, показывать ли весь основной текст или только первую строку, а также манипулировать элементами в структуре, изменяя их уровень. Когда документ отображается в режиме структуры, при его печати печатается отображаемая структура, а не весь документ. Мы предполагаем, что именно это вы и должны получить от текстового процессора WYSIWYG, но это несколько удивительно, поскольку другие представления не влияют на печать.

Поскольку отображение информации на экране – это визуальная проблема, а Автоматизация обычно выполняется «за кулисами», что все это означает для Автоматизации? Вероятно, вам не понадобится отображать структуру в настройках Автоматизации, но вам может потребоваться распечатать ее, для чего потребуется переключиться на режим структуры. К счастью, переключение в режим структуры и последующие манипуляции со структурой можно выполнить, даже если Word скрыт.

Ключевым объектом для работы со структурами является объект View, доступный через свойство View объекта Window. Доступ к объекту Window осуществляется с помощью свойства ActiveWindow документа. Итак, чтобы установить для ActiveDocument значение OutlineView, используйте такой код:

```
#DEFINE wdOutlineView 2
oWord.ActiveDocument.ActiveWindow.View.Type = wdOutlineView
```

На рисунке 3 показаны результаты выполнения этого кода для многоуровневого документа с рисунка 2. Другие значения свойства Type View включают wdNormalView (1) и wdPrintPreview (4).



Рисунок 3. Схема документа. Если заголовки документа используют правильные стили, создать структуру так же просто, как переключиться в режим структуры.

Перейдя в режим Outline View, вы можете развернуть и свернуть структуру целиком или по частям, используя методы View ExpandOutline и CollapseOutline. Каждый принимает один необязательный параметр – диапазон, который нужно расширить или свернуть. Если параметр опущен, текущая точка выбора/вставки определяет, что будет развернуто или свернуто. Например, чтобы свернуть сведения о поставщике «Cactus Comidas para llevar» на рисунке 3, введите следующий код:

```
oRange = oWord.ActiveDocument.Paragraphs[3].Range()
oWord.ActiveDocument.ActiveWindow.View.CollapseOutline( oRange )
```



В ExpandOutline есть ошибка (признанная Microsoft). Когда вы передаете ему диапазон, состоящий из абзаца заголовка, который имеет основной текст как над, так и под ним, он расширяет этот абзац, показывая основной текст под ним, но также показывает основной текст над ним. Эта ошибка расширения вызывает проблемы при попытке свернуть эту часть структуры. Для этого вам придется свернуть гораздо большую часть структуры, чем следовало бы.

Метод ShowHeading позволяет определить, сколько уровней заголовков отображается в целом. Вы передаете ему число, все заголовки до указанного уровня включительно отображаются, а все заголовки ниже этого уровня, а также основной текст скрываются. Например, для отображения заголовков до уровня 3 используйте этот код:

```
oWord.ActiveDocument.ActiveWindow.View.ShowHeading( 3 )
```

Если вы хотите отобразить все заголовки, вызовите метод ShowAllHeadings:

```
oWord.ActiveDocument.ActiveWindow.View.ShowAllHeadings()
```

Чтобы отображать только первую строку основного текста, установите для свойства ShowFirstLineOnly значение .T.; это хороший способ получить краткое описание документа, не показывая только заголовки. (В примере, показанном на рисунке 3, изменение этого свойства не имеет никакого значения, поскольку каждый абзац состоит только из одной строки.) Если вы показываете все уровни, вы получаете первую строку каждого абзаца. К сожалению, это свойство работает только на экране; это не влияет на вывод на печать.

Несмотря на невозможность использовать ShowFirstLineOnly, как только вы расположите схему так, как вам нужно, вызовите метод PrintOut документа (описанный в разделе «Печать» главы 4), чтобы распечатать схему.

Создание оглавления

Если у вас есть документ с заголовками, в которых используются стили уровня заголовков, создание оглавления проще простого. Один вызов метода Add коллекции TablesOfContents документа, и все готово. Все, что вам нужно знать, это где разместить оглавление (диапазон) и какие уровни заголовков вы хотите включить. Этот код добавляет оглавление прямо в начало документа. Он включает заголовки до уровня 3.

```
#DEFINE wdNormalView 1  
oWord.ActiveDocument.ActiveWindow.View.Type = wdNormalView  
oRange = oWord.ActiveDocument.Range(0,0)  
oWord.ActiveDocument.TablesOfContents.Add( oRange, .T., 1, 3 )
```

Рисунок 4 показывает одну страницу результатов для документа на рисунке 2.

TablesOfContents – это коллекция объектов TableOfContents. (Обратите внимание на слово «Tables» во множественном числе в названии коллекции.) Каждый объект TableOfContents представляет одно оглавление в документе. Зачем вам больше одного? Потому что у вас может быть отдельное оглавление для каждой главы или каждого основного раздела документа.

QUICK-Stop.....	9
Toms Spezialitäten	9
Ireland.....	9
Hungry Owl All-Night Grocers.....	9
Italy.....	9
Franchi S.p.A.....	9
Magazzini Alimentari Riuniti.....	9
Reggiani Caseifici.....	9
Mexico.....	10
Ana Trujillo Emparedados y helados.....	10
Antonio Moreno Taquería.....	10
Centro comercial Moctezuma.....	10
Pericles Comidas clásicas.....	10
Tortuga Restaurante.....	10
Norway.....	10
Santé Gourmet.....	10
Poland.....	10
Wolski Zajazd.....	10
Portugal.....	11
Faria Bacalhau e Frutos do Mar.....	11
Princesa Isabel Vinhos.....	11
Spain.....	11
Bóldo Comidas preparadas.....	11
FISSA Fabrica Inter. Salchichas S.A.....	11
Galería del gastrónomo.....	11
Godos Cocina Típica.....	11
Romero y tomillo.....	11
Sweden.....	12
Berglunds snabbköp.....	12
Folk och få HB.....	12
Switzerland.....	12
Chop-suey Chinese.....	12
Richter Supermarkt.....	12
UK.....	12
Around the Horn.....	12
B's Beverages.....	12
Consolidated Holdings.....	13
Eastern Connection.....	13
Island Trading.....	13
North/South.....	13
Seven Seas Imports.....	13
USA.....	13
Great Lakes Food Market.....	13
Hungry Coyote Import Store.....	13
Lazy K Hountry Store.....	13
Let's Stop N Shop.....	13
Lonesome Pine Restaurant.....	14
Old World Delicatessen.....	14
Rattlesnake Canyon Grocery.....	14
Save-a-lot Markets.....	14
Split Rail Beer & Ale.....	14
The Big Cheese.....	14

Рисунок 4. Оглавление. Использование встроенных стилей заголовков упрощает создание оглавления.

Метод Add принимает довольно много параметров. Вот синтаксис, в котором перечислены те из них, которые вы, скорее всего, будете использовать:

```
oDocument.TablesOfContents.Add( oRange, lUseHeadingStyles,
                                nFirstHeadingLevel, nLastHeadingLevel, lUseFields, ,
                                lRightAlignPageNums, lIncludePageNums, cAddedStyles )
```

oRange	Object	Диапазон, в котором должно быть размещено оглавление.
lUseHeadingStyles	Logical	Указывает, следует ли создавать оглавление на основе встроенных стилей заголовков. По умолчанию Т.
nFirstHeadingLevel	Numeric	Стиль заголовка с наименьшим номером, который будет включен в оглавление. По умолчанию 1.
nLastHeadingLevel	Numeric	Стиль заголовка с наибольшим номером, который будет включен в оглавление. По умолчанию 9.
lUseFields	Logical	Указывает, включены ли в оглавление элементы, помеченные как поля оглавления. Дополнительную информацию см. в справке. По

		умолчанию .F.
IRightAlignPageNums	Logical	Указывает, выровнены ли номера страниц на странице по правому краю. По умолчанию .T.
IIncludePageNums	Logical	Указывает, включены ли номера страниц в оглавление. По умолчанию .T.
cAddedStyles	Character	Разделенный запятыми список стилей, отличных от встроенных стилей заголовков, которые также следует использовать при создании оглавления.

После создания оглавления вы можете установить символ, заполняющий пространство между записями и номерами страниц в каждой строке, используя свойство TabLeader объекта TableOfContents (а не коллекции). В таблице 1 показаны значения констант для TabLeader.

Таблица 1. Заполнение пространства. Свойство TabLeader определяет, как заполняется пространство между заголовками и номерами страниц в оглавлении.

Константа	Значение	Константа	Значение
wdTabLeaderSpaces	0	wdTabLeaderLines	3
wdTabLeaderDots	1	wdTabLeaderHeavy	4
wdTabLeaderDashes	2	wdTabLeaderMiddleDot	5

Несколько других свойств TableOfContents позволяют изменять параметры, заданные с помощью Add, например IncludePageNumbers, RightAlignPageNumbers, LowerHeadingLevel (соответствует nFirstHeadingLevel) и UpperHeadingLevel (соответствует nLastHeadingLevel).

Как вы могли догадаться из наших комментариев к параметру IUseFields, существует другой подход к созданию оглавления. Он включает в себя маркировку каждого элемента, который вы хотите включить в список, кодом. Дополнительные сведения см. в разделе «Использование кодов полей для индексов, оглавлений и других таблиц» в обычной справке Word.

6.2 Создание указателей

Длинные документы гораздо полезнее, когда они индексируются. К сожалению, лучшим способом создания высококачественного индекса по-прежнему остается старомодный способ — поручить это сделать человеку. Это связано с тем, что ни один автоматизированный метод не может прочитать содержание рукописи и решить, заслуживает ли конкретное использование слова или фразы упоминания в указателе. Фактически, эта проблема является причиной того, что многие поисковые системы большую часть времени выдают нерелевантные результаты.

Тем не менее, Word предлагает несколько способов создания индексов. Однако только один из методов действительно подходит для Автоматизации, поскольку другой предполагает значительное вмешательство человека. Легко автоматизируемый метод использует таблицу соответствия, в которой перечислены все слова, подлежащие индексированию, и записи индекса для них.

Решение о том, что индексировать

Таблица соответствия представляет собой отдельный документ, содержащий

стандартную таблицу Word с двумя столбцами. Вы можете создать его в интерактивном режиме или с помощью Автоматизации. Таблица 2 является частью таблицы соответствия главы 3 этой книги «Visual FoxPro как клиент Автоматизации» (хотя на самом деле это не та таблица, которая использовалась для индексации книги).

Таблица 2. Создание индекса. Один из способов индексирования документа — использование таблицы, в которой перечислены термины, подлежащие индексированию, вместе с записями индекса.

1426	error:1426
1429	error:1429
Activate	Activate
Application	Application
client	client
collection	collection
collections	collection
Command Window	Command Window
CreateObject()	CreateObject()
CreateObjectEx()	CreateObjectEx()
DCOM	DCOM
debugger	debugger
debugging	debugging

В таблице приведены несколько пунктов, на которые стоит обратить внимание. Во-первых, если слово встречается в документе в нескольких формах, вам необходимо перечислить все эти формы в таблице. (Обратите внимание на «collection» и «collections» в первом столбце, оба индексируются как «collection».)

Во-вторых, вы можете обрабатывать несколько уровней индекса внутри таблицы, разделяя их двоеточием во втором столбце. Обратите внимание на записи «1426» и «1429». В указателе они оба будут частью общей записи «error», но разделены на «1426» и «1429».

В-третьих, большинство строк таблицы содержат одно и то же в двух столбцах. Если вы создаете таблицу в интерактивном режиме, это отличное место, чтобы воспользоваться функцией Word, которая обычно раздражает. В Word почти всегда доступен пункт меню «Правка | Повторить ввод» (сочетание клавиш Ctrl-Y). Когда вы создаете таблицу соответствия, вы можете использовать ее, чтобы сократить объем набора текста почти вдвое. Введите строку в первый столбец, затем нажмите Ctrl-Y, чтобы повторить ее во втором столбце.

Наконец, Word, к сожалению, недостаточно умен, чтобы заметить, когда одна индексная фраза содержится в другой. Например, в Таблице 2 присутствует слово «client (клиент)». Если бы мы добавили в таблицу «клиентское приложение», то при создании индекса фраза типа «Использование VFP в качестве клиентского приложения» была бы проиндексирована дважды: один раз для «клиента» и один раз для «клиентского приложения». Остерегайтесь этой проблемы, когда решаете, что поместить в таблицу.

Маркировка элементов для индекса

После того, как у вас есть таблица соответствия, следующим шагом будет пометка соответствующих элементов в документе. Это та часть, где таблица соответствия начинает работать и фактически экономит время. В интерактивном режиме выберите в меню «Вставка|Ссылка|Оглавление и указатели...». Откроется диалоговое окно, показанное на рисунке 5. Выберите «Автопометка...» и выберите файл соответствия.

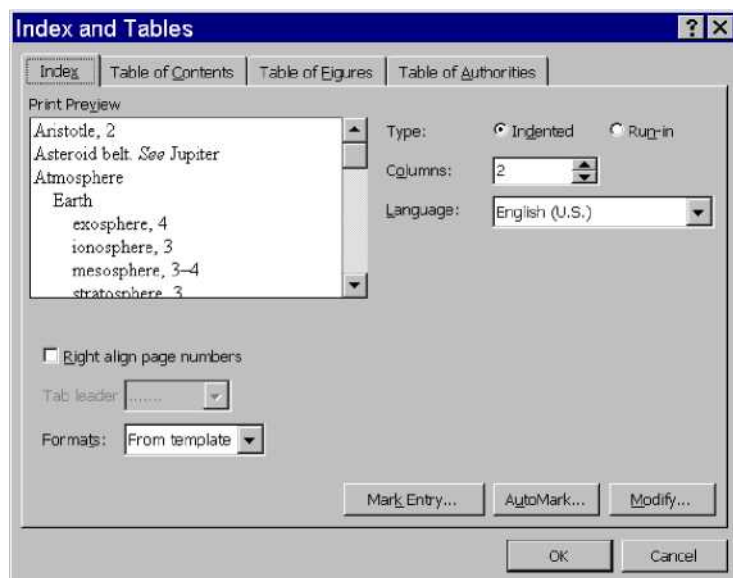


Рисунок 5. Создание индекса. Чтобы пометить элементы в документе на основе таблицы соответствия, выберите «Автопометка...» в этом диалоговом окне.

Word просматривает документ и добавляет в документ специальные поля везде, где находит элемент из левого столбца таблицы соответствия.

Чтобы автоматизировать этот процесс, вы используете метод `AutoMarkEntries` коллекции `Indexes` документа, например:

```
oDocument.Indexes.AutoMarkEntries("d:\autovfp\chapter3\wordlist.doc")
```

Генерация индекса

После того, как элементы отмечены, создать сам индекс очень просто. В интерактивном режиме вы помещаете курсор туда, где должен быть размещен указатель, затем снова вызываете диалоговое окно «Оглавление и указатели» (показанное на рисунке 5), выбираете формат из раскрывающегося списка «Форматы», проверяете, если хотите, выравнивать номера страниц по правому краю и выбираете «ОК». На рисунке 6 показана часть индекса, созданного для главы 3 «Visual FoxPro как клиент Автоматизации» на основе соответствия в таблице 2.

Activate	4	object reference	2, 6, 7
Application	1, 2, 3, 4	Office	1, 2, 6, 8, 9
collection	4, 6, 7	Outlook	2
Command Window	7	parameter	1, 2, 8
CreateObject()	1, 4, 5	PixelsToPoints	3
DCOM	2	polymorphism	1
debugging	2	PowerPoint	1, 2, 4
Developer Download files	3	registry	5
error 1426	8	server	1, 2, 5, 6, 8, 9
error 1429	8	SET OLEOBJECT	4, 5

Рисунок 6. Сгенерированный индекс. Применение таблицы соответствия в Таблице 2 к Главе 3 приводит к получению этого индекса.

Чтобы создать индекс с помощью Автоматизации, вы используете метод `Add Indexes`, передавая ему диапазон, в котором должен быть размещен индекс, и несколько других параметров. В этом примере создается тот же индекс, что и на рисунке 6, и он располагается в конце документа.

```
#DEFINE wdCollapseEnd          0
#DEFINE wdHeadingSeparatorNone 0
#DEFINE wdIndexIndent          0
#DEFINE wdTabLeaderDots        1

oRange = oDocument.Range()
oRange.Collapse( wdCollapseEnd )

oIndex = oDocument.Indexes.Add( oRange, ;
    wdHeadingSeparatorNone, .T., wdIndexIndent, 2, .F.)

* Изменить пунктирную линию на точки
oIndex.TabLeader = wdTabLeaderDots
oIndex.Update()
```

Метод `Add` имеет ряд параметров. В примере после диапазона мы указываем, что для разделения записей для каждой буквы алфавита в индексе не требуется никаких знаков препинания. Другие варианты включают `wdHeadingSeparatorBlankLine` (1) и `wdHeadingSeparatorLetter` (2). Свойство `HeadingSeparator` объекта `Index` соответствует этому параметру.

Третий параметр соответствует свойству `RightAlignPageNumbers` объекта `Index` и определяет, следуют ли номера страниц сразу за записью или выровнены по правому краю.

Четвертый параметр определяет, как индекс обрабатывает подзаписи. Значение по умолчанию, `wdIndexIndent` (0), отображает каждую подзапись с отступом на отдельной строке под основным заголовком. Альтернативный вариант, `wdIndexRunIn` (1), разделяет подзаписи точками с запятой в одной строке с основным заголовком. Не используйте его, когда выравниваете номера страниц по правому краю, в этом случае это выглядит ужасно.

Пятый параметр указывает количество столбцов в индексе, в примере два. Оно соответствует свойству `NumberOfColumns` объекта `Index`.

Последний параметр, показанный в примере, указывает, что акцентированные буквы должны сочетаться с основными буквами. Задайте `.T.` для разделения букв с диакритическими знаками в отдельные индексные списки. Этот параметр соответствует свойству `AccentedLetters` объекта `Index`.

У `Add` есть несколько дополнительных параметров, но они довольно непонятны. Подробности смотрите в справке.

Форматирование индексов

Как и в случае с оглавлениями, после добавления указателя вы можете использовать его свойства, чтобы изменить его внешний вид. Объект `Index` имеет ряд свойств, влияющих на его внешний вид, некоторые из них соответствуют параметрам метода `Add` коллекции `Indexes`, как отмечалось ранее. Кроме того, как и в случае с `TableOfContents`, свойство `TabLeader` позволяет изменять символ, который появляется между записью и номером страницы. Константы такие же, как и для `TableOfContents`, см. Таблицу 1.

Метод Update (используемый в примере из предыдущего раздела), как следует из названия, обновляет индекс.

Это полезно как при изменении отмеченных записей указателя, так и при изменении форматирования.

6.3 Объединение документов с данными

Слияние почты – еще одна из убийственных функций, благодаря которым текстовые процессоры есть у каждого на столе. Возможность написать документ один раз, а затем неоднократно объединять его с данными для создания персонализированных документов имела смысл и для предприятий, ученых и домашних пользователей.

Word предлагает множество способов объединения данных и документов. Самый известный – самый сложный, использующий встроенную возможность слияния почты. Однако также возможно объединить документы, используя функцию «Найти и заменить» и комбинируя встроенные возможности слияния почты с некоторым ручным трудом. Здесь мы рассмотрим каждый из этих подходов и расскажем, почему мы считаем, что третий подход лучше всего подходит для Автоматизации.

Структура слияния почты Word

Слияние почты включено в Word во многих версиях. В интерактивном режиме имеется мастер под названием Mail Merge Helper (см. рис. 7), который помогает пользователям пройти весь процесс. Однако даже с помощью этого инструмента настроить новый документ слияния почты, подключить его к данным и получить результаты непросто. Мы обе провели много часов по телефону, помогая родственникам и друзьям пройти этот процесс.

За помощником по слиянию почты находится ряд объектов. В справке перечислено не менее восьми различных объектов, имена которых начинаются с «MailMerge». Но не сложность объектной модели заставляет нас рекомендовать альтернативные подходы; это хрупкость связи между документом и данными.

Указывая, что документ Word является документом слияния почты, вы указываете источник данных для слияния. Это может быть другой документ Word, текстовый файл или любой из множества файлов базы данных, включая, конечно, FoxPro. Для большинства источников данных Word использует ODBC для чтения данных.

Если файл данных удален или перемещен или что-то происходит с драйвером ODBC, слияние перестает работать. Для многих людей, использующих таблицы FoxPro 2.x при слиянии почты, было неприятным сюрпризом, когда они установили пакет обновления 3 для VFP 6 (или что-нибудь еще, что устанавливало ODBC 4.0), поскольку он отключил драйвер ODBC для таблиц FoxPro 2.x и не смог автоматически заменить его драйвером VFP. Почтовые слияния, которые работали годами, потерпели неудачу.

Необходимость иметь дело с драйверами и подключениями ODBC делает этот подход сложнее и, особенно, более рискованным, чем он должен быть. Если вы не имеете дело с очень большими таблицами, есть другие, более простые способы решения этой проблемы. Если вы имеете дело с большими таблицами, планируйте контролировать аспекты ODBC самостоятельно, а не полагаться на то, что находится на компьютере пользователя. (Чтобы узнать об управлении ODBC и соединениями, прочтите родственный том книги в серии Essentials – «Клиент-серверные приложения с Visual FoxPro 6.0 и SQL Server 7.0».) Главное, что вам нужно сделать, – это передать строку подключения в OpenDataSource объекта MailMerge (обсуждается далее в этой главе).

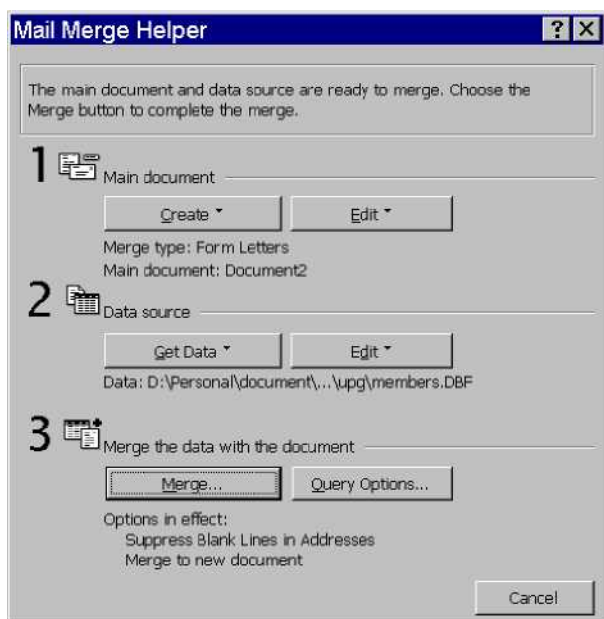


Рисунок 7. Слияние почты. В этом диалоговом окне интерактивные пользователи проходят шаги, необходимые для использования встроенной в Word функции слияния почты.

Замена данных с помощью функции «Найти и заменить»

Одной из альтернатив слиянию почты Word является использование объектов Find и Replace. Поскольку они подробно обсуждались в главе 5 (см. «Поиск и замена»), мы будем говорить здесь только о конкретных проблемах, связанных с их использованием для объединения документов и данных.

Основная идея состоит в том, чтобы создать документ, содержащий желаемый результат, за исключением набора строк, которые необходимо заменить данными из вашей базы данных. Альтернативно, вы можете использовать шаблон, на основе которого вы создаете документ, а затем заменяете строки.

Основная проблема заключается в том, чтобы убедиться, что вы заменяете только то, что собираетесь заменить. Когда вы работаете в интерактивном режиме, это не проблема, поскольку вы можете увидеть каждое совпадение, прежде чем согласиться на его замену. Однако для Автоматизации вам нужно лучшее решение. Одна из возможностей – заключить строки в специальный символ или символы. Например, вы можете заключить заменяемые строки в угловые скобки (например, «<имя>») или восклицательные знаки («!имя!»). Тогда код замены строки будет выглядеть примерно так:

```
#DEFINE wdReplaceAll 2
oRange = oDocument.Range(0,0)
WITH oRange.Find
    .Text = "!имя!"
    .Replacement.Text = TRIM(Employee.First_Name) ;
                        - " " + TRIM(Employee.Last_Name)
    .Execute( , , , , , , , , , wdReplaceAll)
ENDWITH
```

Самая большая проблема при таком подходе – найти подходящий специальный символ для разделения строк. Вам действительно нужно быть уверенным, что это что-то, что не появится где-либо еще в документе. Хотя маловероятно, что текст будет окружен специальным символом, например восклицательным знаком, всегда существует вероятность того, что пользователь может сделать что-то странное.

Для реализации этого подхода вы можете использовать таблицу. Рассмотрите возможность помещения строк и форматирования, которые нужно найти и заменить, в таблицу, а также наличие единого метода для выполнения поиска и замены. Тем не менее, хотя поиск и замена строк с разделителями является жизнеспособным решением для объединения данных в документы, мы все же думаем, что есть лучший способ.

Отступите на 10 ярдов и нанесите удар.

Так что же делать? Существует Word с очень хорошим механизмом слияния почты, но необходимость использования ODBC для доступа к таблицам FoxPro является серьезным препятствием. С другой стороны, остальные возможности слияния почты действительно полезны. Решение состоит в том, чтобы избежать аспектов ODBC, используя при этом все остальные возможности Word.

Способ сделать это – создать источник данных для слияния почты «на лету», отправить данные FoxPro в Word и прикрепить их к документу на время, достаточное для их слияния. Эта стратегия подходит, когда объем данных, подлежащих объединению, мал или умерен, но ее, возможно, придется пересмотреть для чрезвычайно больших наборов данных. (Однако помните, что независимо от того, какой подход вы используете, данные необходимо каким-то образом отправлять в Word, поэтому этот метод может быть не хуже любого другого, и он действительно дает вам больший контроль над процессом и меньшую вероятность проблем, вызванных конечными пользователями, чем традиционный подход.)

Документы, участвующие в слиянии почты

Если исключить ODBC, слияние почты включает в себя два или три документа. Первый – это то, что Word называет *основным документом*. Это документ, содержащий текст письма, конверта, этикетки или чего-то еще, что вы пытаетесь создать. Большая часть этого документа похожа на любой другой документ. Исключением является то, что в некоторых местах *поля содержат* специальные маркеры, указывающие, что в этом месте что-то нужно заменить. Word на самом деле предлагает широкий набор полей, включая такие поля, как дата, время и номер страницы. Для слияния почты нас особенно интересуют поля типа MergeField.

Второй документ в слиянии почты – это *источник данных*. Это документ, содержащий данные для подстановки в поля. Он содержит обычную таблицу Word с одним столбцом для каждого MergeField. В описанной нами стратегии мы создадим этот документ «на лету».

Третий документ является необязательным. Это *результат*, созданный путем слияния основного документа с источником данных.

Мы предпочитаем выполнять объединение в новый документ, а не непосредственно на принтер, но могут возникнуть ситуации, когда вы решите пропустить этот шаг.

Объекты, участвующие в слиянии почты

Фактически основной объект слияния почты называется MailMerge, доступ к нему осуществляется через свойство MailMerge Document. Свойство MailMerge Fields ссылается на коллекцию MailMergeFields, состоящую из объектов MailMergeField. Эти объекты представляют поля слияния почты в основном документе. Когда документ прикрепляется к источнику данных, свойство DataSource MailMerge обращается к объекту MailMergeDataSource. Его свойство DataFields ссылается на коллекцию объектов MailMergeDataField, которые предоставляют информацию о

полях в источнике данных. MailMergeDataSource также имеет свойство FieldNames, которое ссылается на коллекцию MailMergeFieldNames с информацией об именах полей для данных.

Если вам кажется, что объектов много, то это так, но в описанной здесь стратегии вам придется работать напрямую только с объектами MailMerge и MailMergeFields.

Создание основного документа

Первым шагом является создание основного документа. Есть несколько способов сделать это, не все из которых связаны с Автоматизацией. Ваши пользователи могут просто создать основной документ с помощью помощника по слиянию писем. Проблема с этим подходом, конечно, заключается в том, что к таким документам будут прикреплены источники данных, но есть некоторые решения (обсуждаемые в следующем разделе «Прикрепление источника данных»).

Пользователи также могут создавать основные документы вручную, вставляя поля слияния почты, выбрав в меню «Вставить | Поле». Появится диалоговое окно, показанное на рисунке 8. Чтобы добавить поле слияния почты, выберите «Слияние почты» в левом столбце и MergeField справа, затем введите имя поля в поле «Коды полей», как показано на рисунке.



Рисунок 8. Вставка полей слияния почты. Пользователи могут создавать документы слияния почты вручную, а не с помощью помощника по слиянию писем.

Конечно, вы можете создавать основные документы с помощью Автоматизации, как и другие документы. Фактически, вы также можете использовать гибридный подход, сначала настраивая документ с помощью Автоматизации, а затем позволяя пользователю редактировать его.

Чтобы добавить поле слияния почты в документ, используйте метод Add коллекции MailMergeFields. Он требует двух параметров, а именно:

```
oDocument.MailMerge.Fields.Add( oRange, cField )
```

oRange **Object** Ссылка на диапазон, в котором должно находиться поле слияния почты.

cField **Character** Поле слияния почты, которое необходимо вставить.

Прикрепление источника данных

Одна из вещей, которая делает Помощник по слиянию настолько полезным, заключается в том, что он предоставляет список полей в источнике данных и

позволяет вам выбирать из этого списка при создании основного документа. На рисунке 9 показана часть панели инструментов слияния с открытым раскрывающимся списком «Вставить поле слияния», в котором показан список полей из таблицы «Сотрудники» TasTrade.



Рисунок 9. Добавление полей в интерактивном режиме. Когда основной документ прикреплен к источнику данных, вы можете добавлять поля, выбирая их на панели инструментов «Слияние».

Использование источников заголовков для списков полей

Если мы хотим, чтобы пользователи могли создавать и редактировать основные документы, нам нужен способ предоставить им список полей, даже если мы не хотим создавать постоянные связи между основными документами и источниками данных. Несколько методов объекта MailMerge позволяют нам настроить списки полей.

Фактически существует два типа связей между основным документом и данными. Его можно подключить к реальному источнику данных, содержащему записи, доступные для объединения. Однако вместо этого основной документ можно подключить к *источнику заголовка*, документу, который предоставляет только имена полей для слияния, но не содержит фактических данных слияния.

Преимущество источника заголовка в том, что он небольшой и его легко создать. Мы можем использовать источник заголовка, чтобы предоставить пользователям список полей при создании или редактировании основного документа, но подождем, чтобы создать полный источник данных, пока пользователь не будет готов к фактическому слиянию. Мы также можем создать источник заголовка и скрыть его от пользователя, если это подходящая стратегия. (Это может быть уместно, если известно, что пользователи удаляют файлы, когда им этого не следует.)



Листинг 2 – это программа, которая присоединяет источник заголовка к основному документу на основе списка полей в таблице или представлении. (Это CreateHeaderSource.PRG в файлах для загрузки для разработчиков, доступных на сайте www.hentzenwerke.com.) Ключом ко всему процессу является вызов метода CreateHeaderSource MailMerge, остальное – это типичная манипуляция строками FoxPro. Вы можете вызвать эту процедуру следующим образом:

```
DO CreateHeaderSource WITH oDocument, ;  
    _SAMPLES+"TasTrade\Data\Employee", ;
```


"C:\Temp\EmployeeHeader.DOC"

Листинг 2. Создание источника заголовка. Эта программа генерирует источник заголовка «на лету» из любой таблицы или представления и прикрепляет его к документу.

```
* CreateHeaderSource.PRG
* © 2000, Tamar E. Granor and Della Martin
* From: Microsoft Office Automation with Visual FoxPro
* Hentzenwerke Publishing. www.hentzenwerke.com

* Создайте источник заголовка для текущего документа.
* на основе таблицы или представления
* Предполагается:
*     Word открыт.

LPARAMETERS oDocument, cCursor, cDocument
* oDocument = документ, для которого должен быть создан источник заголовка.
* cCursor   = имя файла таблицы или представления, включая путь.
* cDocument = имя файла, включая путь, где
*             исходный документ заголовка должен быть сохранен.

* Проверьте параметры.
IF PCOUNT() < 3
    MESSAGEBOX("Необходимо указать имя таблицы/представления и имя документа")
    RETURN .F.
ENDIF

IF VarType(oDocument) <> "O"
    MESSAGEBOX("Документ не указан")
    RETURN .F.
ENDIF

IF VarType(cCursor) <> "C" OR EMPTY(cCursor)
    MESSAGEBOX("Имя таблицы/представления должно быть символьным")
    RETURN .F.
ENDIF

IF VarType(cDocument) <> "C" OR EMPTY(cDocument)
    MESSAGEBOX("Имя документа должно быть символьным")
    RETURN .F.
ENDIF

LOCAL nFieldCount, cFieldList, aFieldList[1], nField

* Открыть таблицу/представление
USE (cCursor) AGAIN IN 0 ALIAS MergeCursor
SELECT MergeCursor

* Получить список полей
nFieldCount = AFIELDS( aFieldList, "MergeCursor" )
* Пройдите по списку, создав строку, разделенную запятыми
cFieldList = ""
FOR nField = 1 TO nFieldCount
    IF aFieldList[ nField, 2] <> "G"
        * Не могу использовать поля General
        cFieldList = cFieldList + aFieldList[ nField, 1] + ","
    ENDIF
ENDFOR
cFieldList = LEFT( cFieldList, LEN(cFieldList) - 1 )

* Прикрепите заголовок к документу
oDocument.MailMerge.CreateHeaderSource( cDocument, , , cFieldList )
```


USE IN MergeCursor

RETURN

Результирующий файл заголовка представляет собой просто таблицу Word из одной строки, каждый столбец которой содержит имя поля.

Когда вы открываете основной документ в интерактивном режиме и источник заголовка или источник данных отсутствует, Word настаивает на том, чтобы вы либо нашли недостающий источник, либо приняли меры. В Word 2000, когда то же самое происходит с Автоматизацией, Word просто открывает файл и отсоединяет источник заголовка или сам источник данных.



К сожалению, в Word 97, когда вы открываете основной документ с помощью Автоматизации и источник данных отсутствует, Word настаивает на том, чтобы вы нашли недостающий источник данных, хотя это удивительно изобретательно, если вы указываете не на тот файл.

Метод OpenHeaderSource MailMerge присоединяет существующий источник заголовка к основному документу. OpenDataSource прикрепляет существующий источник данных к основному документу. Оба принимают длинные списки параметров, но в каждом случае требуется только первый — имя файла заголовка/источника данных, включая путь. Вот пример:

```
oDocument.MailMerge.OpenHeaderSource( "C:\Temp\EmployeeList.DOC" )
```

Использование источника данных во время слияния



Источник заголовка позволяет вашим пользователям создавать свои собственные основные документы, используя список полей слияния. Источники заголовков не содержат данных, которые вам нужны, — вам нужна возможность создавать и присоединять полный источник данных на лету. Метод CreateDataSource позволяет создать новый источник данных. Как и в случае с CreateHeaderSource, вы создаете таблицу Word, а затем прикрепляете ее к основному документу. Большая часть работы довольно проста. Листинг 3, включенный как CreateDataSource.PRG в файлы загрузки для разработчиков, доступные по адресу www.hentzenwerke.com, создает и прикрепляет источник данных к документу. Он принимает те же три параметра, что и CreateHeaderSource в листинге 2. Поскольку самой медленной частью процесса является отправка фактических данных из VFP в Word, перед вызовом CreateDataSource обязательно создайте курсор или представление, содержащее только те данные, которые вам нужны для слияния почты. Выполните фильтрацию как полей, так и записей на стороне VFP. Метод EditDataSource открывает источник данных, связанный с основным документом. Если он уже открыт, он активирует его.

Листинг 3. Создайте лучший источник данных. Эта программа создает источник данных «на лету». Вместо того, чтобы иметь дело с ODBC, отправьте в источник данных Word только те записи и поля, которые вам нужны, когда вы действительно будете готовы выполнить слияние почты.

```
* CreateDataSource.PRG
* © 2000, Tamar E. Granor and Della Martin
* From: Microsoft Office Automation with Visual FoxPro
* Hentzenwerke Publishing. www.hentzenwerke.com

* Создать источник данных для текущего документа
```

```

* на основе таблицы или представления
* Предполагается:
*     Word открыт.

LPARAMETERS oDocument, cCursor, cDocument
* oDocument = документ, для которого должен быть создан источник заголовка.
* cCursor   = имя файла таблицы или представления, включая путь.
*           Данные уже должны быть отфильтрованы и отсортированы.
* cDocument = имя файла, включая путь, где
*           документ источника данных должен быть сохранен.

* Проверьте параметры.
IF PCOUNT() < 3
    MESSAGEBOX("Необходимо указать имя таблицы/представления и имя документа")
    RETURN .F.
ENDIF

IF VarType(oDocument) <> "O"
    MESSAGEBOX("Документ не указан")
    RETURN .F.
ENDIF

IF VarType(cCursor) <> "C" OR EMPTY(cCursor)
    MESSAGEBOX("Имя таблицы/представления должно быть символьным")
    RETURN .F.
ENDIF

IF VarType(cDocument) <> "C" OR EMPTY(cDocument)
    MESSAGEBOX("Имя документа должно быть символьным")
    RETURN .F.
ENDIF

LOCAL nFieldCount, cFieldList, aFieldList[1], nField
LOCAL oWord, oRange, oSourceDoc, oRow, oTable

* Получить ссылку на Word
oWord = oDocument.Application

* Открыть таблицу/представление
USE (cCursor) AGAIN IN 0 ALIAS MergeCursor
SELECT MergeCursor

* Получить список полей
nFieldCount = AFIELDS( aFieldList, "MergeCursor" )
* Пройдите по списку, создав строку, разделенную запятыми
cFieldList = ""
FOR nField = 1 TO nFieldCount
    IF aFieldList[ nField, 2 ] <> "G"
        * Не могу использовать поля General
        cFieldList = cFieldList + aFieldList[ nField, 1 ] + ","
    ENDIF
ENDFOR
cFieldList = LEFT( cFieldList, LEN(cFieldList) - 1 )

WITH oDocument.MailMerge
    * Прикрепите данные к документу
    .CreateDataSource( cDocument, , , cFieldList )
    .EditDataSource
    oSourceDoc = oWord.ActiveDocument
    oTable = oSourceDoc.Tables[1]
    oRow = oTable.Rows[1]

    * Теперь откройте источник данных и поместите данные в документ
    SCAN

```

```

WITH oRow
  FOR nField = 1 TO nFieldCount
    DO CASE
      CASE TYPE( FIELDS( nField )) = "C"
        * Избавьтесь от конечных пробелов
        .Cells[ nField ].Range.InsertAfter( TRIM( EVAL(FIELDS(nField))) )
      CASE TYPE( FIELDS( nField )) = "G"
        * Ничего не делайте
      OTHERWISE
        * Просто отправь как есть
        .Cells[ nField ].Range.InsertAfter( EVAL(FIELDS( nField )) )
      ENDCASE
    ENDFOR
  ENDWITH
  oRow = oTable.Rows.Add()
ENDSCAN
oRow.Delete()
oSourceDoc.Save()
ENDWITH

USE IN MergeCursor

RETURN

```

Выполнение слияния почты

После того, как вы преодолели все препятствия, чтобы получить данные, выполнение слияния почты становится самой простой частью. Просто вызовите метод Execute объекта MailMerge и – пуф! основной документ и источник данных объединяются в новый документ. Это все, что нужно:

```
oDocument.MailMerge.Execute()
```

Конечно, вы, вероятно, захотите осуществлять больший контроль над слиянием. Различные свойства объекта MailMerge позволяют настроить параметры перед вызовом Execute. Скорее всего, вам придется иметь дело с двумя – Destination и SuppressBlankLines. SuppressBlankLines – это логический параметр, указывающий, следует ли удалять полностью пустые строки документа. По умолчанию используется .T.

Назначение определяет, куда отправляются результаты слияния. По умолчанию используется wdSendToNewDocument (0), это наше предпочтение. Наш выбор – wdSendToPrinter (1), wdSendToEmail (2) и wdSendToFax (3). Существует несколько свойств, каждое из которых начинается с «Mail», посвященных деталям случая, когда результаты отправляются на электронную почту.

Определение типа документа

Объект MailMerge имеет еще несколько важных свойств. Состояние позволяет для любого документа проверить, какую роль он играет в процессе слияния почты. В таблице 3 показаны возможные значения состояния.

Таблица 3. Что я за документ? Свойство State MailMerge сообщает вам, с каким документом вы имеете дело.

Константа	Значение	Константа	Значение
wdNormalDocument	0	wdMainAndHeader	3
wdMainDocumentOnly	1	wdMainAndSourceAndHeader	4
wdMainAndDataSource	2	wdDataSource	5

MainDocumentType сообщает, какой у вас основной документ. В таблице 4 показаны значения, которые может принимать это свойство.

Таблица 4. Объединение типов документов. Свойство MainDocumentType классифицирует документы по различным типам документов слияния, которые вы можете создать.

Константа	Значение	Константа	Значение
wdNotAMergeDocument	-1	wdEnvelopes	2
wdFormLetters	0	wdCatalog	3
wdMailingLabels	1		

Спасение брошенных документов слияния почты

Вы можете столкнуться с основными документами, которые были оторваны от источников данных и к которым у вас нет доступа к соответствующим таблицам. Оказывается, довольно легко просмотреть эти документы и создать для них источник заголовков или источник данных.



Листинг 4 показывает ExtractDataSource.PRG (также включенный в файлы загрузки для разработчиков, доступные на сайте www.hentzenwerke.com), которая проходит по коллекции MailMergeFields документа для создания списка полей, а затем использует этот список для создания источника данных (хотя и пустого).

Листинг 4. Восстановление источников данных. Когда основные документы отсоединяются от источников данных, эта программа может создать пустой источник данных, который позволяет редактировать документ.

```
* ExtractDataSource.PRG
* © 2000, Tamar E. Granor and Della Martin
* From: Microsoft Office Automation with Visual FoxPro
* Hentzenwerke Publishing. www.hentzenwerke.com

* Обработка существующего документа и создание документа источника
* данных на основе полей, используемых в документе.

LPARAMETERS oDocument, cSourceName
* oDocument = существующий документ слияния почты.
* cSourceName = имя файла, включая путь, для источника данных.

#DEFINE wdNotAMergeDocument -1

* Следует проверить параметры здесь. Опущено из-за недостатка места.

WITH oDocument
  IF .MailMerge.MainDocumentType = wdNotAMergeDocument OR ;
    INLIST(.MailMerge.State, 0, 1)
    * Необходимо создать источник данных.
    * Пройдитесь по полям и создайте список.
    LOCAL oField, cCode, cField, cFieldList, cHeaderName

    cFieldList = ""
    FOR EACH oField IN .MailMerge.Fields
      cCode = oField.Code.Text
```

```

* Разобрать лишнюю информацию
cField = ALLTRIM( STRTRAN(STRTRAN(cCode, ;
    "MERGEFIELD",""), "\* MERGEFORMAT", "") )
IF NOT cField+" " $ cFieldList
    cFieldList = cFieldList + cField + " "
ENDIF
ENDFOR

IF LEN(cFieldList) > 1
    cFieldList = LEFT(cFieldList, LEN(cFieldList)-1) + ""
ENDIF

* Теперь создайте документ источника данных.
.MailMerge.CreateDataSource( cSourceName, , , cFieldList )
ENDIF
ENDWITH

RETURN

```

6.4 Собираем все вместе



Чтобы продемонстрировать главный урок этой главы, мы рассмотрим процесс, состоящий из двух частей. В листинге 5 показана программа (WordSample3Pt1.PRG в файлах загрузки для разработчиков, доступных по адресу www.hentzenwerke.com), которая создает шаблон информационных листов о продуктах для тасманских торговцев. Шаблон представляет собой основной документ слияния почты, прикрепленный только к источнику заголовка. Программа запускает запрос, который собирает необходимые данные (в реальном приложении у вас, вероятно, будет представление этих данных), а затем вызывает CreateHeaderSource.PRG (см. листинг 2), чтобы прикрепить заголовок. Затем она заполняет и сохраняет шаблон. На рисунке 10 показан готовый шаблон.

Листинг 5. Создание шаблона слияния почты. Эта программа генерирует как источник заголовка, так и основной документ, в данном случае шаблон основного документа.

```

* WordSample3Pt1.PRG
* © 2000, Tamar E. Granor and Della Martin
* From: Microsoft Office Automation with Visual FoxPro
* Hentzenwerke Publishing. www.hentzenwerke.com

```

```

* Это пример «Собираем все вместе» из главы 6.
* Создайте основной документ для карточек продукции.
* Документ создается как шаблон, чтобы его можно было
* затем использоваться с File|New.

```

```

#DEFINE CR CHR(13)
#DEFINE TAB CHR(9)
#DEFINE wdHeaderFooterPrimary 1
#DEFINE wdGray25 16
#DEFINE wdAlignParagraphCenter 1
#DEFINE wdCollapseEnd 0
#DEFINE wdParagraph 4
#DEFINE wdWord 2
#DEFINE wdLineStyleDouble 7

```

```

#DEFINE wdUserTemplatesPath      2
#DEFINE wdGoToBookmark           -1

LOCAL oWord, oDocument, oRange, oBorderRange, cTemplatePath

* Откройте Word и создайте новый документ-шаблон
oWord = CreateObject("Word.Application")
oDocument = oWord.Documents.Add(, .T.)

* Создать курсор всех товаров
OPEN DATABASE _SAMPLES + "TasTrade\Data\TasTrade"
SET ENGINEBEHAVIOR 70
SELECT product_name, english_name, category_name, ;
       quantity_in_unit, unit_price, ;
       company_name, contact_name, contact_title, ;
       address, city, region, postal_code, country, ;
       phone, fax ;
FROM products ;
   JOIN supplier ;
       ON products.supplier_id = supplier.supplier_id ;
   JOIN category ;
       ON products.category_id = category.category_id ;
ORDER BY Category_Name, Product_Name ;
INTO CURSOR ProductList

* Прикрепите источник заголовка к документу-шаблону.
DO CreateHeaderSource WITH oDocument, DBF("ProductList"), ;
                        AddBs(SYS(2023))+"ProdHeader.DOC"

USE IN ProductList

* Теперь настройте лист продукта
* Сначала назначьте шрифт Normal.

*WITH oDocument.Styles["Normal"].Font
WITH oDocument.Styles["Обычный"].Font
    .Name = "Times New Roman"
    .Size = 12
ENDWITH

* Добавить заголовок.
WITH oDocument.Sections[1].Headers[ wdHeaderFooterPrimary ]
    oRange = .Range()
    WITH oRange
        .Text = "Tasmanian Traders"
        * .Style = oDocument.Styles["Heading 1"]
        .Style = oDocument.Styles["Заголовок 1"]
        .ParagraphFormat.Alignment = wdAlignParagraphCenter
        .Shading.BackgroundPatternColorIndex = wdGray25
    ENDWITH
ENDWITH

* Заголовок страницы

```

```
oRange = oDocument.Range(0,0)
WITH oRange
    * .Style = oDocument.Styles[ "Heading 2" ]
    .Style = oDocument.Styles[ "Заголовок 2" ]
    .ParagraphFormat.Alignment = wdAlignParagraphCenter
    .InsertAfter("Product Information" + CR + CR )
    .Collapse( wdCollapseEnd )

    * Сначала добавьте фиксированный текст и установите
    * закладки там, где мы хотим разместить поля слияния.
    * Добавить категорию продукта.
    * .Style = oDocument.Styles[ "Heading 3" ]
    .Style = oDocument.Styles[ "Заголовок 3" ]
    .InsertAfter( "Product Category: " )
    .Collapse( wdCollapseEnd )
    oDocument.Bookmarks.Add( "ProductCategory", oRange )
    .InsertAfter( CR )
    .Expand( wdParagraph )
    .Borders.OutsideLineStyle = wdLineStyleDouble
    .Collapse( wdCollapseEnd )
    .InsertAfter( CR )

    * Добавить название продукта.
    .InsertAfter( "Product Name: " )
    .Collapse( wdCollapseEnd )
    oDocument.Bookmarks.Add("ProductName", oRange )
    .Collapse( wdCollapseEnd )
    .InsertAfter( CR )
    oBorderRange = oRange.Paragraphs[1].Range()
    .InsertAfter( "English Name: " )
    .Collapse( wdCollapseEnd )
    oDocument.Bookmarks.Add("EnglishName", oRange)
    .InsertAfter( CR )
    .Collapse( wdCollapseEnd )
    oBorderRange.MoveEnd( wdParagraph, 1)
    oBorderRange.Borders.OutsideLineStyle = wdLineStyleDouble

    * Теперь единицы измерения и цена
    * .Style = oDocument.Styles[ "Normal" ]
    .Style = oDocument.Styles[ "Обычный" ]
    .InsertAfter( CR + "Sold in units of: " )
    .Collapse( wdCollapseEnd )
    oDocument.Bookmarks.Add("Quantity", oRange )
    .InsertAfter( " at a price of: " )
    .Collapse( wdCollapseEnd )
    oDocument.Bookmarks.Add("UnitPrice", oRange)
    .InsertAfter( " per unit." + CR + CR )
    .Collapse( wdCollapseEnd )

    * Теперь информация о поставщике
    * Чтобы выровнять элементы, нам понадобится вкладка,
    * поэтому настройте ее.
    WITH oDocument.Paragraphs.TabStops
```

```

        .ClearAll()
        .Add( oWord.InchesToPoints( 1 ) )
    ENDWITH

    .InsertAfter( "Supplier: " + TAB )
    .Collapse( wdCollapseEnd )
    oDocument.Bookmarks.Add("CompanyName", oRange)
    .InsertAfter( CR + TAB )
    .Collapse( wdCollapseEnd )
    oDocument.Bookmarks.Add("Address", oRange)
    .InsertAfter( CR + TAB )
    .Collapse( wdCollapseEnd )
    oDocument.Bookmarks.Add("City", oRange)
    .InsertAfter( CR + TAB )
    .Collapse( wdCollapseEnd )
    oDocument.Bookmarks.Add("Region", oRange)
    .InsertAfter( CR + TAB )
    .Collapse( wdCollapseEnd )
    oDocument.Bookmarks.Add("PostalCode", oRange)
    .InsertAfter( CR + TAB )
    .Collapse( wdCollapseEnd )
    oDocument.Bookmarks.Add("Country", oRange)
    .InsertAfter( CR )
    .InsertAfter( "Contact: " + TAB )
    .Collapse( wdCollapseEnd )
    oDocument.Bookmarks.Add("ContactName", oRange)
    .InsertAfter( CR + TAB )
    .Collapse( wdCollapseEnd )
    oDocument.Bookmarks.Add("ContactTitle", oRange)
    .InsertAfter( CR )
    .InsertAfter( "Phone: " + TAB )
    .Collapse( wdCollapseEnd )
    oDocument.Bookmarks.Add("Phone", oRange )
    .InsertAfter( CR )
    .InsertAfter( "Fax: " + TAB )
    .Collapse( wdCollapseEnd )
    oDocument.Bookmarks.Add("Fax", oRange)
    .InsertAfter( CR )

```

* Теперь вставьте поле слияния в каждую закладку.

```

oRange = oDocument.Bookmarks["ProductCategory"].Range()
oDocument.MailMerge.Fields.Add( oRange, "Category_Name")

```

```

oRange = oDocument.Bookmarks["ProductName"].Range()
oDocument.MailMerge.Fields.Add( oRange, "Product_Name")

```

```

oRange = oDocument.Bookmarks["EnglishName"].Range()
oDocument.MailMerge.Fields.Add( oRange, "English_Name")

```

```

oRange = oDocument.Bookmarks["Quantity"].Range()
oDocument.MailMerge.Fields.Add( oRange, "Quantity_In_Unit")

```



```
oRange = oDocument.Bookmarks["UnitPrice"].Range()
oDocument.MailMerge.Fields.Add( oRange, "Unit_Price")

oRange = oDocument.Bookmarks["CompanyName"].Range()
oDocument.MailMerge.Fields.Add( oRange, "Company_Name")

oRange = oDocument.Bookmarks["Address"].Range()
oDocument.MailMerge.Fields.Add( oRange, "Address")

oRange = oDocument.Bookmarks["City"].Range()
oDocument.MailMerge.Fields.Add( oRange, "City")

oRange = oDocument.Bookmarks["Region"].Range()
oDocument.MailMerge.Fields.Add( oRange, "Region")

oRange = oDocument.Bookmarks["PostalCode"].Range()
oDocument.MailMerge.Fields.Add( oRange, "Postal_Code")

oRange = oDocument.Bookmarks["Country"].Range()
oDocument.MailMerge.Fields.Add( oRange, "Country")

oRange = oDocument.Bookmarks["ContactName"].Range()
oDocument.MailMerge.Fields.Add( oRange, "Contact_Name")

oRange = oDocument.Bookmarks["ContactTitle"].Range()
oDocument.MailMerge.Fields.Add( oRange, "Contact_Title")

oRange = oDocument.Bookmarks["Phone"].Range()
oDocument.MailMerge.Fields.Add( oRange, "Phone")

oRange = oDocument.Bookmarks["Fax"].Range()
oDocument.MailMerge.Fields.Add( oRange, "Fax")
```

ENDWITH

```
* Шаблон создаётся в
* "C:\Dokumente und
Einstellungen\User\Anwendungsdaten\Microsoft\Шаблоны"\
cTemplatePath = oWord.Options.DefaultFilePath(wdUserTemplatesPath)
oDocument.SaveAs( AddBs(cTemplatePath) + "ProdInfo")
```

oWord.Quit()

RETURN

Product Information

Product Category: «Category_Name»

Product Name: «Product_Name»

English Name: «English_Name»

Sold in units of: «Quantity_In_Unit» at a price of: «Unit_Price» per unit.

Supplier: «Company_Name»

«Address»

«City»

«Region»

«Postal_Code»

«Country»

Contact: «Contact_Name»

«Contact_Title»

Phone: «Phone»

Fax: «Fax»

Рисунок 10. Создание документов слияния почты. Этот шаблон был создан в листинге 5. Он имеет источник заголовка и основан на запросе из базы данных TasTrade.



Вторая часть процесса – создать реальный источник данных, когда вы будете готовы выполнить слияние почты. В листинге 6 показан код (WordSample3Pt2.PRG в файлах загрузки для разработчиков, доступных по адресу www.hentzenwerke.com), который создает новый документ на основе шаблона, вызывает CreateDataSource.PRG (см. листинг 3), затем выполняет слияние и показывает результат. На рисунке 11 показана часть результата.

Листинг 6. Выполнение слияния. Этот код использует шаблон, созданный в листинге 5, для создания нового документа, создания источника данных и выполнения слияния.

```
* WordSample3Pt2.PRG
* © 2000, Tamar E. Granor and Della Martin
* From: Microsoft Office Automation with Visual FoxPro
* Hentzenwerke Publishing. www.hentzenwerke.com
```

```
* Это пример «Собираем все вместе» из главы 6.
* Создайте листы с информацией о продукте на основе
* шаблона, используя слияние писем.
```

```
#DEFINE wdUserTemplatesPath 2
```

```
#DEFINE wdWindowStateMaximize 1
```

```
LOCAL cTemplatePath, oDocument, oMergedDocument
```

```
* Создайте экземпляр Word.
```

```
* Сделайте его общедоступным для демонстрационных целей.
```

```
RELEASE ALL LIKE o*
```

```
PUBLIC oWord
```

```
oWord = CreateObject("Word.Application")
```

* Сделайте Word видимым.

oWord.Visible = .t.

* Создайте новый документ на основе шаблона.

cTemplatePath = oWord.Options.DefaultFilePath(wdUserTemplatesPath)

oDocument = oWord.Documents.Add(ADDBS(cTemplatePath) + "ProdInfo")

* Запустите запрос, чтобы создать курсор всех продуктов.

* Создать курсор всех товаров

OPEN DATABASE _SAMPLES + "TasTrade\Data\TasTrade"

SET ENGINEBEHAVIOR 70

```
SELECT product_name, english_name, category_name, ;
       quantity_in_unit, TRANSFORM(unit_price, "@$") AS unit_price, ;
       company_name, contact_name, contact_title, ;
       address, city, region, postal_code, country, ;
       phone, fax ;
FROM products ;
  JOIN supplier ;
    ON products.supplier_id = supplier.supplier_id ;
  JOIN category ;
    ON products.category_id = category.category_id ;
ORDER BY Category_Name, Product_Name ;
INTO CURSOR ProductList
```

* Теперь создайте и прикрепите источник данных.

```
DO CreateDataSource WITH oDocument, DBF("ProductList"), ;
                    AddBs(SYS(2023)) + "ProdData"
```

USE IN ProductList

* Выполните слияние почты.

oDocument.MailMerge.Execute()

oMergedDocument = oWord.ActiveDocument

WITH oMergedDocument

```
  IF .ActiveWindow.WindowState <> wdWindowStateMaximize
```

```
    * Переместите его, чтобы он был видимым - он
```

```
    * почему-то всплывает за пределами экрана.
```

```
    .ActiveWindow.Left = 0
```

```
    .ActiveWindow.Top = 0
```

```
  ENDIF
```

```
  * Предварительный просмотр
```

```
  .PrintPreview()
```

ENDWITH

RETURN

Tasmanian Traders	
<i>Product Information</i>	
Product Category: Beverages	
Product Name: Chang	
English Name: Tibetan Barley Beer	
Sold in units of: 24 - 12 oz bottles at a price of: \$19.0000 per unit.	
Supplier:	Exotic Liquids 49 Gilbert St. London EC1 4SD UK
Contact:	Charlotte Cooper Purchasing Manager
Phone:	(71) 555-2222
Fax:	

Рисунок 11. Результаты слияния почты. Это информационный листок продукта, созданный программами в листингах 5 и 6. Для каждого продукта имеется один лист.

Нам хотелось бы сказать: «Вот и все, ребята!» Но правда в том, что даже несмотря на то, что ему посвящены три полных главы, Word содержит гораздо больше, чем мы смогли охватить здесь. Итак, если вы хотите что-то сделать в документе, а мы не показали вам, как это сделать, просто покопайтесь и попробуйте. Помните ключи к пониманию этого. Во-первых, попробуйте сделать это в интерактивном режиме. Попробуйте записать макрос. Используйте файл справки Word VBA и обозреватель объектов, чтобы узнать, какие объекты, методы и свойства задействованы. Объектная модель Word чрезвычайно богата. Если вы можете себе это представить, то, вероятно, это осуществимо.

Глава 7. Основы Excel

Excel имеет некоторые возможности, которых нет в FoxPro. Хотите табличный отчет, которым может манипулировать обычный конечный пользователь? Попробуйте Эксель. Нужны математические расчеты, которых нет в FoxPro? Excel может это сделать. Хотите немного графиков? Excel имеет отличный графический движок.

Работа с Excel посредством Автоматизации имеет много общего с Автоматизацией Word, связанной с полиморфизмом. К сожалению, различий больше, чем сходства, поскольку два сервера имеют разные возможности и цели.

7.1 Объектная модель

Как и Word, объект верхнего уровня Excel – это объект Application, свойства и методы которого относятся к Excel в целом. Числа, такие как StartupPath, Version и WindowState, одинаковы, а другие, например Calculation, определяющие время выполнения вычислений, относятся к Excel. Объект Application также обеспечивает доступ ко всем другим объектам Excel.

Основным объектом Excel является Workbook (Рабочая книга). Это соответствует файлу XLS. Объект приложения Excel отслеживает все открытые книги в коллекции Workbooks и использует свойство ActiveWorkbook для возврата ссылки на активный объект Workbook.

Каждая книга имеет две основные коллекции: Worksheet и Chart (Рабочие листы и Диаграммы), которые представляют страницы книги и содержащиеся в ней графики соответственно. Рабочая книга имеет свойства ActiveSheet и ActiveChart, содержащие ссылки на текущие объекты Worksheet и Chart. Excel также обеспечивает быстрый доступ, предлагая свойства ActiveSheet, ActiveChart и ActiveCell на уровне приложения.

Как и в Word, файл справки Excel Visual Basic содержит живую диаграмму объектной модели. На рисунке 1 показана часть диаграммы объектной модели, описывающая объект Worksheet.

7.2 Дежавю

Если вы прочитаете эту книгу от начала до конца, читая эту главу, вы скажете себе: «Боже, я уже читал это раньше!» Office 2000 является объектно-ориентированным и полиморфным. Полиморфизм буквально означает «множество форм», а применительно к ООП это означает, что разные объекты имеют свойства и методы, которые ведут себя согласованно. Чтобы записать файл на диск, вы используете метод Save, независимо от того, работаете ли вы в Word, Excel, PowerPoint или Outlook. Это не означает, что команды Save выполняют одно и то же: при сохранении документа Word должны выполняться другие действия, чем при сохранении электронной таблицы Excel. Для каждого могут даже быть разные параметры. Однако вы можете быть уверены, что команда Save сохранит вашу работу на диск.

Преимущество полиморфизма в том, что если вы знаете, как сделать что-то в одном инструменте, вы будете знать, как сделать это и в остальных. В Office это обычно так. Однако полиморфизм включает в себя способность методов принимать разные параметры, поскольку разные объекты совершенно разные. Поэтому вам необходимо знать, что синтаксис может меняться (и меняется) в разных приложениях Office.

Microsoft Excel Objects (Worksheet)

See Also

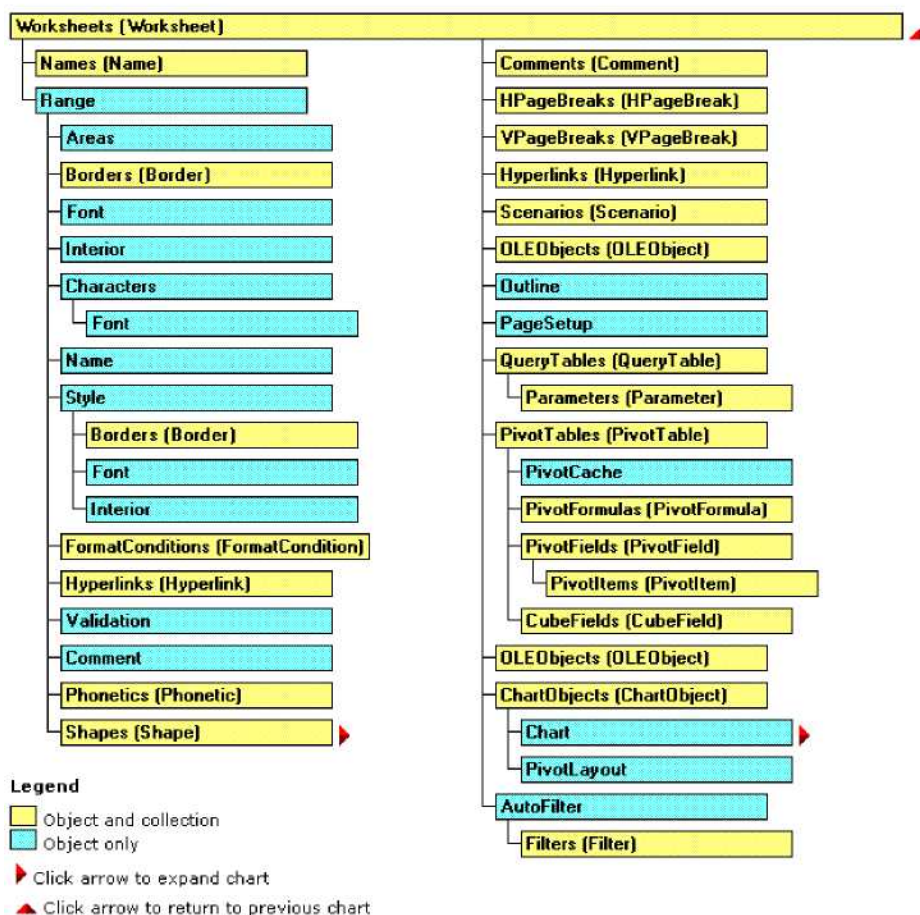


Рисунок 1. Объектная модель Excel. Эта диаграмма доступна в файле справки и показывает иерархию доступных объектов.

Между приложениями Office так много общего, что мы могли бы написать в этой главе что-то вроде: «Просто используйте функцию CreateObject(), как описано в главе Word, но вместо этого используйте «Excel.Application». Мы посчитали, что было бы лучше иметь полное объяснение каждого приложения, чтобы вам не приходилось листать книгу назад и вперед. Это также упрощает объяснение тонких синтаксических различий, существующих между приложениями. После того, как мы преодолеем основы открытия и сохранения файлов, мы перейдем к достаточному количеству специфичных для приложения функций, поэтому это покажется менее повторяющимся.

Преимущество полиморфизма в том, что если вы знаете, как сделать что-то в одном объекте, вы сможете сделать это и во всех из них. Однако объяснение схожих концепций для каждого объекта приводит к избыточности текста. Просто думайте об этом избыточном тексте как об «особенности».

7.3 Получение информации о приложении

Чтобы открыть Excel с помощью Автоматизации, используйте следующую команду (при необходимости введите ее в командном окне):

```
oExcel = CreateObject("Excel.Application")
```

Ничего не произошло! Проверьте диспетчер задач; Excel запущен и работает, но его просто не видно. Как и другие приложения Office, Excel скрывается при создании его экземпляра с помощью Автоматизации, и вы сами можете указать ему, когда он станет видимым. Это мощная функция, автоматизирующая Excel без

необходимости обновления экрана, которая может сократить время выполнения на 10-30 процентов! Когда вы будете готовы показать Excel своим пользователям, установите свойство Visible приложения:

```
oExcel.Visible = .T.
```

Если вы будете следовать инструкциям в командном окне FoxPro, вы увидите, что Excel виден, но электронной таблицы нет; просто серый экран. Когда пользователь запускает Excel в интерактивном режиме, предполагается, что он хочет сразу перейти к пустому листу и начать печатать. Автоматизация ничего не предполагает! У вас есть полный контроль.



Во всех примерах Excel предполагается, что вы уже создали серверный объект Excel, доступ к которому осуществляется через переменную oExcel.

7.4 Управление рабочими книгами и рабочими листами

Настало время для некоторых разъяснений, которые, кажется, заинтересуют новых программистов Автоматизации: объект коллекции обозначается существительным во множественном числе, а объект в коллекции – это то же существительное в единственном числе. Workbooks (с буквой «s») – это коллекция всех открытых объектов Workbook (без «s»). Объект Workbook (без «s») соответствует файлу XLS. В каждой книге есть коллекция под названием Worksheets (с буквой «s»). Каждый Worksheet (без буквы «s») отображается в виде вкладки с заголовками вкладок в нижней части окна.

Открытие новой книги

Поскольку Excel открывается без доступной книги, вам необходимо добавить новую или открыть существующую. Чтобы создать новый объект Workbook, используйте метод Add коллекции Workbooks. Количество листов, содержащихся в рабочей книге при добавлении, контролируется свойством SheetsInNewWorkbook объекта Application (задается вручную в Excel, выбрав в меню «Сервис | Параметры», выбрав вкладку «Общие» и используя счетчик «Листов в новой книге»). Установка этого свойства влияет на все новые книги, добавленные после изменения (не влияет на уже открытые). Выполнение следующей команды перед открытием новой книги дает четыре листа:

```
oExcel.SheetsInNewWorkbook = 4
```

Поскольку этим свойством управляют пользователи, подготовьте свой код для обработки различий в этом параметре. Не думайте, что пользователь не играл со значениями по умолчанию; используйте свойство SheetsInNewWorkbook, чтобы убедиться, что у вас достаточно листов.

Откройте новую книгу, используя метод Add. Он принимает один необязательный параметр, который указывает файл шаблона.

```
oWorkbook = oExcel.Workbooks.Add()
```

Переменная oWorkbook указывает на вашу новую книгу. Существует несколько других способов ссылки на этот объект Workbook. Один из них – использовать свойство ActiveWorkbook приложения: oExcel.ActiveWorkbook. Каждый раз, когда вы добавляете новую книгу, она автоматически становится активной. Другой способ получить доступ к этой книге – через ее индекс в коллекции Workbooks. Это первая открытая книга, поэтому доступ к ней можно получить как oExcel.Workbooks[1].

Коллекция `Workbooks` предоставляет альтернативную ссылку, которая представляет собой имя книги (соответственно сохраненное в свойстве `Name` объекта `Workbook`).

Имя книги по умолчанию – «Книга», объединенное с номером, соответствующим порядку ее открытия. «Книга1» – это свойство `Name` первого объекта `Workbook`. Свойство `Name` доступно только для чтения. Единственный способ установить его – использовать метод `SaveAs` (см. раздел «Сохранение книги» далее в этой главе).

Открытие существующей книги

Существует два способа открыть существующую книгу. Метод `Open` предполагает, что у вас уже есть ссылка на объект приложения `Excel`, и он откроет существующий файл. Функция `GetObject()` `Visual FoxPro` дает вам больший контроль над видимостью книги и не требует ссылки на объект приложения `Excel`.

Метод `Open` коллекции `Workbooks` открывает существующую книгу по имени файла:

```
oWorkbook = oExcel.Workbooks.Open("c:\my document\sample.xls")
```

Имя файла – единственный обязательный параметр для метода `Open`. Есть еще 12 параметров (к счастью, все необязательные), позволяющие контролировать необходимость обновления ссылок, указывать пароль, описывать макет импортируемого текстового файла и другие функции. Мы приглашаем вас просмотреть файл справки для получения дополнительной информации об этих функциях.

Альтернативный способ открыть книгу – использовать функцию `GetObject()`. `GetObject()` имеет две формы. В первом вы передаете ей имя файла, который нужно открыть:

```
oWorkbook = GetObject("c:\my document\sample.xls")
```

`GetObject()` возвращает ссылку на объект `Workbook` (а не на объект `Application`, как в случае `CreateObject()`). Если `Excel` не запущен (или запущен, но не виден), вам необходимо сделать приложение видимым, чтобы его увидеть. Чтобы перейти к объекту `Application`, используйте свойство `Application` `Workbook`, которое ссылается на объект `Application`:

```
oWorkbook.Application.Visible = .T.
```

Новая книга по-прежнему не видна. Помните, Автоматизация ничего не предполагает! Объект `Workbook` доступен для Автоматизации; его окно скрыто. Опять же, есть преимущество в производительности при запуске всех команд Автоматизации и их последующем отображении. Чтобы увидеть окно, используйте метод `Window Activate` (в примере предполагается, что окно, которое вы повторно активируете, является первым открытым окном):

```
oWorkbook.Windows[1].Activate()
```

Бывают случаи, когда вы хотите скрыть окно во время создания книги. Предполагая, что это активное окно, выполните метод `oWorkbook.ActiveWindow.Close`.

Функция `GetObject()` имеет альтернативный синтаксис, который позволяет предотвратить многократное появление объекта. Если вы программируете Office 97, это может быть важной проблемой. Возможно, вы захотите использовать уже открытый экземпляр `Excel`, чтобы не использовать больше памяти для открытия другого экземпляра. В Office 2000 одна из новых функций заключается в том, что существует только один экземпляр приложения; само приложение Office обеспечивает лучшее управление памятью. Однако синтаксис `GetObject()` по-прежнему работает для `Excel 2000` (и открывает его, если он еще не открыт). Выполнение следующей команды открывает `Excel` и возвращает ссылку на

приложение:

```
oExcel = GetObject("", 'Excel.Application')
```

Обратите внимание, что синтаксис открытия имени файла возвращает ссылку на книгу.



Если нет запущенных экземпляров Excel 97, вызов `GetObject('Excel.Application')` приводит к следующей ошибке: «Код ошибки OLE 0x800401e3: операция недоступна». См. раздел «Предварительный просмотр» далее в этой главе, где приведен пример определения того, запущен ли Excel.

Сохранение книги

Просматривая список методов объекта `Workbook`, вы находите ссылки на `Save` и `SaveAs`. Можно подумать, что метод `Save` будет использоваться для сохранения любой старой книги, а метод `SaveAs` сохранит ее в другом формате. Однако в Excel (и PowerPoint, но не в Word) метод `Save` не принимает параметров. Он использует имя файла по умолчанию, состоящее из следующего:

- **Местоположение файла по умолчанию.** Пользователи могут изменить это в пункте меню «Сервис|Параметры...» на вкладке «Общие», указав расположение файла по умолчанию. В коде Автоматизации вы проверяете (или устанавливаете) свойство `DefaultFilePath` объекта `Application`. Этот параметр сохраняется при закрытии Excel; обязательно восстановите настройки пользователя, если вы их изменили.
- **Свойство `Name Workbook`.** Помните, что по умолчанию это «КнигаX» и доступно только для чтения.
- **Расширение XLS.**

Итак, по умолчанию метод `Save` сохраняет электронную таблицу как нечто вроде «C:\My Documents\Book1.XLS». Чтобы дать ей какое-то осмысленное имя, используйте метод `SaveAs`, который позволяет указать полное имя файла. `SaveAs` также устанавливает для свойства `Name` часть имени файла (а не путь или расширение).

`SaveAs` принимает множество дополнительных параметров для таких вещей, как пароли, добавление файла в список «Последние использованные», резервные копии и т. д. См. файл справки для этих параметров.

Самый важный параметр – первый, который определяет имя файла. Не забудьте полностью указать путь; если вы этого не сделаете, будет использовано расположение файла по умолчанию.

```
oWorkbook.SaveAs('c:\ExcelData\FirstTest.XLS')
```

Необязательный второй параметр – это формат файла. Если он не указан, по умолчанию используется используемая версия Excel (или, если файл был сохранен ранее, формат, в котором он был сохранен). В файле справки перечислен 41 тип (для Office 2000), включая несколько версий для Lotus, dBASE и Excel; HTML; DIF, SYLK и CSV и другие. Для этих файлов требуются конвертеры, установленные вместе с Office; если пользователь решил не устанавливать конкретный преобразователь, ваш обработчик ошибок должен быть готов обработать возникшую ошибку. Некоторые форматы файлов сохраняют только текущий лист; другие сохраняют всю книгу. Файл справки Excel содержит отличную информацию, если вы посмотрите по ключевому слову «Save_As». В частности, в теме

«Конвертеры форматов файлов, поставляемые с Microsoft Excel» содержит ценная информация о каждом из форматов файлов. В файле справки VBA Excel есть особенно полезная тема «Сохранение документов в виде веб-страниц».

Как избежать пользовательских диалогов Excel

При использовании SaveAs сохранение в уже существующий файл вызывает диалоговое окно пользователя, показанное на рисунке 2 (конечно, если у пользователя включен Office Assistant, это объяснит ошибку). Если приложение не отображается, появится всплывающее окно с сообщением. Если есть видимое окно, будут мигать строка заголовка окна и кнопка на панели задач. Когда окно Excel свернуто (или скрыто), а панель задач находится в режиме автоматического скрытия (когда вам нужно провести мышкой вниз по экрану), пользователю кажется, что у вас зависание. Если пользователь видит мигание и выбирает Excel, а затем нажимает «Нет» или «Отмена», возникает следующая ошибка: «Код исключения OLE IDispatch 0 из Microsoft Excel: невозможно получить свойство SaveAs класса Workbook». Будьте готовы справиться с этой ошибкой (#1428). А еще лучше предотвратить возникновение этой ошибки.

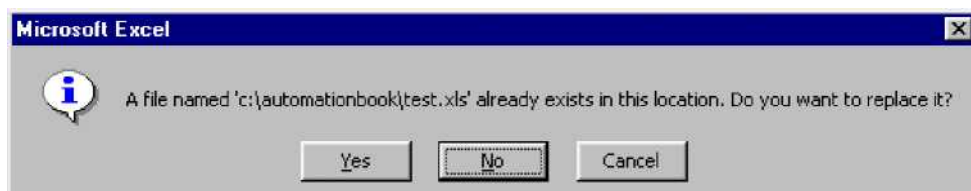


Рисунок 2. Диалоговое окно, вызываемое при перезаписи файла. Обратите внимание, что у пользователя есть три варианта выбора, и только «Да» позволяет программе работать без ошибок.

Есть несколько шагов, которые вы можете предпринять, чтобы предотвратить эту ошибку. Сначала проверьте, отличается ли oWorkbook.Name от первого имени файла (без пути и расширения). Используйте функцию JUSTSTEM() VFP, чтобы вернуть первое имя файла. Если они различаются, используйте свойство SaveAs, чтобы задать имя файла (а затем для свойства Name задается первое имя файла). Если они одинаковы, используйте команду Save, которая предполагает, что вы хотите перезаписать файл. Во-вторых, прежде чем использовать функцию SaveAs, убедитесь, что рассматриваемое имя файла не существует:

```
MyFile = "c:\ExcelData\MyFile.XLS"
```

* Используйте JUSTFNAME() для вычисления этого значения

* в VFP 6.0 или функцию FoxTools в предыдущих версиях

```
MyFileJustName = JustFName(MyFile)
```

* Определите, следует ли использовать SaveAs или Save

```
IF oWorkbook.Name <> MyFileJustName
```

```
    * ЕСЛИ файл уже существует, удалите его
```

```
    IF FILE(MyFile)
```

```
        * Предупреждение: возможно, вы удаляете файл,
```

```
        * который не имеет никакого отношения к данной
```

```
        * ситуации. Убедитесь, что ваше приложение знает об этом.
```

```
        ERASE (MyFile)
```

```
    ENDIF
```

```
    * Сохраните его, не опасаясь диалогового окна пользователя.
```

```
oWorkbook.SaveAs(MyFile)
```

```
ELSE
```

```
* Сохраните его, поскольку он уже был сохранен с помощью
* SaveAs.
oWorkbook.Save()
```

```
ENDIF
```

Работа с рабочими листами

Каждая рабочая книга (Workbook) может содержать множество рабочих листов (Worksheet) в коллекции рабочих листов (Worksheets) рабочей книги (Workbook). Рабочие листы хранят данные. Объект коллекции Worksheets (это множественное число, поэтому это коллекция) хранит столько объектов Worksheet (единственное число), сколько Excel может обработать (в Excel 2000 количество листов ограничено доступной памятью). Как и в случае с другими объектами в коллекциях, существует несколько способов вернуть ссылку на рабочий лист (в рабочей книге должен быть хотя бы один рабочий лист). Существует свойство ActiveSheet Workbook: oWorkbook.ActiveSheet. В коллекции есть индекс: oWorkbook.Worksheets[1]. Далее следует имя рабочего листа, которое по умолчанию равно «Лист» плюс номер, соответствующий его индексу при открытии: oWorkbook.Worksheets[«Лист1»]. Свойство Name листа доступно для чтения и записи, поэтому вы можете задать имя, которое оценят пользователи, поскольку имя отображается на вкладке внизу листа.

Вы также можете получить ссылку при добавлении листа. Здесь усердно работает полиморфизм; Я уверен, что вы уже догадались, что используете метод Add. Синтаксис немного другой:

```
oWorksheet = oWorkbook.Worksheets.Add([oBefore |, oAfter],
                                         [nCount], [nType])
```

oBefore	Object	Ссылка на объект рабочего листа, перед которым добавляется новый лист. (Необязательный)
oAfter	Object	Ссылка на объект рабочего листа, после которого добавляется новый лист. (Необязательный)
nCount	Numeric	Количество листов, которые необходимо добавить. (По умолчанию = 1) (Необязательно)
nType	Numeric	Одна из констант типа листа. Если вы не пишете макросы Excel 4, единственной применимой константой является xlWorksheet (-4167), которая используется по умолчанию. (Необязательный)

Если никакие параметры не передаются, новый лист помещается непосредственно перед ActiveSheet. Вы не можете указать одновременно объект oBefore и объект oAfter, поэтому просто передайте пустой параметр для того, который вы не используете. VBA допускает именованные параметры, что значительно упрощает использование такого синтаксиса: параметр выглядит примерно так:

Before:=oMySheet. Этот синтаксис позволяет легко запомнить передачу только одного параметра. Но VFP не поддерживает именованные параметры, поэтому мы должны указывать их в указанном порядке. В следующем примере кода показано, как добавить один лист перед первым листом, а затем еще один после первого:

* Получить ссылку на первый рабочий лист.

* Предполагается, что oWorkbook указывает на рабочую книгу

```
oWorksheet1 = oWorkbook.Worksheets[1]
```

* Добавьте лист перед первым листом.

```
oWorksheet2 = oWorkbook.Worksheets.Add(oWorksheet1)
```

* Добавляйте лист за ним. Обратите внимание на использование

* пустого первого параметра.

```
oWorksheet3 = oWorkbook.Worksheets.Add( , oWorksheet1)
```

Добавление третьего параметра указывает, сколько листов вы хотите добавить. Если вы хотите добавить три листа перед первым, введите следующую команду:

```
oWorkbook.Worksheets.Add(oWorksheet1, , 3)
```

Обратите внимание, что второй параметр пуст. Четвертый параметр, nType, предназначен для обратной совместимости. Для простоты опустите четвертый параметр.

Строки, столбцы, ячейки и диапазоны

Рабочий лист состоит из ячеек, адреса которых имеют вид X9, где X – одна или две буквы, обозначающие столбец, а 9 – одна или несколько цифр, обозначающих строку. Верхняя левая ячейка листа – A1. 29-я ячейка в 32-ом столбце – AF29.

Группа ячеек называется диапазоном. Обычно диапазон представляет собой прямоугольную группу смежных ячеек, например от D2 до F8. Диапазон также может представлять собой одну ячейку. Используя метод Union, вы можете объединить диапазоны. Диапазон на самом деле является объектом, и доступ к нему осуществляется через свойство Range рабочего листа.

Чтобы указать диапазон, вы используете свойство Range, чтобы указать адреса границ диапазона. Например, чтобы получить доступ к данным в ячейке C5, используйте:

```
? oExcel.ActiveSheet.Range("C5").Value
```

Чтобы создать диапазон, содержащий прямоугольный диапазон ячеек от D12 до F19, используйте:

```
oRange = oExcel.ActiveSheet.Range("D12:F19")
```

Другой способ доступа к ячейкам – использование свойств Rows (Строки), Columns (Столбцы) и Cells (Ячейки) Worksheet и Range. Эти свойства принимают соответствующие значения индекса и возвращают диапазон, содержащий указанные ячейки. Чтобы проверить значение всех ячеек в третьей строке диапазона, вы можете написать:

```
FOR nColumn = 1 TO oRange.Columns.Count
    ? oRange.Cells[3, nColumn].Value
ENDFOR
```

Чтобы просмотреть содержимое всех ячеек диапазона, используйте:

```
FOR nRow = 1 TO oRange.Rows.Count
    FOR nColumn = 1 TO oRange.Columns.Count
        ? oRange.Cells[nRow, nColumn].Value
    ENDFOR
ENDFOR
```

Обратите внимание, что индексы ячеек указывают сначала строку, а затем столбец (так же, как это делают массивы в VFP), но в адресах ячеек сначала указывается столбец.

Условно говоря

До сих пор мы использовали *абсолютные* адреса, то есть адрес ячейки относится к первой ячейке электронной таблицы. Мы также можем использовать *относительную* адресацию, которая основана на расстоянии от указанной начальной точки. Чтобы создать диапазон относительно другого диапазона, используйте свойство Offset. В этом примере создается диапазон на 20 строк ниже и на 30 колонок правее oRange. Новый диапазон имеет тот же размер и форму:

```
oRange2 = oRange.Offset(20, 30)
```

Множество диапазонов

Диапазоны также не обязательно должны состоять из одного прямоугольника. При создании диапазона можно указать несколько групп:

```
oRangeMixed = oExcel.ActiveSheet.Range("F21:F30, H21:H30")
```

Метод Union объединяет несколько диапазонов в один. Здесь два диапазона oRange и oRange2 объединены в один диапазон, на который ссылается oBigRange:

```
oBigRange = oExcel.Union(oRange, oRange2)
```

Обход диапазона с помощью цикла, подобного предыдущему, не работает для диапазона, состоящего из несмежных ячеек. В коллекции Areas есть запись для каждой прямоугольной части (называемой *областью*) диапазона, поэтому для обхода всех ячеек диапазона, независимо от того, прямоугольный он или нет, вы можете использовать такой код:

```
FOR nArea = 1 TO oRange.Areas.Count
  FOR nRow = 1 TO oRange.Areas[nArea].Rows.Count
    FOR nColumn = 1 TO oRange.Areas[nArea].Columns.Count
      ? oRange.Areas[nArea].Cells[nRow, nColumn].Value
    ENDFOR
  ENDFOR
ENDFOR
```

7.5 Обработка значений и формул

Как показывают предыдущие примеры, вы можете получить доступ к содержимому ячейки через ее свойство Value. Но при работе с электронными таблицами нас не всегда интересует значение ячейки. Чтобы получить доступ к формуле, содержащейся в ячейке, используйте свойство Формула:

```
? oExcel.ActiveSheet.Range("C22").Formula
```

Если ячейка содержит только значение, Formula возвращает значение в виде строки, а Value возвращает числа в виде чисел. Если в ячейке есть реальная формула, она возвращается в том формате, который вы использовали для ее ввода в Excel, начиная с "=".

Вы можете задать значения и формулы, назначив их соответствующим ячейкам:

```
oExcel.ActiveSheet.Range("C13").Value = 100
oExcel.ActiveSheet.Range("C22").Formula = "=SUM(C5:C20)"
```

Добавление, копирование и перемещение данных (включая формулы)

Хотя таблицы и представления VFP можно скопировать в формат XLS с помощью команды COPY TO или EXPORT, Автоматизация обеспечивает большую гибкость процесса, включая возможность помещения данных из нескольких таблиц на лист.

Заполнение листа по ячейкам

Автоматизация позволяет вводить значения данных ячейку за ячейкой. Обычный метод – настроить цикл SCAN и установить для каждой ячейки значения, соответствующие данным в каждой записи. Задайте для объекта Range только те ячейки, которые будут заполнены для первой записи, и используйте коллекцию Columns, чтобы установить значения ячеек. Непосредственно перед циклом SCAN используйте метод Offset, чтобы переместить диапазон на одну строку вниз. Метод Offset позволяет перемещать диапазон для каждой итерации без необходимости отслеживать строку, например:

SCAN

* Выполните обработку и форматирование здесь

* Переместить диапазон вниз на одну строку

oRange = oRange.Offset(1, 0)

ENDSCAN

Вставка данных с использованием этого метода чрезвычайно гибка, поскольку вы можете заполнять электронную таблицу без необходимости создавать главный курсор или таблицу, как при COPY TO или EXPORT. Вы имеете полный контроль над ячейками и можете условно выбирать значения ячеек на основе данных. Это отличная замена необходимому вам отчету, для которого требуется четыре разных диапазона условных подробностей (если средство создания отчета поддерживает только один) или несколько столбцов, в которых перечислены дочерние записи из разных дочерних таблиц (поскольку мы не можем установить SKIP TO для нескольких таблиц в один раз).

Когда Excel извлекает XLS-файл, созданный на основе таблицы или представления, в заголовках столбцов автоматически устанавливаются имена полей. Пользователи могут оценить «Номер заказа», «Дата» и «Сумма» вместо «cOrd_ID», «dOrd_Date» и «nOrd_Total». Явная установка заголовков столбцов (с использованием свойства Value каждой ячейки) необходима независимо от того, извлекаете ли вы XLS или добавляете данные ячейку за ячейкой.

Формулы легко программировать, если вы знакомы с формулами Excel. Все, что вы вводите в ячейку, – это то, что вы генерируете в виде строки символов. Например, чтобы суммировать ячейки с C6 по C11, установите значение ячейки «=SUM(C6:C11)». Как и при интерактивном использовании Excel, если вы забудете знак равенства, формула превратится в строку символов.



Пример, показанный в листинге 1, демонстрирует создание электронной таблицы путем ввода данных ячейка за ячейкой. Он использует представление истории заказов TasTrade и добавляет данные для заказов одного клиента. Этот пример доступен как XLAddData.PRG в файлах загрузки для разработчиков, доступных по адресу www.hentzenwerke.com.

Листинг 1. Добавление данных в электронную таблицу. В этом примере используется представление «История заказов» TasTrade для создания простой электронной таблицы для заказов одного клиента.


```
* XLAddData.prg
* © 2000, Tamar E. Granor and Della Martin
* From: Microsoft Office Automation with Visual FoxPro
* Hentzenwerke Publishing. www.hentzenwerke.com

* Поместите информацию о заказе для одного клиента на лист Excel.

* Удалите все существующие ссылки на серверы.
* Это предотвращает потерю памяти оставшимися экземплярами.
RELEASE ALL LIKE o*

* В демонстрационных целях сделайте oExcel
* доступным после выполнения этой программы.
PUBLIC oExcel

LOCAL oBook, oRange

* Откройте представление «История заказов», которое
* содержит сводку заказов для одного клиента.
OPEN DATABASE _SAMPLES + "\TASTRADE\DATA\Tastrade"
USE CUSTOMER IN 0
SELECT 0
USE "Order History" ALIAS OrderHistory

* Добавьте рабочую книгу, используя настройки по умолчанию.
oExcel = CREATEOBJECT("Excel.Application")
oExcel.Visible = .T.
oBook = oExcel.Workbooks.Add()

WITH oExcel.ActiveSheet
    * Поместите имя клиента вверху.
    .Range("B2").Value = Customer.Company_Name

    * Поместите заголовки столбцов в строку 5.
    .Range("A5").Value = "Order Number"
    .Range("B5").Value = "Date"
    .Range("C5").Value = "Amount"

    oRange = .Range("A6:C6")
ENDWITH

* Просматривайте заказы и отправляйте данные.
SCAN
    WITH oRange
        .Columns[1].Value = Order_Id
        .Columns[2].Value = Order_Date
        .Columns[3].Value = Ord_Total
    ENDWITH

    * Переместить диапазон вниз на одну строку.
    oRange = oRange.Offset(1,0)
ENDSCAN
```

```

* Теперь добавьте итоговую строку.
nLastRow = oRange.Row && Свойство Row всегда дает первую строку
&& диапазона.
&& В этом диапазоне только одна строка.
nTotalRow = nLastRow + 2

WITH oExcel.ActiveSheet
    .Cells( nTotalRow, 1 ) = "Total"

    * Необходимо преобразовать nLastRow в char для использования в
    * формуле суммы
    oExcel.ActiveSheet.Cells( nTotalRow, 3 ).Formula = ;
        "=SUM( C6:C" + LTRIM(STR(nLastRow)) + " )"
ENDWITH

USE IN OrderHistory
USE IN Customer

```

Рисунок 3 показывает результаты. Таблица успешно заполнена. Однако требуется некоторое форматирование.

	A	B	C
1			
2		Alfreds Futterkiste	
3			
4			
5	Order Num	Date	Amount
6	953	2/7/95	\$507.06
7	836	12/9/94	\$835.43
8	703	9/6/94	\$337.44
9	693	8/27/94	\$851.22
10	644	7/19/94	#####
11	63	8/21/92	\$857.58
12			
13			
14	Total		#####

Рисунок 3. Результаты выполнения кода показаны в листинге 1. Символьные заголовки, данные из таблиц и формулы успешно импортированы. Небольшое форматирование, обсуждаемое в последующих разделах, решает проблемы с отображением.

Получение сгенерированного XLS

Этот метод заполнения данных ячейка за ячейкой хорошо работает с небольшими наборами данных. Если данные становятся длинными (скажем, более 300 записей), лучше использовать EXPORT или COPY TO TYPE XL5 и извлекать данные. Код для этого прост:

```

m.DataFile = "C:\My Documents\MyWorksheet"
SELECT DataTable
COPY TO (m.DataFile) TYPE XL5

* Откройте лист данных, созданный командой COPY TO...
oDataBook = oExcel.Workbooks.Open(m.DataFile + ".XLS")

```

Перемещение листов

Когда вы используете команду COPY TO, вы создаете книгу с одним листом. Если вам необходимо объединить несколько таких файлов XLS в книгу, состоящую из

нескольких листов, воспользуйтесь следующим приемом: используйте метод `Move` объекта `Worksheet`, чтобы переместить лист между книгами. В следующем примере предполагается, что вы хотите переместить первый (единственный) рабочий лист из объекта `oDataBook` (того, который создан из файла XLS) в объект `oOtherWorkBook`:

```
oDataBook.Worksheets(1).Move(oOtherWorkBook.ActiveSheet)
RELEASE oDataBook
```

Этот небольшой трюк перемещает единственный рабочий лист из `oDataBook` в `oOtherWorkBook`. Поскольку в результате в `oDataBook` не останется рабочих листов, Excel автоматически закроет книгу. Все, что вам нужно сделать, это удалить файл XLS и освободить переменную `oDataBook`, чтобы гарантировать освобождение ссылки на объект и корректное закрытие объекта `Application`.

Наполни ее

Бывают случаи, когда вам нужно взять число или формулу и повторить эту формулу определенное количество раз. Например, вы можете скопировать формулу из первой ячейки во все ячейки столбца. Методы `FillDown`, `FillUp`, `FillLeft` и `FillRight` объекта `Range` автоматически копируют содержимое ячейки. Например, метод `FillDown` берет ячейку вверху диапазона и копирует ее содержимое вниз. Следующий код устанавливает для первой ячейки значение 1, а затем использует метод `FillDown` для копирования верхней ячейки диапазона в каждую ячейку диапазона:

```
oExcel.ActiveSheet.Range("A1").Value = 1
oExcel.ActiveSheet.Range("A1:A10").FillDown
```

После запуска этого кода первые 10 строк в столбце А заполняются цифрой 1. «Не так впечатляюще, просто копирование числа», – скажете вы? Что ж, представьте, если бы эта ячейка представляла собой сложную формулу, и вам пришлось бы воспроизвести эту формулу в столбце. Команда `FillDown` начинает выглядеть довольно хорошо.

В Excel есть функция, позволяющая заполнять диапазоны автоматически сгенерированными данными. Например, вы можете пронумеровать список (скажем, от 1 до 50) или создать столбец дат, обозначающих пятницы в течение следующих 12 недель. Метод `DataSeries` объекта `Range` вычисляет значения ячеек в соответствии с параметрами серии.

Синтаксис следующий:

```
oExcel.ActiveSheet.Range("A1").DataSeries(nRowCol, nType, nDate,
                                           nStep, xStop, xTrend)
```

nRowCol	Numeric	Значение, указывающее, вводится ли ряд данных в строках или столбцах. Если этот параметр опущен, используется форма указанного диапазона. (Необязательный)			
		xlRows	1	xlColumns	2
nType	Numeric	Значение, указывающее тип ряда. (Необязательный)			
		xlDataSeriesLinear	-4132	xlGrowth	2
		xlChronological	3	xlAutoFill	4
nDate	Numeric	Используется только в том случае, если nType имеет значение xlChronological (3). Значение, указывающее тип хронологического ряда. (Необязательный)			

		xlDay (default)	1	xlWeekDay	2
		xlMonth	3	xlYear	4
nStep	Numeric	Значение, представляющее приращение между каждым значением в серии. По умолчанию – 1. (Необязательно)			
xStop	Numeric or date	Значение на котором следует остановиться. Тип может быть числовым или датой (или любым типом ряда, который вы создаете). По умолчанию используется конец диапазона. (Необязательный)			
XTrend	Logical or Numeric	True (или -1), чтобы установить линейную линию тренда (регрессионный анализ) для прогнозирования будущих тенденций, False (или 0), чтобы создать стандартный ряд. По умолчанию установлено значение «Ложь» (или 0). (Необязательный)			

Следующий код создает простую серию, нумеруя строки от 1 до 50. Обратите внимание: если вы не введете значение в первую ячейку серии, начинать серию будет не с чего, и команда будет выполняться без ошибок, но результирующий диапазон будет пустым.

```
#DEFINE xlColumns 2
#DEFINE xlDataSeriesLinear -4132
#DEFINE xlDay 1
oExcel.ActiveSheet.Range("A1").Value = 1
oExcel.ActiveSheet.Range("A1:A100").DataSeries(xlColumns, ;
    xlDataSeriesLinear, xlDay, 1, 50, .F.)
```

В следующем примере кода показано, как форматировать серию дат. В этом примере показано большее значение шага, вычисляя даты с интервалом в неделю (на самом деле это список пятниц между начальной датой, пятницей и концом года).

```
#DEFINE xlColumns 2
#DEFINE xlChronological 3
#DEFINE xlDay 1

oExcel.ActiveSheet.Range("B1").Value = "10/22/1999"
oExcel.ActiveSheet.Range("B1:B100").DataSeries(xlColumns, ;
    xlChronological, xlDay, 7, "12/31/99", .F.)
```



Excel интерпретирует строку «31.12.99» как данные типа даты. Вы также можете использовать дефисы. Excel выполняет некоторые преобразования дат, поскольку они хранятся в виде числового значения, называемого «последовательным значением». Это значение представляет собой количество дней, прошедших с 31 декабря 1899 года. Следовательно, значение 1 соответствует 1 января 1900 года.

Excel использует форматирование для отображения ячейки как значения даты, или вы можете выбрать числовой формат, чтобы увидеть серийное значение. Кроме того, значения времени сохраняются в виде долей суток; значение 1,5 соответствует 12 часам дня 01.01.1900. Опять же, вы будете использовать функции форматирования Excel для отображения даты, времени или того и другого. Подробную информацию о форматировании даты и времени см. в разделе

«Форматирование значений» далее в этой главе.

Обратите внимание, что этот метод вычисляет даты и вводит их как значение, а не как формулу. Если вы измените одну дату, не ждите, что изменятся и другие.

7.6 Форматирование

Перенести данные FoxPro в Excel – это очень здорово. Но еще круче сделать так, чтобы это выглядело хорошо. В Excel имеется широкий набор параметров форматирования ячейки, включая изменение атрибутов шрифта, форматирование внешнего вида значения (например, в процентах или отрицательных числах), а также настройку границ и заливки.

Ширина и выравнивание столбцов

Давайте вернемся к электронной таблице, которую мы создали на рисунке 3, используя XLAddData.PRG из файлов для загрузки разработчика, доступных по адресу: www.hentzenwerke.com. Проблема форматирования, которая в первую очередь бросается в глаза, – это ряд ячеек, заполненных знаками решетки, указывающими на то, что ширина столбца слишком мала.

Один действительно классный метод – AutoFit, который работает со столбцами в заданном диапазоне. На рисунке 3 мы можем автоматически подогнать ширину первых трех столбцов. Однако если мы запрашиваем весь диапазон, заголовок в ячейке B2 «Альфредс Futterkiste» приводит к тому, что столбец B становится намного шире, чем необходимо, что затрудняет чтение каждой строки. Хотя метод AutoFit влияет на все столбцы в диапазоне, он вычисляет максимальную ширину только для тех ячеек, которые фактически содержатся в вызывающем диапазоне. Указав A5:C14, он вычисляет максимальную ширину для столбцов A, B и C, но использует только ячейки в строках с 5 по 14. Метод AutoFit не принимает никаких параметров. Вот пример использования автоподбора:

```
oSheet.Range("A5:C14").Columns.AutoFit()
```

Как показано на рисунке 4, ситуация выглядит немного лучше. Хотя AutoFit генерирует ширину столбца, соответствующую самой широкой ячейке, ее едва хватает для самой широкой ячейки. Возможно, вы захотите явно указать один или два столбца.

	A	B	C
1			
2		Alfreds Futterkiste	
3			
4			
5	Order Number	Date	Amount
6	953	2/7/95	\$507.06
7	836	12/9/94	\$835.43
8	703	9/6/94	\$337.44
9	693	8/27/94	\$851.22
10	644	7/19/94	\$1,006.86
11	63	8/21/92	\$857.58
12			
13			
14	Total		\$4,395.59
15			

Рисунок 4. AutoFit творит чудеса со столбцами. Но впереди еще много форматирования!

Задайте ширину столбца вручную с помощью свойства ColumnWidth коллекции Columns. В свойстве ColumnWidth используются единицы измерения в символах, где один символ равен средней ширине всех символов шрифта по умолчанию. Свойству

можно задать дробные символы; фактически ширина столбца по умолчанию составляет 8,43 символа. Следующий код показывает, как добавить два символа к ширине столбца B.

```
WITH oExcel.ActiveSheet.Columns[2]
    .ColumnWidth = .ColumnWidth + 2
ENDWITH
```

Выравнивание

По умолчанию символьные данные выравниваются по левому краю, а числовые данные и данные даты/времени – по правому краю, и все они выравниваются по нижней части ячейки. Свойства HorizontalAlignment и UpperAlignment объекта Range хранят значения параметров выравнивания. Как и в большинстве свойств Office, здесь имеется ряд констант. На рисунке 5 показаны имена констант, значения и пример ячейки, отформатированной с учетом выравнивания. И HorizontalAlignment, и UpperAlignment действуют на одну и ту же ячейку, что предоставляет большое количество возможностей по выравниванию текста в ячейках.

	A	B	C	D	E	F	G
1	HORIZONTAL ALIGNMENT CONSTANTS:						
2	xlHAlignCenter	-4108		Test			
3	xlHAlignCenterAcrossSelection	7			Test (D-F)		
4	xlHAlignFill	5		TestTestTest			
5	xlHAlignGeneral	1		Test			
6	xlHAlignJustify	-4130		Test. This is a test. It is only a test.			
7	xlHAlignLeft	-4131		Test			
8	xlHAlignRight	-4152		Test			
9							
10	VERTICAL ALIGNMENT CONSTANTS:						
11	xlVAlignBottom	-4107		Test			
12	xlVAlignCenter	-4108		Test			
13	xlVAlignDistributed	-4117		Test			
14	xlVAlignJustify	-4130		This is a test. It is only a test.			
15	xlVAlignTop	-4160		Test			
16							

Justify works only for word-wrapped cells. It has no effect on a single line.

Рисунок 5. Константы выравнивания. В столбце A показано имя константы, в столбце B показано значение, а в столбце D показана форматированная ячейка, содержащая слово «Тест» (за исключением констант Justify, для которых требуется ячейка с переносом на слова, чтобы показать эффект). Обратите внимание, что выравнивание по центру выделения центрируется по трем столбцам.

Вы увидите, что xlHAlignGeneral и xlHAlignLeft выглядят одинаково; это потому, что ячейка содержит текст. Общий формат – это формат по умолчанию, в котором текст выравнивается по левому краю, а числа и даты – по правому краю. xlHAlignLeft и xlHAlignRight принудительно выравнивают по левому или правому краю, независимо от типа данных, содержащихся в ячейке.

Снова взглянув на пример TasTrade, мы можем центрировать имя клиента по первым трем столбцам, а затем отформатировать заголовки столбцов, используя следующий код:

```
#DEFINE xlHAlignCenterAcrossSelection 7
#DEFINE xlHAlignCenter -4108
```

* Центрируйте имя клиента по первым трем столбцам.

```
oSheet.Range("A2:C2").HorizontalAlignment = ;
xlHAlignCenterAcrossSelection
```

* Центрируйте каждый заголовок столбца.

```
oSheet.Range("A5:C5").HorizontalAlignment = xlHAlignCenter
```

Если вы хотите использовать выравнивание по ширине, текст должен переноситься внутри ячейки (в противном случае он просто выравнивает текст по левому краю). Используйте свойство WrapText, чтобы настроить ячейку для переноса слов:

```
oSheet.Range("A5").WrapText = .T.
```

Шрифты

Объект Font Excel очень похож на объект Word со свойствами Name, Size, Bold, Italic и т. д. В Excel шрифты управляются на вкладке «Шрифт» диалогового окна «Формат ячеек» (см. рис. 6). Если мы сравним это диалоговое окно с диалоговым окном Word, мы обнаружим, что Excel имеет подмножество свойств шрифта, которые есть в Word. К счастью, большинство свойств объекта Font Excel находятся в Word и имеют схожие названия.

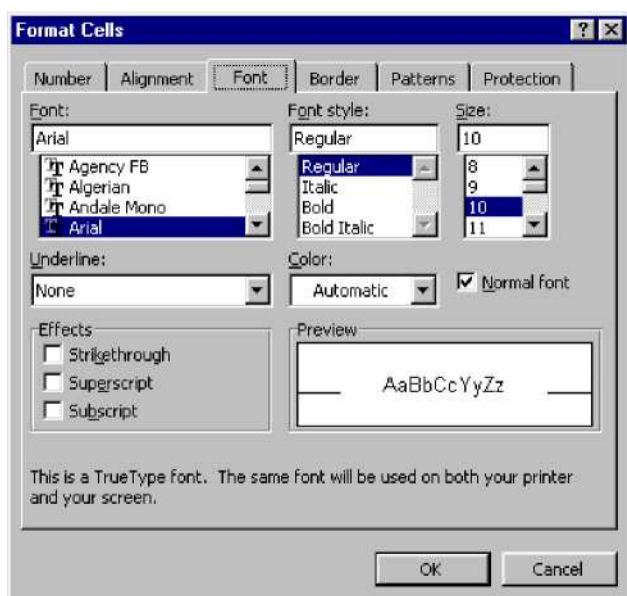


Рисунок 6. Вкладка «Шрифт» Excel в диалоговом окне «Формат ячеек». Excel предоставляет широкий спектр функций форматирования шрифтов.

Чтобы получить доступ к объекту Font, используйте свойство Font объекта Range (многие другие объекты, например различные текстовые объекты на диаграмме, также используют свойство Font). Чтобы установить для диапазона (в этом примере это ячейка B2) жирный шрифт Times New Roman размером 14 пунктов, используйте следующее:

```
WITH oSheet.Range("B2").Font
    .Name = "Times New Roman"
    .Size = 14
    .Bold = .T.
```

ENDWITH

Таблица 1 показывает свойства шрифта, наиболее часто используемые в Excel. Доступно еще несколько, хотя и не так много, как в объекте Font Word. Дополнительные свойства см. в файле справки в разделе «Объект Font».

Таблица 1. Свойства шрифта. Объект Font управляет внешним видом шрифта: от его начертания до его размера, стиля и многого другого. В этой таблице показаны наиболее распространенные свойства. Проверьте справку для получения дополнительных настроек.

Свойство	Тип	Описание
Name	Character	Название шрифта.
Size	Numeric	Размер шрифта в пунктах.
Color	RGB Color	Цвет шрифта.
Bold	Numeric or Logical	Указывает, выделен ли текст жирным шрифтом.
Italic	Numeric or Logical	Указывает, выделен ли текст курсивом.
Underline	Numeric	Тип подчеркивания. xlUnderlineStyleDouble -4119 xlUnderlineStyleDoubleAccounting 5 xlUnderlineStyleNone -4142 xlUnderlineStyleSingle 2 xlUnderlineStyleSingleAccounting 4
Superscript, Subscript	Numeric or Logical	Указывает, является ли текст надстрочным или подстрочным индексом соответственно.



В Excel есть некоторые свойства, которые мы перечисляем как «числовые или логические». Эти свойства могут быть установлены с помощью логических значений VFP .T. и .F., или -1 (истина) и 0 (ложь). Однако при запросе этих свойств Excel возвращает числовое значение. Итак, хотя вы можете включить курсив с помощью `oRange.Font.Italic = .T.` или `oRange.Font.Italic = -1`, вы должны помнить, что возвращается числовое значение, как в случае `IF oRange.Font.Italic = -1 ... ENDIF`.

Форматирование значений

Excel предлагает функцию форматирования чисел, которая действует как свойство InputMask в FoxPro на стероидах. Его можно найти в диалоговом окне «Формат ячеек», изображенном на рисунке 7. Всего имеется 12 категорий, каждая из которых имеет до 31 предопределенного кода. Кроме того, если вы не можете найти тот, который соответствует вашим потребностям, вы можете создать свой собственный.

Вероятно, вы ожидаете увидеть длинный список свойств с длинным списком констант, представляющих каждый числовой формат. Нет, не в этот раз. Существует только одно свойство – свойство NumberFormat Range. Ему присваивается строка, представляющая формат, во многом аналогично тому, как работает свойство InputMask в VFP. Вы можете увидеть несколько примеров строк, если выберете категорию «Произвольная» на вкладке «Число» в диалоговом окне «Формат ячеек» (см. рис. 7).

Наиболее часто используемые коды форматных строк показаны в Таблице 2. Как и

в кодах InputMask VFP, знак решетки (#) отображает число, а запятая ставит разделитель тысяч. Многие символы, такие как %, \$ и (символ евро), являются литералами; вам все равно нужно вставить заполнители для отображения чисел. Коды, относящиеся к одному и тому же типу данных, можно смешивать и сопоставлять, чтобы точно задать форматирование ячейки.

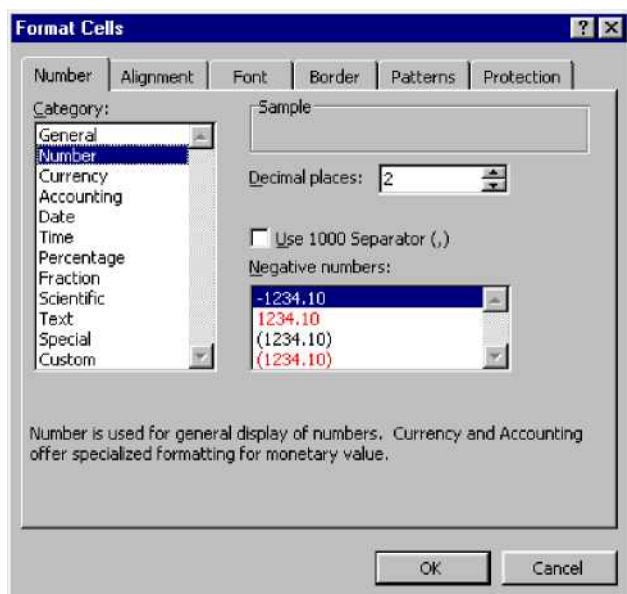


Рисунок 7. Вкладка «Число» диалогового окна «Формат ячеек» в Excel. Как и свойство InputMask в FoxPro на стероидах, Excel предлагает, казалось бы, неограниченные способы форматирования данных.

Таблица 2. Часто используемые коды для форматирования чисел. Есть еще много доступных; см. раздел справки «О числовых форматах» в обычном файле справки Excel.

Код	Описание	Пример значения	Пример строки	Пример вывода
"General"	Возврат к формату по умолчанию.	1234.5	"General"	1234.5
#	Отображает число (пусто, если в начале или в конце стоит 0).	1234.5	"#####.##"	1234.5
0	Отображает число, включая ведущие и конечные 0.	1234.5	"00000.00"	01234.50
,	Добавляет разделитель тысяч.	1234.5	"###,###.##"	1,234.5
%	Отображает числа в процентах.	.08	"###%"	8%
\$	Вставляет знак доллара.	1.25	"\$##.00"	\$1.25
€	Вставляет символ евро.	3.00	CHR(128) + "##.00"	€3.00
M	Отображает месяц в виде числа от 1 до 12.	10/22/99	"M"	10
Mmm	Отображает месяц в виде трехсимвольной аббревиатуры.	10/22/99	"Mmm"	Oct
D	Отображает день в виде числа от 1 до 31.	10/22/99	"D"	22

Ddd	Отображает день в виде трехсимвольного дня недели.	10/22/99	"Ddd"	Fri
Yy	Отображает двузначный год.	10/22/99	"Yy"	99

Давайте посмотрим на несколько числовых примеров. В этих примерах предполагается, что у вас есть пустой лист, на который ссылается oSheet. Результаты всех примеров числового кода показаны на рисунке 8 в конце числовых примеров. Для нашего первого примера давайте отформатируем 1234,5 так, чтобы оно отображалось как 1234,50:

```
oSheet.Range("A1").Value = "Format a Cell"
oSheet.Columns[1].ColumnWidth = 15
oSheet.Range("A2").Value = 1234.5
oSheet.Range("A2").NumberFormat = "##,###.##"
```

Если это должна быть валюта, с разделителем тысяч, например \$1234,50, попробуйте следующий NumberFormat:

```
oSheet.Range("A3").Value = 1234.5
oSheet.Range("A3").NumberFormat = "$##,###.##"
```

Для тех из вас, кто использует разные валюты, используйте код ASCII, представляющий символ вашей валюты: € – CHR(128), £ – CHR(163) и ¥ – CHR(162).

Excel допускает четыре раздела кодов, что-то вроде встроенного оператора регистра: положительные числа, отрицательные числа, нулевые значения и текст. Укажите строку для каждого раздела, разделенную точкой с запятой. Вот пример того, как форматировать отрицательные числа с помощью круглых скобок вместо знака минус, используя столбец B:

```
oSheet.Range("B1").Value = "Format +/-"
oSheet.Columns[2].ColumnWidth = 15
oSheet.Range("B2").Value = 1234.5
oSheet.Range("B3").Value = 0
oSheet.Range("B4").Value = -1234.5
oSheet.Range("B2:B4").NumberFormat = "##,###.##; (##,###.##)"
```

В ячейке B2 отображается значение 1234,50, а в ячейке B4 – (1234,50). Одна из проблем, с которыми мы часто сталкивались, заключается в том, как обрабатывать нули. В этом случае ноль отображается как «.0», но во многих случаях мы хотим, чтобы ячейка оставалась пустой. Используя ту же последовательность чисел, на этот раз в столбце C, попробуйте следующее:

```
oSheet.Range("C1").Value = "Format +/-/0"
oSheet.Columns[3].ColumnWidth = 15
oSheet.Range("C2").Value = 1234.5
oSheet.Range("C3").Value = 0
oSheet.Range("C4").Value = -1234.5
oSheet.Range("C2:C4").NumberFormat = "##,###.##; (##,###.##); ;"
```

Если вы посмотрите на рисунок 8, вы увидите результаты, если оставить третий раздел строки NumberFormat пустым: для нулей ничего не отображается. Что делать, если вы хотите, чтобы в ячейке с нулевым значением отображалось «N/A»? Совсем не сложно. Измените строку формата на это:

```
[##,###.##; (##,###.##); "N/A"; ]
```

Внимание: Excel нужны литеральные строки, заключенные только в двойные

кавычки, поэтому не забудьте ограничить всю строку FoxPro квадратными скобками или одинарными кавычками. Если вы забудете, вы получите сообщение об ошибке, указывающее, что Excel «не может установить свойство NumberFormat».

Четвертый раздел NumberFormat позволяет форматировать текстовые ячейки. Текстовые форматы немного другие; все значение текстовой строки ячейки обозначается знаком @, и вы можете поместить буквальный текст (только через двойные кавычки) по обе стороны от знака @. Почему вы хотите добавить что-то в текст? Возможно, ваши данные содержат только имя клиента, и там должно быть написано «Итоги клиентов». Вот пример, включающий форматирование текста (результаты см. на рисунке 8):

* Введите некоторые данные

```
oSheet.Range("A6").Value = "Январь"
oSheet.Range("A7").Value = 102
oSheet.Range("B7").Value = "Widget"
oSheet.Range("A8").Value = -24
oSheet.Range("B8").Value = "Whatzit"
oSheet.Range("A9").Value = 0
oSheet.Range("B9").Value = "Whatnot"
oSheet.Range("A10").Value = "Февраль"
oSheet.Range("A11").Value = 123
oSheet.Range("B11").Value = "Widget"
oSheet.Range("A12").Value = 0
oSheet.Range("B12").Value = "Whatzit"
oSheet.Range("A13").Value = 332
oSheet.Range("B13").Value = "Whatnot"
```

* Отформатируйте первый столбец, используя все четыре раздела.

```
oSheet.Range("A6:A13").NumberFormat=["$###; ($###); Нет"; @ "Итоги"]
```

* Отформатируйте второй столбец только для текста. Строки-литералы размещаются по обе стороны от фактического значения ячейки.

```
oSheet.Range("B6:B13").NumberFormat = ["Общий объем продаж "@""]
```

	A	B	C
1	Format a Cell	Format +/-	Format +/-0
2	1,234.50	1,234.50	1,234.50
3	\$1,234.50	.0	
4		(1,234.50)	(1,234.50)
5			
6	January Totals		
7	\$102	Total Widget Sales	
8	(\$24)	Total Whatzit Sales	
9	No	Total Whatnot Sales	
10	February Totals		
11	\$123	Total Widget Sales	
12	No	Total Whatzit Sales	
13	\$332	Total Whatnot Sales	
14			
15			

Рисунок 8. Объединенные результаты примеров числового форматирования. Ячейка A2 показывает базовое форматирование; A3 показывает валюту; B2:B4 по-разному форматирует положительные и отрицательные числа; и C2:C4 различают положительные, нулевые и отрицательные значения. A6:B13 показывает использование всех четырех разделов форматирования в столбце A, а текст столбца B также форматируется.

Даты также имеют несколько форматов. Используя «D» для обозначения дня, «M» для обозначения месяца и «Y» для обозначения года вместе с любыми

литералами, вы можете создать практически любой формат. При использовании D и M день и месяц отображаются в виде числа без ведущего нуля. Используйте Dd или Mm, чтобы указать начальный ноль. Три символа, например «Ддд» или «Ммм», образуют трехсимвольное сокращение, например «Пят» или «Янв». Четыре символа, например «Дддд» или «Мммм», отображают полное имя, например «Пятница» или «Январь». Строка Yyyy дает четырехзначный год, соответствующий требованиям Y2K. Если вам нужна дата в военном формате, попробуйте следующее:

```
oSheet.Range("A1").Value = "01/01/2000"
oSheet.Range("A1").NumberFormat = "Dd-Mmm-Yyyy"
```

В ячейке отображается «01 января 2000 г.». Что, если вы хотите, чтобы там было написано «Суббота, 1 января 2000 г.»? Измените строку формата на «Dddd, D Mmmm, Yyy», чтобы отобразить желаемый формат.

Существует множество комбинаций и перестановок форматирования ячеек. Вместо того, чтобы утомлять вас на следующих нескольких страницах попытками объяснить каждую деталь, мы просто укажем вам на файл справки Excel (обычную справку Excel, а не справку VBA), в которой это подробно описано. Найдите тему «О числовых форматах», в которой объясняется все, что вы когда-либо хотели знать о строках NumberFormat.

Границы

Коллекция Borders содержит объекты Border, каждый из которых представляет восемь границ ячейки. Эта коллекция не была бы полной без серии констант, описывающих восемь различных границ: xlDiagonalDown (5), xlDiagonalUp (6), xlEdgeLeft (7), xlEdgeTop (8), xlEdgeBottom (9), xlEdgeRight (10), xlInsideVertical (11) и xlInsideHorizontal (12). Границы xlEdge представляют собой внешний периметр диапазона ячеек. Границы xlInside – это внутренние границы диапазона, состоящего из нескольких ячеек. Если диапазон представляет собой одну ячейку, границы xlInside игнорируются.

Таблица 3 показывает свойства для каждой границы. Свойства LineStyle и Weight задаются постоянными значениями.

Таблица 3. Свойства границы. Границы можно размещать вокруг любого диапазона, а их внешний вид можно изменять с помощью этих свойств.

Свойство	Тип	Описание
Color	RGB Color	Цвет границы.
LineStyle	Numeric	Числовое значение, соответствующее предустановленному стилю линии. xlContinuous 1 xlDot -4118 xlDash -4115 xlDouble -4119 xlDashDot 4 xlLineStyleNone -4142 xlDashDotDot 5 xlSlantDashDot 13
Weight	Numeric	Ширина линии. Здесь используются константы, а не пункты. xlHairline 1 xlMedium -4138 xlThin 2 xlThick 4

Если вы хотите разместить пунктирную границу внизу диапазона ячеек, используйте этот код:

```
#DEFINE xlEdgeBottom      9
#DEFINE xlDot              -4118
```

```
oSheet.Range("A2:C2").Borders(xlEdgeBottom).LineStyle = xlDot
```

В ходе этого процесса мы обнаружили несколько ситуаций. По большей части вы можете установить любой стиль линии, какой захотите. Когда вы устанавливаете ширину, результаты становятся непредсказуемыми, обычно это приводит к появлению непрерывной линии, а иногда и к случайному цвету. Например, следующий код создает толстую сплошную линию, а не толстую пунктирную линию:

```
#DEFINE xlEdgeBottom      9
#DEFINE xlDashDot          4
#DEFINE xlThick            4
```

```
oSheet.Range("A2:C2").Borders(xlEdgeBottom).LineStyle = xlDashDot
oSheet.Range("A2:C2").Borders(xlEdgeBottom).Weight = xlThick
```

Кажется, работает любой Weight с непрерывной линией или любой LineStyle без установки свойства Weight. В VFP установка для свойства BorderWidth значения больше 1 подразумевает одностороннюю границу. То, что мы обнаружили в границах Excel, может соответствовать дизайну VFP. Мы не нашли никакой документации по дизайну Excel.

Затенение

Затенение диапазона ячеек осуществляется с помощью объекта Interior. Обратите внимание, что в Word используется объект Shading, а в PowerPoint – объект Fill.

Таблица 4 показывает свойства объекта Interior. По сути, вы выбираете узор, затем применяете к нему PatternColor, и свойство Color становится цветом фона. Однако предсказать результат сложно. Тот факт, что свойство Color является цветом фона, не означает, что это менее преобладающий цвет. Рисунок 9 иллюстрирует эту ситуацию. Белый (фактически автоматический цвет) – это настройка Color, а настройка PatternColor – черный. Глядя на образцы имеющихся узоров, можно увидеть, что некоторые из них преимущественно белые, а некоторые преимущественно черные.

Очевидно, что Excel дает нам широкие возможности управления формированием рисунка ячейки. Здесь следует соблюдать осторожность при проектировании: если вы попытаетесь оформить ячейку чем-либо, кроме сплошного цвета, любой текст размером менее 36 пунктов, который появляется в этом шаблоне, будет трудно прочитать. Мы не обнаружили особой бизнес-потребности в каких-либо шаблонах, кроме сплошных.

На самом деле, чтобы закрасить ячейку, вам не нужно ничего делать, кроме изменения свойства Interior.Color. Поскольку он действует как цвет фона и нет шаблона для отображения, ячейка затеняется, и для этого требуется всего одна строка кода (вместо того, чтобы задавать для Pattern значение xlPatternSolid, а затем задавать PatternColor). Затенение осуществляется следующим образом:

&& Светло-серый

```
oSheet.Range("A14:C14").Interior.Color = RGB(192,192,192)
```

Таблица 4. Внутренняя отделка. Объект Interior имеет несколько свойств для изменения отображения ячейки.

Свойство	Тип	Описание																																								
Color	RGB Color	Цвет фона, просвечивающий сквозь узор. Если узор представляет собой мелкий узор (например, узоры xlPatternLight), этот цвет становится преобладающим цветом, поскольку сам узор не доминирует в ячейке.																																								
InvertIfNegative	Logical or Numeric	Инвертируется ли шаблон, если значение заштрихованной ячейки отрицательное. True (.T. или 1) для инвертирования, если оно отрицательное, False (.F. или 0), чтобы оставаться неизменным, если оно отрицательное.																																								
Pattern	Numeric	Создает шаблоны диапазона с помощью одного из 20 предопределенных шаблонов. <table><tr><td>xlPatternAutomatic</td><td>-4105</td><td>xlPatternHorizontal</td><td>-4128</td></tr><tr><td>xlPatternChecker</td><td>9</td><td>xlPatternLightDown</td><td>13</td></tr><tr><td>xlPatternCrissCross</td><td>16</td><td>xlPatternLightHorizontal</td><td>11</td></tr><tr><td>xlPatternDown</td><td>-4121</td><td>xlPatternLightUp</td><td>14</td></tr><tr><td>xlPatternGray8</td><td>18</td><td>xlPatternLightVertical</td><td>12</td></tr><tr><td>xlPatternGray16</td><td>17</td><td>xlPatternNone</td><td>-4142</td></tr><tr><td>xlPatternGray25</td><td>-4124</td><td>xlPatternSemiGray75</td><td>10</td></tr><tr><td>xlPatternGray50</td><td>-4125</td><td>xlPatternSolid</td><td>1</td></tr><tr><td>xlPatternGray75</td><td>-4126</td><td>xlPatternUp</td><td>-4162</td></tr><tr><td>xlPatternGrid</td><td>15</td><td>xlPatternVertical</td><td>-4166</td></tr></table>	xlPatternAutomatic	-4105	xlPatternHorizontal	-4128	xlPatternChecker	9	xlPatternLightDown	13	xlPatternCrissCross	16	xlPatternLightHorizontal	11	xlPatternDown	-4121	xlPatternLightUp	14	xlPatternGray8	18	xlPatternLightVertical	12	xlPatternGray16	17	xlPatternNone	-4142	xlPatternGray25	-4124	xlPatternSemiGray75	10	xlPatternGray50	-4125	xlPatternSolid	1	xlPatternGray75	-4126	xlPatternUp	-4162	xlPatternGrid	15	xlPatternVertical	-4166
xlPatternAutomatic	-4105	xlPatternHorizontal	-4128																																							
xlPatternChecker	9	xlPatternLightDown	13																																							
xlPatternCrissCross	16	xlPatternLightHorizontal	11																																							
xlPatternDown	-4121	xlPatternLightUp	14																																							
xlPatternGray8	18	xlPatternLightVertical	12																																							
xlPatternGray16	17	xlPatternNone	-4142																																							
xlPatternGray25	-4124	xlPatternSemiGray75	10																																							
xlPatternGray50	-4125	xlPatternSolid	1																																							
xlPatternGray75	-4126	xlPatternUp	-4162																																							
xlPatternGrid	15	xlPatternVertical	-4166																																							
PatternColor	RGB Color	Цвет переднего плана узора. Свойство Color определяет цвет фона.																																								

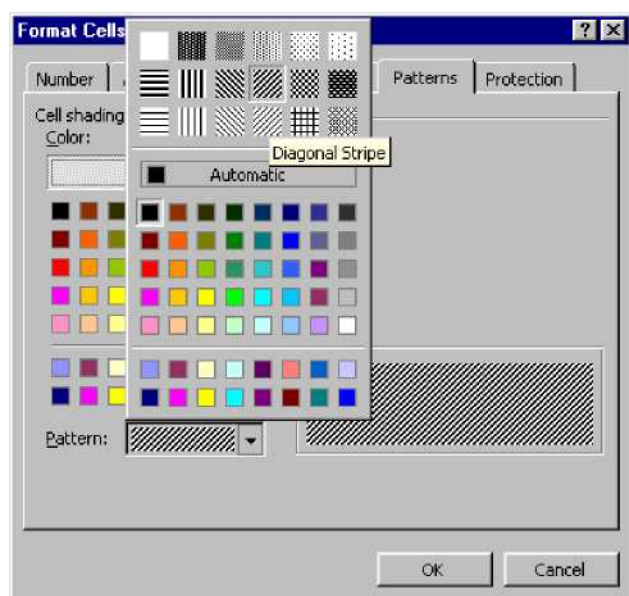


Рисунок 9. Диалоговое окно «Формат ячеек» с выбранным узором «Диагональная

полоса». Обратите внимание, что PatternColor – черный, а Color – Automatic или белый (он скрыт окном шаблона). Глядя на все образцы узоров, легко увидеть эффект свойств Color и PatternColor.

Стили

Как и Word, Excel имеет возможность работать со стилями. Стили весьма полезны для стандартизации внешнего вида вашей электронной таблицы. Назначив все свойства форматирования, включая шрифты, выравнивание, числовое форматирование, цвет, границы и затенение, одному стилю, вам не придется тратить время на вырезание и вставку кода снова и снова (в надежде, что вы вырезали и вставили правильный код форматирования).

Объект Workbook имеет свойство Styles, которое возвращает объект коллекции Styles. Коллекция стилей начинается с нескольких стилей по умолчанию: Денежный, Денежный [0], Обычный, Процентный, Финансовый и Финансовый [0] (Normal, Comma, Comma[0], Currency, Currency[0] и Percent). В стилях с суффиксом «[0]» два десятичных знака после числа опускаются. Вы также можете определить свои собственные стили. Обычный стиль является стилем по умолчанию. Используйте это, чтобы вернуться к настройкам по умолчанию. Остальные используйте для быстрой установки наиболее распространенного числового форматирования.

Хотя стили по умолчанию полезны, они довольно просты. Наши клиенты требуют изящества, поэтому мы хотим чего-то более изощренного, чем просто запятые в наших цифрах. Поскольку Office является полиморфным, а объект Style принадлежит коллекции Styles, мы знаем, что нужно искать метод Add. После добавления стиля мы можем изменить его атрибуты. Все форматирование, обсуждаемое в этой главе, было проиллюстрировано с помощью объекта Range, но оно также работает и с объектом Style.

Следующий код создает стиль, подходящий для заголовка. В стиле используется Times New Roman размером 14 пунктов, он выделен жирным шрифтом, центрирован и затенен светло-серым цветом.

```
#DEFINE xlHAlignCenter -4108
#DEFINE xlVAlignCenter -4108
#DEFINE xlEdgeBottom 9
#DEFINE xlEdgeTop 10
#DEFINE xlEdgeRight 8
#DEFINE xlEdgeLeft 7
#DEFINE xlDouble -4119
#DEFINE xlMedium -4138

* Добавьте новый стиль
oSheet.Parent.Styles.Add("Title")

* Измените атрибуты стиля
WITH oSheet.Parent.Styles("Title")

    * Шрифт
    .Font.Name = "Times New Roman"
    .Font.Size = 14
    .Font.Bold = .T.

    * Выравнивание
    .HorizontalAlignment = xlHAlignCenter
```

```
.VerticalAlignment = xlVAlignCenter
```

```
* Затенение
```

```
.Interior.Color = RGB(220, 220, 220) && Светло-серый
```

```
ENDWITH
```

```
* Присвойте ячейке A1 значение, затем примените стиль.
```

```
oSheet.Range("A1").Value = "Testing"
```

```
oSheet.Range("A1").Style = "Title"
```

Единственный аспект, который радикально отличается между Styles и Ranges, — это объект Border. Меняется весь ряд констант, и это не описано в файле справки.

Единственный способ узнать об этом — запустить средство записи макросов и убедиться, что константы, которые он записывает для Styles, сильно отличаются от констант, используемых для Ranges. Для объекта Border необходимы следующие константы: xlLeft (-4131), xlTop (-4160), xlBottom (-4107), xlRight (-4152), xlDiagonalUp (6) и xlDiagonalDown (5). Обратите внимание, что внутренние границы отсутствуют; мы не уверены, куда они пошли.

Если вас это утешит, то в границах есть определенная последовательность. Та же ошибка, которая влияет на свойство Weight, возникает и со стилями.

7.7 Вывод данных

Если у вас есть красивая электронная таблица, ваши пользователи неизбежно захотят ее распечатать.

Настройка страницы

Excel имеет мощный набор функций, используемых для печати страницы, о чем свидетельствует **рисунок 10**, диалоговое окно «Параметры страницы». Диалоговое окно «Параметры страницы» Excel соответствует его объекту PageSetup. Каждый лист (и каждая диаграмма) имеет свой собственный объект PageSetup.

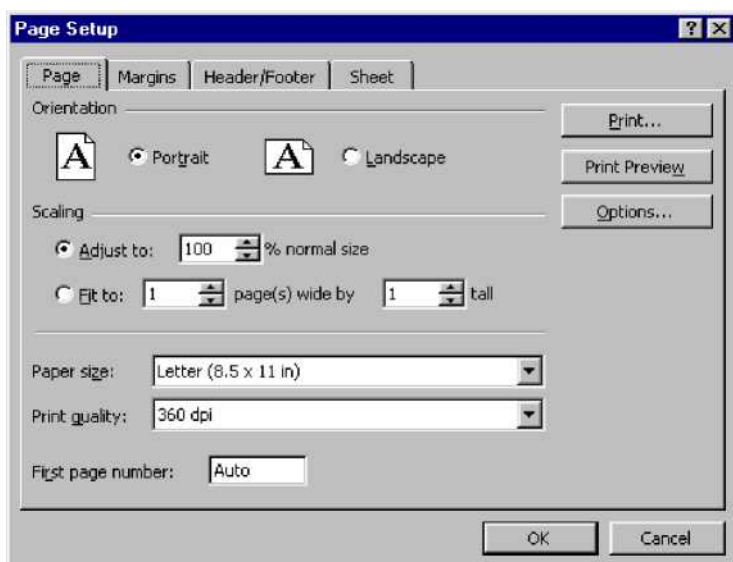


Рисунок 10. Настройка страниц. Это одна из четырех вкладок свойств, используемых при печати.

PageSetup имеет 35 свойств и один метод... это надежно! Многие из них полезны для повседневных задач. В **Таблице 5** показано большинство свойств — они сгруппированы по вкладкам, на которых они находятся в диалоговом окне

«Параметры страницы» (полезный порядок, если вы используете средство записи макросов).

Таблица 5. Свойства объекта PageSetup. Напоминание: свойства, перечисленные как логические или числовые, могут быть заданы как с логическими, так и с числовыми значениями, но их следует проверять только на числовых значениях.

Свойство	Тип	Описание
Orientation	Numeric	Ориентация печати на бумаге. xlPortrait 1 xlLandscape 2
Zoom	Logical or Numeric	Если числовое, применяется процентное значение (от 10 до 400) для масштабирования листа. Если установлено значение False (.F. или 0), FitToPagesTall и/или FitToPagesWide используются для масштабирования рабочего листа.
FitToPagesTall	Logical or Numeric	Масштабирует вертикальную область печати, чтобы она поместилась на указанное количество страниц. Если значение False (.F. или 0), вертикальное масштабирование не происходит. Игнорируется, если свойство Zoom имеет значение True.
FitToPagesWide	Logical or Numeric	Масштабирует горизонтальную область печати, чтобы она поместилась на указанное количество страниц. Если значение False (.F. или 0), горизонтальное масштабирование не происходит. Игнорируется, если свойство Zoom имеет значение True.
PaperSize	Numeric	Один из примерно 45 предустановленных размеров бумаги. Некоторые из наиболее распространенных: xlPaperLetter 1 xlPaperLegal 5 xlPaperA4 9 xlEnvelope9 19
FirstPageNumber	Numeric	Номер страницы, используемый для начала нумерации страниц. Значение по умолчанию – 1.
TopMargin	Numeric	Расстояние от верха страницы до верха первой строки текста в пунктах.
HeaderMargin	Numeric	Расстояние от верха страницы до верха первой строки текста в шапке, в пунктах.
LeftMargin	Numeric	Расстояние от левого края страницы до левого края текста в пунктах.
RightMargin	Numeric	Расстояние от правого края страницы до правого края текста в пунктах.
BottomMargin	Numeric	Расстояние от нижнего края страницы до нижней строки текста, в пунктах.
FooterMargin	Numeric	Расстояние от нижнего края страницы до нижней строки текста нижнего колонтитула, в пунктах.

CenterHorizontally	Logical or Numeric	True (.Т. или -1), чтобы центрировать текст на странице по горизонтали; Значение False (.F. или 0), чтобы начать печать с левого поля.
CenterVertically	Logical or Numeric	True (.Т. или -1) для центрирования текста на странице по вертикали; Значение False (.F. или 0), чтобы начать печать с верхнего поля.
LeftHeader	Character	Эти свойства задают левую, центральную или правую части верхнего или нижнего колонтитула, как указано. Им можно задать строку символов или коды, или их комбинацию. Полный набор кодов находится в файле справки Excel VBA в разделе «Коды форматирования верхних и нижних колонтитулов».
CenterHeader	Character	
RightHeader	Character	
LeftFooter	Character	
CenterFooter	Character	
RightFooter	Character	Печатает текущую дату &D Печатает имя файла &F Печатает имя активного листа &A Печатает номер текущей страницы &P Печатает символ амперсанда && Печатает общее количество страниц &N в документе.
PrintArea	Character	Диапазон ячеек для печати, например «A1:Z40». Если пусто, он напечатает их все.
PrintGridlines	Logical or Numeric	True (.Т. или -1) для печати линий сетки ячеек, False (.F. или 0) для их пропуска.
BlackAndWhite	Logical or Numeric	True (.Т. или -1) для печати цветов в высококонтрастных оттенках серого для наилучшей печати на черно-белом принтере; Ложь (.F. или 0), чтобы печатать в цвете или чтобы цвета печатались в любых оттенках серого, на которые запрограммирован принтер.
Draft	Logical or Numeric	True (.Т. или -1) для печати данных без графики; Ложь (.F. или 0) печатает все.
Order	Numeric	Порядок печати и нумерации страниц, если лист слишком велик и не помещается на одной странице. xlDownThenOver 1 xlOverThenDown 2

Эти свойства довольно просты. Вот пример кода, который гарантирует, что электронная таблица будет напечатана на одном листе с полями в полдюйма, в альбомной ориентации и с номерами страниц в верхнем левом заголовке.

```
#DEFINE xlLandscape 2
#DEFINE autoIn2Pts 72
```

```
oSheet.PageSetup.Orientation = xlLandscape
oSheet.PageSetup.FitToPagesTall = 1
```

```
oSheet.PageSetup.FitToPagesWide = 1
oSheet.PageSetup.TopMargin      = .5 * autoIn2Pts
oSheet.PageSetup.BottomMargin   = .5 * autoIn2Pts
oSheet.PageSetup.LeftMargin     = .5 * autoIn2Pts
oSheet.PageSetup.RightMargin    = .5 * autoIn2Pts
oSheet.PageSetup.LeftHeader     = "Page &P"
```

Предварительный просмотр печати

В некоторых наших приложениях Автоматизации окончательный документ отображается в режиме предварительного просмотра. Это позволяет пользователю анализировать данные, а затем решать, достаточно ли они важны для печати или данные требуют дополнительной проверки. Печать Excel основана на рабочих листах, а не на книгах. Вы вызовете метод PrintPreview из объекта Sheet. Это также означает, что вы увидите один лист в PrintPreview; нельзя прокручивать листы в книге (хотя пользователи Excel к этому вполне привыкли, это становится шоком для разработчиков VFP, которые могут прокручивать весь отчет, и пользователей Word, которые могут прокручивать весь документ).

Чтобы вызвать режим предварительного просмотра, используйте метод PrintPreview листа:

```
oSheet.PrintPreview()
```

Однако это переводит FoxPro в состояние ожидания; пользователю необходимо нажать либо на мигающий заголовок окна, либо на мигающую кнопку в строке меню. Как только пользователь это сделает, откроется окно PrintPreview. Но FoxPro не может продолжить работу, пока окно PrintPreview не будет закрыто. Перед вводом команды PrintPreview убедитесь, что окно объекта приложения Excel находится в нормальном состоянии, и даже в этом случае пользователь может не увидеть ни одного мигающего элемента и предположить, что приложение зависло.

Альтернативный синтаксис, использующий метод PrintOut и параметр PrintPreview, по-прежнему сталкивается с той же проблемой.

```
oSheet.PrintOut(,,,T.)
```

Существует способ отобразить экран PrintPreview, но для продолжения работы приложения FoxPro требуется, чтобы пользователь закрыл окно PrintPreview. Он использует некоторые вызовы API для управления окнами. Функция FindWindowA получает дескриптор окна приложения на основе его класса и имени окна. В Таблице 6 показаны классы приложений Office (действительны для версий 97 и 2000), а также различные версии FoxPro. Функция SetWindowPos устанавливает окно как самое верхнее (или нет) на основе дескриптора окна. Вы должны не забыть снова установить окно на NoTopMost, чтобы обеспечить возможность вывода других приложений вперед.

Таблица 6. Имена классов, используемые в FindWindowA. Передаваемый в API вместе с заголовком окна, он возвращает действительный дескриптор окна, который можно использовать для управления окном с помощью других API.

Приложение	Класс
Excel 97 and 2000	XLMAIN
Outlook 97 and 2000	rctrl_renwnd32
PowerPoint 97	PP97FrameClass
PowerPoint 2000	PP9FrameClass

Word 97 and 2000	OpusApp
VFP 3.0 and 5.0	Fox4000001
VFP 6.0	VFP66400000

```
#DEFINE swp_nosize      1
#DEFINE swp_nomove      2
#DEFINE hwnd_topmost    -1
#DEFINE hwnd_notopmost  -2

* FindWindowA возвращает дескриптор окна из заголовка окна
DECLARE LONG FindWindowA IN WIN32API STRING class, STRING title
* SetWindowPos перемещает окно вверх, используя дескриптор окна
DECLARE SetWindowPos IN WIN32API LONG HWND, LONG hwndafter, ;
    LONG x, LONG Y, LONG cx, LONG cy, LONG flags

* Захватите дескрипторы окон для Excel и VFP.
hWndXL = FindWindowA("XLMAIN", oExcel.Caption)
hWndVFP = FindWindowA("VFP66400000", _VFP.Caption)

* Переместите VFP наверх и попросите пользователя закрыть
* PrintPreview.
= SetWindowPos(hWndVFP, hwnd_topmost, 0,0,0,0, swp_nosize + ;
    swp_nomove)
= MessageBox("Закройте это сообщение, и Excel откроется." + ;
    CHR(13) + "Закройте предварительный просмотр печати, "+;
    " когда будете готовы вернуться в FoxPro.")

* Теперь поднимите окно Excel наверх.
= SetWindowPos(hWndVFP, hwnd_notopmost, 0,0,0,0, swp_nosize + ;
    swp_nomove)
= SetWindowPos(hWndXL, hwnd_topmost, 0,0,0,0, swp_nosize + ;
    swp_nomove)

* Выполните метод PrintPreview перед печатью.
* Убедитесь, что ваш код может терпеть состояние ожидания
* и что пользователь знает, что для продолжения необходимо
* закрыть окно предварительного просмотра.
oSheet.PrintPreview()

* Теперь поместите окно Excel в NoTopMost, чтобы VFP мог выйти
* вперед.
= SetWindowPos(hWndXL, hwnd_notopmost, 0,0,0,0, swp_nosize + ;
    swp_nomove)

* Переместите окно VFP вперед, затем установите для него
* NoTopMost, чтобы другие приложения могли быть перенесены вперед.
= SetWindowPos(hWndVFP, hwnd_topmost, 0,0,0,0, swp_nosize + ;
    swp_nomove)
= SetWindowPos(hWndVFP, hwnd_notopmost, 0,0,0,0, swp_nosize + ;
    swp_nomove)
MESSAGEBOX('Fox снова активен!')
```

Это может работать для многих приложений, особенно если вы разместите окно сообщения с объяснением того, что делать, непосредственно перед вызовом этой процедуры.

Печать

Немного менее проблематично, чем PrintPreview, распечатать электронную таблицу. Основная часть свойств, управляющих печатью, задается в объекте PageSetup. Когда вы будете готовы к печати, вызывается метод PrintOut. Хотя у него есть некоторые параметры для управления номерами начальной и конечной страниц, количеством копий и устройством, на которое отправляется вывод (имя принтера, режим предварительного просмотра или имя файла), он отлично справляется со своей задачей без каких-либо параметров. Предполагается, что вы хотите, чтобы все это было распечатано на принтере по умолчанию.

```
oSheet.Printout()
```

Файл справки Excel VBA прекрасно объясняет параметры; см. раздел «Метод PrintOut».

Сохранение данных в разных форматах

В Excel 2000 определены константы для 41 различного формата (хотя не все они могут быть доступны в зависимости от нескольких факторов, таких как язык [например, американский английский] или то, установил ли их пользователь и т. д.). Многие из этих форматов представляют собой вариации темы электронных таблиц, включая восемь версий Excel и семь версий WK*. Также доступны форматы CSV, SYLK, DBF, DIF, текста и HTML и другие.

Метод SaveAs принимает множество параметров, но наиболее полезными являются первые два. Первый параметр – это новое имя файла. Имя файла должно быть полным; Excel не знает настройки VFP SET DEFAULT. Второй параметр – это числовое значение, представляющее формат. Разумеется, нам понадобится таблица констант, представленная в Таблице 7. Мы покажем некоторые из них, остальные вы сможете найти в Обзорщике объектов.

Вы заметите значение xlWorkbookNormal. Используйте это, чтобы сохранить копию файла в формате текущей версии.

Таблица 7. Константы формата файла для Excel. Excel поддерживает сохранение во многих различных форматах. Вот некоторые из 41 в Excel 2000.

Константа	Значение	Константа	Значение
xlCSV	6	xlSYLK	2
xlDBF3	8	xlTextWindows	20
xlDIF	9	xlWK4	38
xlExcel9795	43	xlWorkbookNormal	-4143

7.8 Собираем все вместе



Листинг 2 показывает программу (XLSampel.PRG в файлах загрузки для разработчиков, доступных на сайте www.hentzenwerke.com), которая создает рабочую книгу с рабочими листами для истории заказов первых трех клиентов (первые три – это просто числа, выбранные из воздуха; вы можете поиграть со счетчиком циклов, если хотите). В этом примере рассматривается ввод данных,

добавление итоговой формулы, форматирование ячеек, настройка листов для печати, просмотр PrintPreview и, наконец, сохранение электронной таблицы (см. также рис. 11).

Листинг 2. Книга истории клиентов Tasmanian Traders. Этот пример охватывает многие темы этой главы, включая добавление данных и формул, форматирование ячеек, сохранение электронной таблицы, предварительный просмотр и другие.

```
* XLSample1.PRG
* © 2000, Tamar E. Granor and Della Martin
* From: Microsoft Office Automation with Visual FoxPro
* Hentzenwerke Publishing. www.hentzenwerke.com

* Это пример «Собираем все вместе» из главы 7.
* Поместите информацию о заказах для нескольких клиентов на
* лист Excel. Измените константу LoopCounter, чтобы изменить
* количество клиентов, введенных в Excel.

* Удалите все существующие ссылки на серверы.
* Это предотвращает потерю памяти оставшимися экземплярами.
RELEASE ALL LIKE o*

#DEFINE xlEdgeBottom          9
#DEFINE xlEdgeTop             8
#DEFINE xlContinuous          1
#DEFINE xlMedium              -4138
#DEFINE xlHAlignCenter        -4108
#DEFINE xlHAlignCenterAcrossSelection 7
#DEFINE xlVAlignBottom        -4107
#DEFINE xlAutomatic           -4105
#DEFINE xlPortrait            1
#DEFINE autoIn2Pts            72

#DEFINE swp_nosize            1
#DEFINE swp_nomove            2
#DEFINE hwnd_topmost         -1
#DEFINE hwnd_notopmost       -2

#DEFINE LoopCounter          3

LOCAL oBook, oRange, oSheet

* Откройте представление «История заказов», которое
* содержит сводку заказов для одного клиента.
CLOSE DATA
OPEN DATABASE _SAMPLES + "\TASTRADE\DATA\Tastrade"
USE CUSTOMER IN 0
SELECT 0
USE "Order History" ALIAS OrderHistory

* Добавьте книгу, используя настройки по умолчанию.
oExcel = CREATEOBJECT("Excel.Application")
oExcel.Visible = .T.
oBook = oExcel.Workbooks.Add()
oSheet = oBook.ActiveSheet

* Добавьте стили в книгу.
WITH oBook
    .Styles.Add("Bold")
    WITH .Styles["Bold"]
        .Font.Bold = .T.
        .Font.Size = 12
        .HorizontalAlignment = xlHAlignCenter
        .VerticalAlignment = xlVAlignBottom
```

```

        .WrapText = .T.
    ENDWITH
ENDWITH

FOR I = 1 TO LoopCounter

    REQUERY()

    WITH oSheet
        * Назовите лист с идентификатором клиента.
        .Name = Customer.Customer_ID
        .Select()

        * Поместите имя клиента вверху.
        .Range("A2").Value = Customer.Company_Name

        * Поместите заголовки столбцов в строку 5.
        .Range("A5").Value = "Order Number"
        .Range("B5").Value = "Date"
        .Range("C5").Value = "Amount"

        oRange = .Range("A6:C6")
    ENDWITH

    * Просматривайте заказы и отправляйте данные.
    SCAN
        WITH oRange
            .Columns[1].Value = Order_Id
            .Columns[2].Value = Order_Date
            .Columns[3].Value = Ord_Total
        ENDWITH

        * Переместить диапазон вниз на одну строку.
        oRange = oRange.Offset(1,0)
    ENDSCAN

    * Теперь добавьте итоговую строку.
    nLastRow = oRange.Row && Свойство Row всегда дает первую строку диапазона.
        && Этот диапазон имеет только одну строку.
    cLastRow = ALLTRIM(STR(nLastRow))

    WITH oSheet
        .Cells[ nLastRow , 1 ] = "Total"

        * Необходимо преобразовать nLastRow в char для использования
        * в формуле суммы
        .Cells[ nLastRow , 3 ].Formula = ;
            " =SUM( C6:C" + LTRIM(STR(nLastRow - 1)) + " )"
    ENDWITH

    * Начните форматировать лист
    WITH oSheet
        * Примените полужирный стиль к заголовкам столбцов
        WITH .Range("A5:C5")
            .Style = "Bold"
            .Borders[xlEdgeBottom].LineStyle = xlContinuous
        ENDWITH

        * Применить жирный стиль к метке «Итого»
        .Range("A" + cLastRow).Style = "Bold"
        .Range("A" + cLastRow + ":C" + cLastRow).Borders[xlEdgeTop].LineStyle = ;
            xlContinuous

        * Примените полужирный стиль к имени клиента, а затем переопределите

```

```

* горизонтальное выравнивание, чтобы распределить его по столбцам.
.Range("A2").Style = "Bold"
.Range("A2:C2").HorizontalAlignment = xlHAlignCenterAcrossSelection

* Изменение формата столбца даты
.Range("B6:B" + cLastRow).NumberFormat = "Dd-Mmm-YYY"

* Исправить ширину столбцов.
.Range("A5:C" + cLastRow).Columns.AutoFit()

* По какой-то причине первый столбец не расширяется достаточно,
* если заголовок столбца не переносит один символ последнего слова.
* Добавьте еще один символ к ширине столбца.
.Range("A5").Columns.ColumnWidth = .Range("A5").Columns.ColumnWidth + 1

* AutoFit гарантирует, что наибольшее значение едва уместится.
* Мы предпочитаем, чтобы между двумя последними столбцами было
* немного больше места, поэтому добавьте еще один символ в каждый
* из этих столбцов.
.Range("B5").Columns.ColumnWidth = .Range("B5").Columns.ColumnWidth + 1
.Range("C5").Columns.ColumnWidth = .Range("C5").Columns.ColumnWidth + 1

* Теперь настройте страницу для печати.
WITH .PageSetup
    * Не думайте, что это значения по умолчанию;
    * это то, что пользователи могут изменить!
    .Orientation = xlPortrait

    .FitToPagesTall = 1
    .FitToPagesWide = 1

    .CenterHorizontally = .T.
    .CenterVertically = .T.

    .TopMargin = 1.0 * autoIn2Pts
    .BottomMargin = 1.0 * autoIn2Pts
    .LeftMargin = 1.5 * autoIn2Pts
    .RightMargin = 1.5 * autoIn2Pts
    .HeaderMargin = 0.5 * autoIn2Pts
    .FooterMargin = 0.5 * autoIn2Pts

    .LeftHeader = "TasTrade Client ID &A"
    .RightHeader = "Page &P of &N"
    .RightFooter = "Printed On &D"
ENDWITH

ENDWITH

* Добавьте еще один лист для следующего клиента
IF I <> LoopCounter
    oSheet = oBook.Worksheets.Add()
ENDIF

* Получить следующего клиента
SKIP IN Customer

ENDFOR

* Предварительный просмотр книги.
* Объявить API...
* FindWindowA возвращает дескриптор окна из заголовка окна.
DECLARE LONG FindWindowA IN WIN32API STRING class, STRING title
* SetWindowPos перемещает окно вверх, используя дескриптор окна.

```



```

DECLARE SetWindowPos IN WIN32API LONG HWND, LONG hwndafter, ;
    LONG x, LONG Y, LONG cx, LONG cy, LONG flags

* Запишите дескрипторы окон для Excel и VFP.
hWndXL = FindWindowA("XLMAIN", oExcel.Caption)
hWndVFP = FindWindowA("VFP66400000", _VFP.Caption)

* Переместите VFP наверх и попросите пользователя закрыть PrintPreview.
= SetWindowPos(hWndVFP, hwnd_topmost, 0,0,0,0, swp_nosize + swp_nomove)
= MessageBox("Закройте это сообщение, и Excel откроется." + CHR(13) + ;
    CHR(13) + "Закройте предварительный просмотр печати, "+ ;
    "когда будете готовы вернуться в FoxPro.")

* Теперь поднимите окно Excel наверх.
= SetWindowPos(hWndVFP, hwnd_notopmost, 0,0,0,0, swp_nosize + swp_nomove)
= SetWindowPos(hWndXL, hwnd_topmost, 0,0,0,0, swp_nosize + swp_nomove)

* Вызовите метод предварительного просмотра, чтобы отобразить первого клиента.
* Убедитесь, что ваш код допускает состояние ожидания и что пользователь знает,
* что для продолжения необходимо закрыть окно предварительного просмотра.

oSheet.PrintPreview()

* Когда пользователь закрывает PrintPreview, он возвращается сюда.
* Поднимите VFP наверх, а затем уведомите пользователя.
= SetWindowPos(hWndXL, hwnd_notopmost, 0,0,0,0, swp_nosize + swp_nomove)
= SetWindowPos(hWndVFP, hwnd_topmost, 0,0,0,0, swp_nosize + swp_nomove)
= MessageBox("Создание файла Excel завершено." + CHR(13) + CHR(13) + ;
    "Готово сохранить файл Excel.")

* Сброс Excel и VFP до NoTopMost.
= SetWindowPos(hWndXL, hwnd_notopmost, 0,0,0,0, swp_nosize + swp_nomove)
= SetWindowPos(hWndVFP, hwnd_notopmost, 0,0,0,0, swp_nosize + swp_nomove)

* Сохраните файл.
* Настройте имя файла
XLFile = FULLPATH(CURDIR()) + "XLSample1.XLS"
XLFileJustName = JUSTSTEM(XLFile)

* Определите, следует ли использовать SaveAs или Save.
IF oBook.Name <> XLFileJustName

    * Если файл уже существует, удалите его.
    IF FILE(XLFile)
        ERASE (XLFile)
    ENDIF

    * Сохраните его, не опасаясь диалогового окна пользователя.
    oBook.SaveAs(XLFile)

ELSE

    * Сохраните его, поскольку он уже был сохранен с помощью SaveAs.
    oBook.Save()

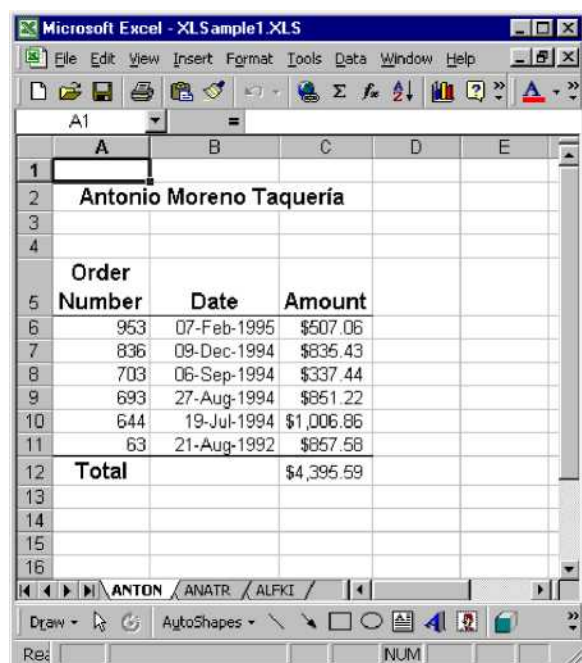
ENDIF

=MessageBox("Файл Excel сохранен как:" + CHR(13) + CHR(13) + XLFile)

oExcel.Quit()
RELEASE oExcel

USE IN OrderHistory
USE IN Customer

```



	A	B	C	D	E
1					
2	Antonio Moreno Taqueria				
3					
4					
5	Order				
6	Number	Date	Amount		
7	953	07-Feb-1995	\$507.06		
8	836	09-Dec-1994	\$835.43		
9	703	06-Sep-1994	\$337.44		
10	693	27-Aug-1994	\$851.22		
11	644	19-Jul-1994	\$1,006.86		
12	63	21-Aug-1992	\$857.58		
13	Total		\$4,395.59		
14					
15					
16					

Рисунок 11. Готовая книга истории клиентов Tasmanian Traders. В этом примере описывается ввод данных, добавление итоговой формулы, форматирование ячеек, настройка листов для печати, просмотр PrintPreview и, наконец, сохранение электронной таблицы.

В этой главе мы многое рассмотрели, но Excel может предложить еще больше. Мы работали над механизмом создания книги со стандартными функциями электронных таблиц. В следующей главе мы рассмотрим более продвинутые функции Excel, которые помогают вашим пользователям анализировать свои данные.

Глава 8. Расширенные функции Excel

Excel имеет множество расширенных функций для анализа данных, стандартизации листов и добавления изображений и фигур.

Функции анализа Excel очень мощные. Сводная таблица превосходна благодаря своей способности анализировать и форматировать данные в перекрестных таблицах. Анализ «что если» упрощается благодаря таблицам данных, позволяющим видеть несколько результатов, используя несколько входных данных. Поиск цели — это способ найти один из параметров функции. В Excel есть еще больше аналитических инструментов, и они помогут вам начать работу.

В расширенную категорию также включены использование шаблонов и Автоматизация макросов. Это помогает стандартизировать ваши электронные таблицы.

Наконец, вы можете добавлять фигуры и графику, чтобы ваши рабочие листы действительно выделялись.

8.1 Сводные таблицы

Сводные таблицы (PivotTables) — это круто. Это перекрестные таблицы на стероидах, созданные на основе простого списка данных, который содержит категории перекрестных таблиц в данных в каждом столбце — тот же самый список, который вам придется сканировать и условно задать множество переменных или написать сложный оператор SQL для построения кросс-таблицы. В VFP есть мастера кросс-таблиц и сводных таблиц, но они гораздо более ограничены, чем механизм Excel.

На первый взгляд, простые сводные таблицы могут сопоставлять данные, например общее количество товаров для каждой категории продуктов, проданных за каждый месяц. Сводные таблицы звучат еще лучше, когда вы обнаружите, что они могут суммировать несколько полей для перекрестных таблиц, суммирующих общее количество проданных товаров и общую цену продажи по каждому проданному продукту за месяц. Они также будут создавать страницы, на которых будет суммироваться общее количество товаров и цена продажи каждого проданного продукта по месяцам, показывая «страницу» для каждого года. Вы также можете передумать: вместо того, чтобы показывать итоговые суммы по каждому продукту по месяцам, вы можете изменить одно свойство, чтобы отображать продукты, проданные в стране покупателя. Если это было недостаточно круто, это интерактивно для пользователя: вы передаете данные в Excel и настраиваете сводную таблицу, а ваши пользователи могут нарезать данные по своему усмотрению. Меньше кода для разработчика, больше возможностей для пользователя — лучше этого не найти!

Сводные таблицы настолько хороши, что в VFP 6.0 есть мастер сводных таблиц, но для его использования необходимо установить Excel и Microsoft Query. Мы обнаружили, что Query отличается темпераментностью с точки зрения программного обеспечения правильной установки и настройки на компьютере пользователя; если ваше приложение не может проверить, что Query будет работать, ваши пользователи могут (и будут) получать неприятные сообщения об ошибках, побуждающие их совершать гневные звонки в службу технической поддержки. Однако мы можем решить всю эту проблему, поместив данные в Excel непосредственно из FoxPro, используя Автоматизацию, а не запрос, а затем напрямую обратившись к Excel для создания сводной таблицы и управления ею.



В Excel имеется несколько источников данных для сводных таблиц: диапазон данных на листе, несколько диапазонов, которые сводная таблица будет объединять, другой отчет сводной таблицы или внешний источник данных. Доступ к внешнему источнику данных осуществляется через технологии оперативной аналитической обработки (OLAP), ODBC, ADO или DAO, в зависимости от источника данных. Поскольку COPY TO ... TYPE XL5 и автоматизировать Excel для открытия файла, созданного FoxPro в виде рабочего листа, очень легко, единственным источником данных для сводных таблиц, который мы рассмотрим, является диапазон на рабочем листе. Мы не будем пытаться охватить сложности Автоматизации OLAP, ODBC, ADO или DAO с помощью Excel. Хотя технически возможно использовать эти подходы (мы успешно реализовали некоторые из этих технологий в наших собственных приложениях), детализация тонкостей этих сложных технологий на самом деле является отдельной темой, входящей в фокус этой книги.

Данные сводной таблицы

Первым шагом в определении сводной таблицы является настройка данных на листе с использованием формата строк и столбцов, который Excel называет *списком* или *базой данных*. Не бойтесь; это похоже на то, что разработчики FoxPro называют базой данных. Есть только одно отличие: первая строка списка зарезервирована для меток столбцов, которые используются для обозначения сводной таблицы и используется как имя поля для доступа к данным в столбцах.

При программировании отчета с перекрестной таблицей список уникальных элементов данных в одном поле становится метками для столбцов, а список уникальных элементов данных в другом поле становится метками для строк. Пересечение этих строк и столбцов отражает сводку числовых данных, таких как сумма, среднее значение, количество или какая-либо другая числовая функция. Как и в отчете с перекрестными таблицами, сводная таблица использует по одному полю для свойств RowFields и ColumnFields и суммирует числовые данные, содержащиеся в свойстве DataFields. Мы вернемся к свойствам позже; прямо сейчас нам нужно настроить данные. Итак, мы знаем, что нам нужно как минимум три столбца (и позже мы увидим, что можно использовать гораздо больше).



База данных TasTrade идеально подходит для иллюстрации сводных таблиц. Мы проиллюстрируем примеры сводных таблиц таблицей, содержащей список всех позиций заказа. Таблицу Order_Line_Items мы могли бы взять практически как есть, за исключением того, что она содержит много кодов, для которых нам нужны текстовые описания. **Листинг 1** (XLData.PRG в файлах загрузки для разработчиков, доступных по адресу www.hentzenwerke.com) показывает, как создать курсор, скопировать его в файл XLS, открыть файл XLS в Excel, а затем подогнать столбцы, чтобы мы могли лучше его читать. **На рисунке 1** показана часть полученного рабочего листа.

Листинг 1. Создание примера курсора для использования в сводных таблицах.

```
* XLData.prg
* © 2000, Tamar E. Granor and Della Martin
* From: Microsoft Office Automation with Visual FoxPro
* Hentzenwerke Publishing. www.hentzenwerke.com
```

* Удалите все существующие ссылки на серверы.
 * Это предотвращает потерю памяти оставшимися экземплярами.
 RELEASE ALL LIKE o*

* В демонстрационных целях сделайте oExcel и oBook
 * доступны после выполнения этой программы.
 PUBLIC oExcel, oBook

OPEN DATABASE (_SAMPLES + "\TasTrade\Data\TasTrade")

SET ENGINEBEHAVIOR 70

```
SELECT Customer.Company_Name, ;
       Customer.Country, ;
       CMONTH(Orders.Order_Date) AS Order_Month, ;
       YEAR(Orders.Order_Date) AS Order_Year, ;
       Category.Category_Name, ;
       Products.Product_Name, ;
       Order_Line_Items.Quantity, ;
       Order_Line_Items.Unit_Price * Order_Line_Items.Quantity ;
       AS Total_Price;
FROM Orders, Order_Line_Items, Customer, Products, Category ;
WHERE Order_Line_Items.Order_ID = Orders.Order_ID ;
      AND Orders.Customer_ID = Customer.Customer_ID ;
      AND Order_Line_Items.Product_ID = Products.Product_ID ;
      AND Products.Category_ID = Category.Category_ID ;
INTO CURSOR Pivot
```

```
LastLine = ALLTRIM(STR(_TALLY + 1))
COPY TO (CURDIR() + "Pivot") TYPE XL5
```

oBook = GETOBJECT(CURDIR() + "Pivot.XLS")

* Откройте книгу и расположите все столбцы наилучшим образом.
 WITH oBook
 oExcel = .Application
 .Application.Visible = .T.
 .Windows[1].Activate()
 .Sheets[1].Range("A1:H" + LastLine).Columns.AutoFit()
 ENDWITH

	A	B	C	D	E	F	G	H
1	company_name	country	order_month	order_year	category_name	product_name	quantity	total_price
2	B's Beverages	UK	May	1992	Seafood	Ikura	5	155
3	Cactus Comidas para llevar	Argentina	May	1992	Seafood	Boston Crab Meat	998	18363.2
4	Cactus Comidas para llevar	Argentina	May	1992	Dairy Products	Raclette Courdavault	24	925.2
5	Cactus Comidas para llevar	Argentina	May	1992	Grains/Cereals	Wimmers gute Semmelknödel	10	332.5
6	Folk och få HB	Sweden	May	1992	Dairy Products	Gorgonzola Telino	15	120
7	Folk och få HB	Sweden	May	1992	Beverages	Chartreuse verte	19	239.4
8	Folk och få HB	Sweden	May	1992	Dairy Products	Flotemysost	15	225
9	Simons bistro	Denmark	May	1992	Seafood	Carnarvon Tigers	12	524.4
10	Vaffeljernet	Denmark	May	1992	Meat/Poultry	Thüringer Rostbratwurst	35	3010
11	Vaffeljernet	Denmark	May	1992	Condiments	Vegie-spread	6	184.2
12	Wartian Herkku	Finland	May	1992	Confections	Tarte au sucre	10	340
13	Franchi S.p.A	Italy	May	1992	Seafood	Kontu	10	42
14	Franchi S.p.A	Italy	May	1992	Confections	Valkoinen suklaa	4	45.2
15	Morgenstern Gesundkost	Germany	May	1992	Dairy Products	Queso Manchego La Pastora	4	104
16	Morgenstern Gesundkost	Germany	May	1992	Meat/Poultry	Perth Pasties	30	867
17	Morgenstern Gesundkost	Germany	May	1992	Condiments	Vegie-spread	20	814
18	Furia Bacalhau e Frutos do Mar	Portugal	May	1992	Produce	Tofu	20	324
19	Furia Bacalhau e Frutos do Mar	Portugal	May	1992	Confections	Sir Rodney's Scones	15	105
20	Furia Bacalhau e Frutos do Mar	Portugal	May	1992	Produce	Manjimup Dried Apples	20	742
21	Seven Seas Imports	UK	May	1992	Grains/Cereals	Tunnbrod	70	420
22	Seven Seas Imports	UK	May	1992	Produce	Manjimup Dried Apples	10	370
23	Simons bistro	Denmark	May	1992	Beverages	Ipoh Coffee	10	320
24	Simons bistro	Denmark	May	1992	Dairy Products	Flotemysost	10	150
25	Wellington Importadora	Brazil	May	1992	Seafood	Carnarvon Tigers	12	524.4
26	Wellington Importadora	Brazil	May	1992	Confections	Teatime Chocolate Biscuits	15	96
27	Wellington Importadora	Brazil	May	1992	Seafood	Inlaid Sil	10	133
28	Wido Delicatessen	Germany	May	1992	Confections	Teatime Chocolate Biscuits	10	242.6

Рисунок 1. Образец листа для примеров сводной таблицы. Этот лист содержит 2822 строки. Большинство столбцов содержат повторяющиеся данные, и при использовании в качестве поля строки или столбца уникальные значения внутри столбца становятся заголовками.

Этот лист содержит 2822 строки и множество полей, которые можно использовать в качестве строк и столбцов сводной таблицы. Небольшое напоминание: в отличие от таблиц FoxPro с неограниченным количеством записей, Excel имеет ограничение в 65 536 строк.

Создание сводной таблицы.

Если у вас есть список на листе, вы можете создать сводную таблицу. Для этого используйте метод `PivotTableWizard` класса `Worksheet`. Метод `PivotTableWizard` чрезвычайно гибок (и, следовательно, сложен).

Однако для использования данных, содержащихся на листе, необходимы только первые шесть параметров. Соответствующий синтаксис `PivotTableWizard` выглядит следующим образом:

```
oBook.Sheets[1].PivotTableWizard( nSourceType, oSourceData,
                                   oTableDestination,
                                   cTableName, lRowGrandTotals,
                                   lColumnGrandTotals )
```

nSourceType	Numeric	Числовое значение, представляющее тип исходных данных. Константа, которую мы здесь обсудим, – это <code>xlDatabase</code> (1). Другими константами являются <code>xlConsolidation</code> (3), <code>xlPivotTable</code> (-4148) и <code>xlExternal</code> (2).
oSourceData	Object	Объект <code>Range</code> , содержащий данные для сводной таблицы.
oTableDestination	Object	Объект <code>Range</code> , содержащий местоположение сводной таблицы (вы можете указать только верхнюю левую ячейку; вам не нужно определять размер).
cTableName	Character	Имя таблицы.
lRowGrandTotals	Logical	Указывает, следует ли включать общие итоги по строкам (наиболее полезно, если используется несколько полей строк).
lColumnGrandTotals	Logical	Указывает, следует ли включать общие итоги по столбцам (наиболее полезно, только если используется несколько полей столбцов).

Вот источник для добавления сводной таблицы в предыдущую электронную таблицу, начиная с ячейки J1:

```
#DEFINE xlDatabase 1
```

```
oSourceData = oBook.Sheets[1].Range("A1:H2822")
oDestination = oBook.Sheets[1].Range("J1")
oPivotTable = oBook.Sheets[1].PivotTableWizard(xlDatabase, ;
        oSourceData, oDestination, "SalesAnalysis", .T., .T.)
```

После запуска метода PivotTableWizard диапазон J1:K2 выделяется жирной синей рамкой. В результате был создан объект сводной таблицы. В отличие от большинства методов мастера, которые создают готовый объект, метод PivotTableWizard создает пустой объект, для завершения которого требуется дополнительное программирование.

Мастер сводных таблиц не содержит параметров для заполнения коллекции PivotFields, которая представляет собой коллекцию всех полей, используемых в сводной таблице. Для нас это нормально, поскольку мы не можем себе представить, насколько сложным будет синтаксис! Используйте метод AddFields объекта сводной таблицы, который имеет следующий синтаксис:

```
oPivotTable.AddFields( cRowField, cColumnField, cPageField,
                        lAddFields)
```

cRowField	Character	Имя поля, используемого для создания строк сводной таблицы. Имя поля представляет собой строку символов, соответствующую метке столбца в первой строке списка. (Необязательный)
cColumnField	Character	Имя поля, используемого для создания столбцов сводной таблицы. (Необязательный)
cPageField	Character	Имя поля, используемого для создания страниц сводной таблицы. Думайте о страницах как об отфильтрованном представлении, а не как о странице с вкладками. Страницы выбираются из раскрывающегося списка в верхней части сводной таблицы. (Необязательный)
lAddFields	Logical	Указывает, следует ли добавлять поля к существующим полям (.T.) или заменять поля, содержащиеся в сводной таблице (.F.). По умолчанию – ложь. (Необязательный)

В качестве примера задайте в строках поле страны, а в столбцах – имя категории с помощью следующего кода:

```
oPivotTable.AddFields("country", "category_name")
```

Эта команда устанавливает для RowField и ColumnField поля Country и Category_name. Поле данных для суммирования не установлено. Это достигается путем прямого доступа к соответствующему объекту PivotField. Коллекция PivotFields создается при запуске метода PivotTableWizard и содержит по одному объекту для каждого поля. Метод AddFields сводной таблицы использует перечисленные имена полей и устанавливает определенные свойства соответствующих объектов PivotField, чтобы указать, что поля используются для RowFields или ColumnFields. Одним из более чем 50 свойств объекта PivotField является свойство Orientation, которое определяет расположение поля в сводной таблице. Задайте для свойства Orientation одно из следующих значений: xlColumnField (2), xlDataField (4), xlHidden (0), xlPageField (3) или xlRowField (1). Чтобы суммировать поле Quantity, введите следующие команды:

```
#DEFINE xlDataField 4
oPivotTable.PivotFields["Quantity"].Orientation = xlDataField
```

Полученная сводная таблица показана на рисунке 2. Создание её на тестовой машине Деллы заняло всего полторы секунды (после того, как данные были

собраны). Обратите внимание, что метки полей строк и столбцов представляют собой раскрывающиеся списки. Эта интерактивная функция позволяет вашим пользователям выбирать, какие столбцы или строки отображать. Например, они могут выбрать только категории «Напитки» и «Кондитерские изделия» для США, Канады и Мексики. Для этого пользователь щелкает раскрывающийся список, а затем проверяет те элементы в списке, которые он хотел бы видеть.

Это можно сделать в коде, манипулируя коллекцией PivotItems объекта PivotField. Каждый PivotField имеет коллекцию PivotItems, каждая из которых соответствует строке или столбцу. Метки заголовков используются в качестве индексного имени. Одним из 19 свойств объекта PivotItem является свойство Visible. Установка параметра Visible на .F. удаляет его из поля зрения. Следующий код оставляет видимыми только столбцы «Напитки» и «Кондитерские изделия» для стран Северной Америки.

```
WITH oPivotTable.PivotFields("category_name")
    .PivotItems("Condiments").Visible = .F.
    .PivotItems("Dairy Products").Visible = .F.
    .PivotItems("Grains/Cereals").Visible = .F.
    .PivotItems("Meat/Poultry").Visible = .F.
    .PivotItems("Produce").Visible = .F.
    .PivotItems("Seafood").Visible = .F.
ENDWITH
WITH oPivotTable.PivotFields("country")
    .PivotItems("Argentina").Visible = .F.
    .PivotItems("Austria").Visible = .F.
    .PivotItems("Belgium").Visible = .F.
    .PivotItems("Brazil").Visible = .F.
    .PivotItems("Denmark").Visible = .F.
    .PivotItems("Finland").Visible = .F.
    .PivotItems("France").Visible = .F.
    .PivotItems("Germany").Visible = .F.
    .PivotItems("Ireland").Visible = .F.
    .PivotItems("Italy").Visible = .F.
    .PivotItems("Norway").Visible = .F.
    .PivotItems("Poland").Visible = .F.
    .PivotItems("Portugal").Visible = .F.
    .PivotItems("Spain").Visible = .F.
    .PivotItems("Sweden").Visible = .F.
    .PivotItems("Switzerland").Visible = .F.
    .PivotItems("UK").Visible = .F.
    .PivotItems("Venezuela").Visible = .F.
ENDWITH
```

	J	K	L	M	N	O	P	Q	R	S
1	Sum of quantity	category								
2	country	Beverages	Condiments	Confections	Dairy Products	Grains/Cereals	Meat/Poultry	Produce	Seafood	Grand Total
3	Argentina	118	50	57	78	64		33	1052	1452
4	Austria	1058	1010	600	807	655	362	473	876	5841
5	Belgium	423	206	358	411	178	89	204	227	2096
6	Brazil	1262	770	768	505	315	205	194	724	4043
7	Canada	364	506	580	373	311	221	154	276	2785
8	Denmark	339	261	221	99	20	250	161	294	1645
9	Finland	200	94	159	204	140	116	93	143	1149
10	France	808	374	830	453	368	312	381	696	4222
11	Germany	1832	1380	1858	1498	890	727	772	1858	10815
12	Ireland	317	216	119	459	172	316	161	757	2517
13	Italy	247	112	174	174	158	72	105	166	1208
14	Mexico	100288	83	146	297	93	101	126	158	101292
15	Norway	64	28	26	38		5	18	58	237
16	Poland	65	42	27	35		3	28	15	215
17	Portugal	148	189	112	19	104	70	73	44	759
18	Spain	226	122	166	83	157	129	50	277	1210
19	Sweden	743	274	408	288	235	304	292	611	3155
20	Switzerland	314	160	176	162	232	183	161	375	1763
21	UK	627	431	431	653	445	324	318	600	3829
22	USA	2072	1206	1837	1657	993	1453	759	1895	11862
23	Venezuela	781	271	721	495	269	302	264	961	4064
24	Grand Total	112316	7785	9774	8788	5799	5624	4820	12053	166959

Рисунок 2. Простая сводная таблица. Всего за несколько строк кода была создана простая перекрестная таблица. Обратите внимание на раскрывающиеся списки е Country и Category_Name, ваши пользователи могут прекрасно провести время, настраивая данные.

Как видите, к каждому пункту нужно подходить индивидуально, а значит, нужно хорошо знать свои данные. Но поскольку полученная сводная таблица полностью интерактивна, возможно, ваши пользователи захотят манипулировать полями в Excel вместо того, чтобы писать основной интерфейс сводной таблицы.

Использование цикла FOR EACH для перемещения по всем элементам PivotItem может немного облегчить жизнь. Проверьте значение поля Name элемента PivotItem, чтобы определить, следует ли устанавливать свойство Visible. Следующий код дает те же результаты, что и предыдущий пример кода:

```
FOR EACH oItem IN
oPivotTable.PivotFields("category_name").PivotItems
    IF NOT oItem.Name $ ("Beverages Confections")
        oItem.Visible = .F.
    ENDIF
ENDFOR
FOR EACH oItem IN oPivotTable.PivotFields("country").PivotItems
    IF NOT oItem.Name $ ("Canada Mexico USA")
        oItem.Visible = .F.
    ENDIF
ENDFOR
```

Давайте немного усложним работу со сводной таблицей. Чтобы эта сводная таблица суммировала поле Total_Price, используйте следующее:

```
#DEFINE xlDataField 4
oPivotTable.PivotFields["Total_Price"].Orientation = xlDataField
```

Рисунок 3 показывает результаты (теперь вы понимаете, почему мы удалили множество строк и столбцов, чтобы показать пример разумного размера!). Фактически, вы можете иметь любое количество DataFields. Установка свойства Orientation для PivotField не влияет на состояние других PivotField.

J	K	L	M	N
country	Data	category_name		
Canada	Sum of quantity	Beverages	Confections	Grand Total
	Sum of total_price	364	580	944
Mexico	Sum of quantity	13829.8	13316.1	27145.9
	Sum of total_price	100288	146	100434
USA	Sum of quantity	1907975.7	2282.95	1910258.65
	Sum of total_price	2072	1837	3909
	Sum of total_price	75442.15	42674.75	118116.9
Total Sum of quantity		102724	2563	105287
Total Sum of total_price		1997247.65	58273.8	2055521.45

Рисунок 3. Все становится немного сложнее. Мы отключили видимость для ряда строк и столбцов и добавили второй DataField.

Этим значениям Total_Price требуется небольшое форматирование, чтобы они выглядели как денежная единица. Вы можете установить форматирование для всего PivotField, используя свойство NumberFormat объекта PivotField (это описано в главе 7 «Основы Excel», необходимые коды перечислены в таблице 2 в этой главе). Изменение числового формата на денежный работает следующим образом:

```
oPivotTable.PivotFields["Sum of Total_Price"].NumberFormat =
"$###,###,###.##0"
```

Обратите внимание, что имя поля меняется на сводное значение. Это связано с тем, что для одного и того же поля могут быть два разных расчета. Например, мы можем добавить поле средней общей цены. Сначала мы снова установим ориентацию поля Total_Price. Это добавит *второй* экземпляр поля сводных данных. Его имя по умолчанию – «Sum of Total_Price2». Изменение имени также приводит к изменению отображаемой метки, так что это должно быть следующим шагом, тогда как фактическая настройка функции – это последний шаг. Установите для свойства Function одну из констант, показанных в таблице 1.

Таблица 1. Значения свойств функции. Свойство Function управляет тем, как вычисляется DataField.

Констаета	Значение	Констаета	Значение
xlAverage	-4106	xlStDev	-4155
xlCount	-4112	xlStDevP	-4156
xlCountNums	-4113	xlSum	-4157
xlMax	-4136	xlVar	-4164
xlMin	-4139	xlVarP	-4165
xlProduct	-4149		

Код для добавления третьего сводного поля и установки для него среднего значения поля Total_Price выглядит следующим образом:

```
#DEFINE xlDataField 4
#DEFINE xlAverage -4106

* Добавьте еще одно поле сводки Total_Price
oPivotTable.PivotFields["Total_Price"].Orientation = xlDataField

* Изменить его имя
oPivotTable.PivotFields["Sum of Total_Price2"].Name = ;
    "Average Total Sale"
```

* Установите формулу для усреднения данных, а не для суммирования
* данных

```
oPivotTable.PivotFields["Average Total Sale"].Function = xlAverage
```

* Отформатируйте его так, чтобы числа выглядели как денежная
* единица.

```
oPivotTable.PivotFields["Average Total Sale"].NumberFormat = ;  
"$###,###,###.##0"
```

Теперь, чтобы по-настоящему оценить возможности сводных таблиц, вы можете добавить второй RowField. Какова цель второго поля? Чтобы увидеть продажи каждой страны с разбивкой по годам. Здесь картинка стоит тысячи слов; Следующая строка кода добавляет поле Order_Year в качестве еще одного поля RowField и объединяет его с предыдущими примерами, создавая рисунок 4 .

```
#DEFINE xlRowField 1
```

```
oPivotTable.PivotFields["Order_Year"].Orientation = xlRowField
```

			category_name		
country	Data	order_year	Beverages	Confections	Grand Total
Canada	Sum of quantity	1992	61	85	146
		1993	219	201	420
		1994	35	171	206
		1995	49	123	172
	Sum of total_price	1992	\$562.10	\$1,871.0	\$2,433.10
		1993	\$12,431.20	\$4,683.50	\$17,114.70
		1994	\$210.0	\$2,243.30	\$2,453.30
		1995	\$626.50	\$4,518.30	\$5,144.80
	Average Total Sale	1992	\$187.37	\$623.67	\$405.52
		1993	\$2,071.87	\$390.29	\$950.82
		1994	\$70.0	\$320.47	\$245.33
		1995	\$208.83	\$1,506.10	\$857.47
Canada Sum of quantity			364	580	944
Canada Sum of total_price			\$13,829.80	\$13,316.10	\$27,145.90
Canada Average Total Sale			\$921.99	\$532.64	\$678.65
Mexico	Sum of quantity	1992		27	27
		1993	66	40	106
		1994	182	72	254
		1995	40	7	47
	Sum of total_price	1992		\$216.0	\$216.0
		1993	\$796.20	\$560.0	\$1,356.20
		1994	\$6,729.50	\$1,442.55	\$8,172.05
		1995	\$450.0	\$64.40	\$514.40
	Average Total Sale	1992	\$1,900,000.0		\$1,900,000.0
		1993		\$108.0	\$108.0
		1994	\$159.24	\$280.0	\$193.74
		1995	\$672.95	\$360.64	\$583.72
Mexico Sum of quantity			100288	146	100434
Mexico Sum of total_price			\$1,907,975.70	\$2,282.95	\$1,910,258.65
Mexico Average Total Sale			\$105,998.65	\$253.66	\$70,750.32
USA	Sum of quantity	1992	191	190	381
		1993	615	310	925
		1994	796	899	1695
		1995	470	438	908
	Sum of total_price	1992	\$7,070.0	\$2,819.20	\$9,889.20
		1993	\$14,599.40	\$8,852.60	\$23,452.0
		1994	\$31,604.75	\$21,474.55	\$53,079.30
		1995	\$22,168.0	\$9,528.40	\$31,696.40
	Average Total Sale	1992	\$1,178.33	\$281.92	\$618.08
		1993	\$811.08	\$553.29	\$689.76
		1994	\$929.55	\$631.60	\$780.58
		1995	\$1,477.87	\$680.60	\$1,092.98
USA Sum of quantity			2072	1837	3909
USA Sum of total_price			\$75,442.15	\$42,674.75	\$118,116.90
USA Average Total Sale			\$1,033.45	\$576.69	\$803.52
Total Sum of quantity			102724	2563	105287
Total Sum of total_price			\$1,997,247.65	\$58,273.80	\$2,055,521.45
Total Average Total Sale			\$18,841.96	\$539.57	\$9,605.24

Рисунок 4. Сложная сводная таблица с тремя DataFields и двумя RowFields. Это лишь малая часть возможностей сводных таблиц.

Здесь мы едва коснулись поверхности сводных таблиц. Сводные таблицы настолько сложны, что даже в файле справки VBA Excel в разделе «Объект сводной таблицы» есть такое замечание: «Поскольку программирование отчетов сводных таблиц может быть сложным, обычно проще всего записывать действия отчета сводных таблиц, а затем редактировать записанный код». Мы согласны; это единственный способ глубже погрузиться в сложности этой чрезвычайно богатой функции Excel.

8.2 Анализ «что если» с использованием таблиц данных

Самая простая форма анализа «что если» (What-if analysis) – добавить в электронную таблицу несколько ячеек данных вместе с формулой, которая использует эти ячейки данных, а затем изменить значения ячеек данных, чтобы увидеть, как изменится формула. Эта простая функция анализа радует многих пользователей Excel.

Недостаток этого подхода в том, что вы видите только один результат за раз. Иногда было бы неплохо увидеть результаты серии значений для одной из ячеек данных одновременно. Например, легче выяснить, за сколько продать товар, если вы видите розничные цены для различных значений наценки. Возможно, вы ищете виджет, стоимость которого составляет 5,96 доллара США. Вы хотели бы видеть цену продажи этого товара с наценкой 15, 20, 25, 30 и 35 процентов. Используя таблицу данных, вы можете построить диаграмму, подобную той, что показана на рисунке 5. Таблица данных создается из диапазона, содержащего формулу и значения для подстановки в формулу в столбце (или строке).

	A	B	C
1	Widget Cost		\$5.96
2			
3	0.15	\$6.85	
4	0.2	\$7.15	
5	0.25	\$7.45	
6	0.3	\$7.75	
7	0.35	\$8.05	
8			

Рисунок 5. Простая таблица данных. Таблицы данных отображают несколько значений одной (или двух) ячеек, используемых в формуле. В этом случае формула $=C1 * (1 + A3)$ сохраняется в ячейке B3. Последовательные строки таблицы заменяют ячейку A3 в формуле значением в столбце A.

Давайте посмотрим, как устроена эта таблица. Во-первых, у нас есть стоимость виджета в ячейке C1. Значения разметки для анализа находятся в ячейках A3:A7. Используемая формула: « $=C1 * (1 + A3)$ » находится в ячейке B3. Вот код для настройки данных. Если вы следите за книгой в FoxPro, обязательно закройте последнюю таблицу и начните новый лист с первыми тремя строками кода:

```
RELEASE oPivotTable, oBook, oSourceData, oDestination
oBook = oExcel.Workbooks.Add
oSheet = oBook.Sheets[1]
```

* Настройте и отформатируйте строку Widget Cost.

```
oSheet.Range("A1") = "Widget Cost"
oSheet.Range("C1") = 5.96
oSheet.Range("C1").NumberFormat = "$#,###.##0"
```

* Добавьте значения разметки для анализа.

```
oSheet.Range("A3") = ".15"
oSheet.Range("A4") = ".20"
oSheet.Range("A5") = ".25"
```



```
oSheet.Range("A6") = ".30"  
oSheet.Range("A7") = ".35"
```

* Введите формулу и отформатируйте ячейки (включая результаты
* таблицы данных), чтобы они выглядели как денежная единица.

```
oSheet.Range("B3") = "=C1 * (1 + A3)"  
oSheet.Range("B3:B7").NumberFormat = "$#,###.##0"
```

Один из способов заполнить остальные значения – слегка изменить эту формулу на « $=\$C\$1 * (1 + A3)$ » и скопировать ее в остальные ячейки. А еще лучше, мы могли бы использовать одну строку для настройки таблицы данных. Используйте метод Table объекта Range. Расположение данных было тщательно настроено, чтобы гарантировать, что таблица данных будет работать. Формула располагается на один столбец справа и на одну строку выше самого верхнего значения в столбце значений, подставляемых в формулу. Значение, которое мы заменяем в A3, может находиться где угодно в книге. Однако, поскольку это одно из рассматриваемых значений, имеет смысл сделать его похожим на первое значение в таблице. Это место в таблице может быть пустым, если значение находится где-то еще.

Метод Table работает с диапазоном, который представляет собой прямоугольную область, включающую формулу и значения; должно быть как минимум две строки и два столбца. В данном примере это диапазон A3:B7. Метод принимает два параметра. Каждый из них представляет собой диапазон, найденный в формуле: первый – это диапазон, который нужно заменить значениями, найденными в строках диапазона таблицы данных, а второй – это диапазон, который нужно заменить значениями, найденными в столбцах таблицы данных. В этом примере используются только столбцы. Следующая строка создает таблицу данных, показанную на рисунке 5:

```
oSheet.Range("A3:B7").Table("", oSheet.Range("A3"))
```

Так зачем создавать таблицу данных, если можно почти так же легко скопировать формулу? Причин довольно много. Возможно, самая приятная причина заключается в том, что вы можете изменить формулу в таблице данных (ячейка B3 в нашем примере), и остальная часть таблицы изменится. Если вы исследуете содержимое ячеек B4:B7, то увидите, что они содержат значение {=TABLE(A3)}. Это означает, что значение каждой ячейки рассчитывается по формуле вверху таблицы (или по самому левому значению, если используются строки). Нет необходимости копировать формулу в остальные ячейки, если вы хотите изменить формулу.

Другая причина полезности таблиц данных заключается в том, что вы можете включать больше формул в последовательные столбцы таблицы. Например, предположим, что вы также хотите сравнить стоимость Wadget с одинаковыми наценками. Добавьте данные Wadget в первую строку и добавьте формулу Wadget в C3, которая находится рядом с формулой Widget в B3:

* Настройте и отформатируйте строку "Стоимость гаджета".

```
oSheet.Range("E1") = "Wadget Cost"  
oSheet.Range("G1") = 4.23  
oSheet.Range("G1").NumberFormat = "$#,###.##0"
```

* Введите формулу и отформатируйте ячейки (включая результаты
* таблицы данных), чтобы они выглядели как денежная единица.

```
oSheet.Range("C3") = "=G1 * (1 + A3)"  
oSheet.Range("C3:C7").NumberFormat = "$#,###.##0"
```

Теперь вы готовы добавить таблицу данных. Расширьте диапазон, включив в него

столбцы для значений, формулу виджета и формулу виджета.

```
oSheet.Range("A3:C7").Table("", oSheet.Range("A3"))
```

Рисунок 6 показывает результаты. Для копирования формул не требуется дополнительных усилий, и вы все равно можете изменить одну (или обе) формулы и повлиять на соответствующие столбцы в таблице.

	A	B	C	D	E	F	G
1	Widget Cost		\$5.96		Wadget Cost		4.23
2							
3	0.15	\$6.85	\$4.86				
4	0.2	\$7.15	\$5.08				
5	0.25	\$7.45	\$5.29				
6	0.3	\$7.75	\$5.50				
7	0.35	\$8.05	\$5.71				

Рисунок 6. Таблица данных с двумя вычисляемыми столбцами. В столбце B показаны Widget, а в столбце C показаны Wadget, при этом затраты рассчитываются на основе столбца A. В строке 3 можно включить множество формул для включения в таблицу данных, просто включите столбцы в объект Range.

Еще один способ создания таблицы данных – разрешить изменения в двух ячейках формулы. В этом случае значения первой ячейки помещаются в первую строку таблицы (начиная со столбца 2), а значения второй ячейки – в первый столбец таблицы (начиная со строки 2). Первая ячейка в первой строке и столбце содержит формулу.



Листинг 2 (XLData2.PRG в файлах загрузки для разработчиков, доступных по адресу www.hentzenwerke.com) показывает код для настройки таблицы данных, показанной на рисунке 7. В этой таблице данных указана стоимость приобретения до четырех виджетов, а также стоимость доставки (которая является фиксированной стоимостью независимо от количества приобретенных виджетов). Переменная «Количество» размещается в столбце, а переменная «Стоимость доставки» – в строке.

Листинг 2. Настройка таблицы данных с двумя переменными.

```
* XLData2.prg
* © 2000, Tamar E. Granor and Della Martin
* From: Microsoft Office Automation with Visual FoxPro
* Hentzenwerke Publishing. www.hentzenwerke.com

* Удалите все существующие ссылки на серверы.
* Это предотвращает потерю памяти оставшимися экземплярами.
RELEASE ALL LIKE o*

* В демонстрационных целях сделайте oExcel, oBook и oSheet
* доступными после выполнения этой программы.
PUBLIC oExcel, oBook, oSheet

#DEFINE xlEdgeBottom 9
#DEFINE xlEdgeRight 10
#DEFINE xlContinuous 1

oExcel = CREATEOBJECT("Excel.Application")
oExcel.Visible = .T.
oBook = oExcel.Workbooks.Add()
oSheet = oBook.Sheets[1]
```


* Настройте переменные для формулы.

```
oSheet.Range("A1").Value = "Widget Cost"
oSheet.Range("C1").Value = 7.00
oSheet.Range("C1").NumberFormat = "$###.00"
oSheet.Range("C1").Name = "Cost"
```

```
oSheet.Range("A2").Value = "Widget Quantity"
oSheet.Range("C2").Value = 1
oSheet.Range("C2").Name = "Quantity"
```

```
oSheet.Range("A3").Value = "Shipping"
oSheet.Range("C3").Value = 2.00
oSheet.Range("C3").NumberFormat = "$###.00"
oSheet.Range("C3").Name = "Shipping"
```

* Настройте заголовки строк.

```
oSheet.Range("C5").Value = "Shipping Costs"
oSheet.Range("C6").Value = "Ground"
oSheet.Range("D6").Value = "Overnight"
oSheet.Range("E6").Value = "2-Day"
```

* Установите значения строк.

```
oSheet.Range("C7").Value = 2
oSheet.Range("D7").Value = 16
oSheet.Range("E7").Value = 8
```

* Настройте заголовки столбцов.

```
oSheet.Range("A8").Value = "Widget"
oSheet.Range("A9").Value = "Quantity"
```

* Настройте значения столбцов.

```
oSheet.Range("B8").Value = 1
oSheet.Range("B9").Value = 2
oSheet.Range("B10").Value = 3
oSheet.Range("B11").Value = 4
```

* Добавьте формулу.

```
oSheet.Range("B7").Value = "=(Cost * Quantity) + Shipping"
```

* Создать таблицу

```
oSheet.Range("B7:E11").Table(oSheet.Range("Shipping"), oSheet.Range("Quantity"))
```

* Отформатируйте ячейки для валюты.

```
oSheet.Range("C7:E11").NumberFormat = "$###.00"
```

* Укажите границы ячеек для наглядности.

```
oSheet.Range("A7:E7").Borders(xlEdgeBottom).LineStyle = xlContinuous
oSheet.Range("B5:B11").Borders(xlEdgeRight).LineStyle = xlContinuous
```

Попробуйте изменить значения в ячейках формулы: стоимость виджета (C1), стоимость доставки (C7:E7) и количество виджета (B8:B11). Вы также можете изменить формулу в B7. Вместо фиксированной стоимости доставки вы можете изменить ее на стоимость доставки за единицу товара, изменив формулу на «=(Cost * Quantity) + (Quantity * Shipping)». Посмотрите, как изменяются соответствующие части таблицы, без необходимости копировать формулы. Это довольно круто.

	A	B	C	D	E
1	Widget Cost		\$7.00		
2	Widget Quantity		1		
3	Shipping		\$2.00		
4					
5			Shipping Costs		
6			Ground	Overnight	2-Day
7		9.00	\$2.00	\$16.00	\$8.00
8	Widget	1	\$9.00	\$23.00	\$15.00
9	Quantity	2	\$16.00	\$30.00	\$22.00
10		3	\$23.00	\$37.00	\$29.00
11		4	\$30.00	\$44.00	\$36.00

Рисунок 7. Таблица данных с двумя переменными. Формула – « $=(\text{Cost} * \text{Quantity}) + \text{Shipping}$ » (без названий диапазонов – « $=(C1*C2) + C3$ ») содержится в ячейке B7. Диапазон таблицы: B7:E11. Значения в столбце заменяют C2 в формуле, а значения в строке заменяют C3. Если вы измените стоимость виджета (C1), стоимость доставки (C7:E7) или количество (B8:B11), вся таблица пересчитывается, чтобы отразить значения.

8.3 Поиск цели

Поиск цели (GoalSeek) – это функция, которая позволяет вам настроить формулу и указать результат и все значения, кроме одного, в формуле, а затем заставить Excel найти это значение. Хотя наш простой пример можно полностью решить с помощью функций FoxPro, он иллюстрирует, как работает функция поиска цели. Есть много видов задач, для решения которых можно использовать поиск цели; самый популярный пример – функция PMT(), которая принимает три параметра (сумму кредита, срок в месяцах и процентную ставку) и определяет платеж. Большинство из нас, обращаясь за кредитом, знают сумму кредита, желаемый срок и желаемый платеж. Поиск цели позволяет нам предоставить решение функции PMT, а затем найти значение одного из параметров процентной ставки для достижения желаемого платежа.

Нам нужна простая электронная таблица с тремя ячейками данных и еще одной ячейкой для функции оплаты. Предположим, что наша ипотека представляет собой кредит в размере 150 000 долларов США сроком на 30 лет (360 месяцев), и мы хотели бы ежемесячно выплачивать по 975 долларов США. Все, что нам нужно найти, это процентная ставка. Сначала мы настраиваем электронную таблицу с тремя значениями (используйте любую процентную ставку, которую хотите) для настройки платежа, например:

```
oSheet.Range("A1").Value = "Amount"
oSheet.Range("B1").Value = 150000
oSheet.Range("B1").NumberFormat = "$###,###"

oSheet.Range("A2").Value = "Term"
oSheet.Range("B2").Value = 360
oSheet.Range("C2").Value = "months"

oSheet.Range("A3").Value = "Rate"
oSheet.Range("B3").Value = .075
oSheet.Range("B3").NumberFormat = "0.00%"

oSheet.Range("A4").Value = "Payment"
oSheet.Range("B4").Value = "=PMT(B3/12, B2, B1)"
oSheet.Range("B4").NumberFormat = "$###,###"
```

Платеж, показанный в ячейке B4, составляет 1048,82 доллара США (ну, технически это -1048,82; это отрицательное число, потому что вы его должны, поэтому обязательно установите свою цель на отрицательное число). Чтобы довести его до 975 долларов, используйте метод GoalSeek для определения процентной ставки. GoalSeek – это метод объекта Range. Указанный диапазон – это ячейка, содержащая цель, к которой необходимо стремиться. Метод принимает два параметра: первый – это цель или ответ для диапазона, а второй – диапазон (ячейка), который нужно изменить, чтобы решить задачу. Чтобы получить процентную ставку, укажите следующее:

```
oSheet.Range("B4").GoalSeek(-975, oSheet.Range("B3"))
```

И, вуаля, результат у вас: 6,77% (см. рисунок 8). Вы заметили, что цифры замигали и пересчитались? Это потому, что Excel использует итеративный процесс: он меняет значение ячейки переменной, затем проверяет результат. Если результат неверен, он получает новое значение переменной и повторяет процесс. Как правило, этот процесс выполняется очень быстро, хотя сложные уравнения или начальное значение, резко отличающееся от результата, могут его замедлить. Теперь самое сложное – найти банк, который одолжит вам деньги под 6,77%.

	A	B	C
1	Amount	\$150,000	
2	Term	360 months	
3	Rate	6.77%	
4	Payment	(\$975.00)	
5			

Рисунок 8. Результаты метода GoalSeek. Желаемый платеж составляет 975 долларов США, сумма и срок фиксированы; GoalSeek нашел необходимую процентную ставку.

8.4 Защита

Когда вы создаете электронные таблицы для использования другими пользователями, вы не хотите, чтобы они случайно удалили формулы и данные, для которых вы приложили большие усилия, чтобы поместить на лист. В Excel есть несколько способов защитить данные:

- требовать пароли для открытия, редактирования, закрытия и/или сохранения книги;
- рекомендуем открывать книгу как «только для чтения»;
- скрыть формулу ячейки: ячейка остается видимой, но формула скрывается в строке формул;
- заблокируйте книгу, лист или ячейки, что сделает их видимыми, но неизменяемыми.

Чтобы заблокировать или скрыть ячейки, сначала установите для ячеек свойства «Скрытые» и/или «Заблокированные», а затем включите защиту рабочего листа. Если вы хотите защитить все ячейки на листе, не указывайте «Скрытые» или «Заблокированные»; просто установите защиту листа.

Защита клетки

По умолчанию все ячейки заблокированы и видимы. Однако блокировка ячейки имеет значение только в том случае, если лист защищен, а листы не защищены по умолчанию. По сути, вам нужно разблокировать ячейки, которые пользователю необходимо изменить. Объект Range имеет свойство Locked, которое по умолчанию имеет значение .T. Чтобы разблокировать диапазон, введите следующую команду:

```
oSheet.Range("A1:F30").Locked = .F.
```

Скрыть формулу можно с помощью свойства FormulaHidden (результат формулы по-прежнему отображается в ячейке, но сама формула не отображается):

```
oSheet.Range("A1:F30").FormulaHidden = .T.
```

Если рабочий лист уже заблокирован, генерируется ошибка о том, что Excel не может установить свойство Locked (или FormulaHidden). Убедитесь, что вы защищаете ячейки на разблокированном листе. Рабочий лист имеет несколько логических свойств, которые вы можете проверить: ProtectContents и ProtectData – два свойства, которые проверяют наличие данных; другие проверяют защиту на различных объектах.

Защита листа

После того как свойства Locked и FormulaHidden настроены так, как вы хотите, вы можете защитить рабочий лист с помощью метода Protect объекта Worksheet. Требуется несколько параметров. Первый – это строка символов, чувствительная к регистру, которую можно использовать в качестве пароля рабочего листа. Остальные параметры представляют собой логические значения, относящиеся к тому, что необходимо защитить. По порядку это: объекты рисования, содержимое (все ячейки рабочего листа), сценарии (аналитический инструмент) и пользовательский интерфейс (если это правда, макросы не защищаются).

Чтобы установить защиту только ячеек листа, передайте .T. в качестве третьего параметра:

```
oSheet.Protect(, , .T.)
```

Чтобы защитить лист паролем, передайте строку пароля в качестве первого параметра:

```
oSheet.Protect("ThePassword", , .T.)
```

Предупреждение: если вы забудете пароль заблокированного документа, невозможно определить пароль (с помощью Excel или Автоматизации), чтобы разблокировать рабочий лист. Вам необходимо ввести точный пароль с учетом регистра, иначе вы не сможете разблокировать рабочий лист. Используйте пароли на свой страх и риск.

Чтобы разблокировать рабочий лист, передайте .F. в качестве третьего параметра. Если использовался пароль, вы должны указать точную строку с учетом регистра. Если вы не передадите строку пароля, Excel предложит пользователю ввести пароль; убедитесь, что ваше приложение может обработать это состояние ожидания.

Защита книги

Рабочие книги также можно защитить. Структура книги, то есть положение листов в книге, может быть защищена, как и окна книги. Кроме того, книга может иметь собственный пароль. Метод Protect Workbook принимает три параметра: первый – это строка пароля с учетом регистра, второй – логический индикатор того, следует ли защищать структуру книги, а третий – логический индикатор того, следует ли защищать позиции окон книги. Таким образом, любой из этих двух синтаксисов будет работать:

```
oBook.Protect("ThePassword", .T., .T.)
```

```
oBook.Protect(, .T., .T.)
```

Опять же предупреждение: если вы забудете пароль заблокированной книги,

невозможно определить пароль (с помощью Excel или Автоматизации), чтобы разблокировать книгу. Вам необходимо ввести точный пароль с учетом регистра, иначе вы не сможете разблокировать книгу. Используйте пароли на свой страх и риск.

Процедура разблокировки книги такая же, как и для разблокировки листа.

8.5 Шаблоны

Шаблоны Excel позволяют обеспечить стандартное форматирование и стили для документов. Если вы читали эту книгу последовательно, вы уже читали о шаблонах Word в главе 5 «Промежуточный Word» и о том, как они позволяют указывать общие элементы, такие как настройки страницы, верхние и нижние колонтитулы и даже шаблонный текст, чтобы каждый новый документ содержал одно и то же. Excel предлагает то же самое.

Каждая электронная таблица, которую вы создаете в Excel, основана на шаблоне файла с расширением XLT. Если вы не укажете, какой шаблон использовать, будет использован шаблон по умолчанию. По умолчанию это пустой документ, содержащий стандартный набор стилей. Он устанавливается вместе с Excel вместе с рядом других шаблонов. Чтобы просмотреть доступные шаблоны, выберите в меню «Файл | Создать», как и в Word.

Что входит в шаблон?

Шаблон может содержать все, что находится в электронной таблице, от самой простой электронной таблицы до сложного аналитического отчета, включающего множество формул, сводных таблиц и диаграмм, ожидающих добавления нескольких соответствующих элементов данных. Шаблоны также могут содержать макросы, в том числе тот, который запускается автоматически при создании листа на основе шаблона. Как и шаблоны в Word, шаблоны Excel могут быть такими, какими вы их хотите создать.

Поиск шаблонов

Шаблоны хранятся в нескольких разных местах. Шаблоны, установленные Excel 2000, помещаются в подкаталог «Templates» установки Office, а затем перемещаются на один уровень ниже в подкаталог, названный числовым кодом используемого вами языка (1033 для американского английского). Кроме того, каждый пользователь может установить каталог шаблонов пользователя и каталог шаблонов рабочей группы.

Подробная информация о том, как Excel хранит пользовательские шаблоны, содержится в разделе «Поиск шаблонов» главы 5. Хотя это глава Word, это то, что Office стандартизировал.

Единственное, что немного отличается, это то, как найти путь к шаблонам. В объекте Application доступны два свойства: `TemplatesPath` и `NetworkTemplatesPath`. `TemplatesPath` — это путь к шаблонам вашего локального компьютера, а `NetworkTemplatesPath` позволяет указать на другой набор шаблонов в сети.

Использование шаблонов

При Автоматизации Excel можно использовать шаблоны несколькими способами. Самый простой — создавать новые документы на основе существующих шаблонов. Для этого укажите шаблон, включая путь, в качестве первого параметра метода `Workbooks.Add`. При установке Office/Excel по умолчанию шаблоны не устанавливаются, поэтому не ждите, что это имя шаблона будет работать:

```
oBook = oExcel.Workbooks.Add( oExcel.TemplatesPath + ;  
                                "MyTemplate.XLT")
```

Создав новую книгу на основе шаблона, вы можете обращаться с ней так же, как с любой другой новой книгой. Однако у вас есть то преимущество, что она содержит весь специальный текст, форматирование и стили, хранящиеся в шаблоне.

Создание шаблонов

Вы также можете создавать шаблоны с помощью Автоматизации. Любую книгу можно сохранить как шаблон, передав соответствующий параметр методу SaveAs. Чтобы создать новый шаблон, создайте книгу, отформатируйте ее по желанию, создайте любые стили, диаграммы, сводные таблицы или другие функции, которые вы хотите, чтобы шаблон их имел, а затем вызовите SaveAs следующим образом:

```
#DEFINE xlTemplate 17  
oBook.SaveAs(oExcel.TemplatesPath + "MyNewTemplate.XLT", ;  
             xlTemplate)
```

Как и в интерактивном Excel, вы можете сохранить шаблон в подкаталоге, чтобы он отображался на другой странице в диалоговом окне «Файл | Создать». Конечно, если вы работаете с ним через Автоматизацию, вам все равно, где он появится. Фактически, при использовании Автоматизации не имеет значения, где вы храните шаблоны, поскольку вы можете указать, где Excel должен их искать. Однако сохранение их вместе с другими шаблонами означает, что интерактивные пользователи также смогут их найти.

8.6 Автоматизация макросов

Хотя с помощью Автоматизации вы можете выполнять практически любые действия Excel, если у вас уже есть макрос Excel для выполнения определенной задачи, возможно, имеет смысл использовать существующий макрос, чем переписывать его в VFP. Это особенно актуально, если макрос не предполагает передачу информации между двумя приложениями или если это часть предварительно упакованного набора макросов. Преимущество запуска макросов Excel вместо их перевода в код Автоматизации заключается в том, что макросы выполняются быстрее, чем аналогичный код Автоматизации.

Метод Run объекта приложения позволяет выполнять макросы Excel. Вы передаете имя макроса и любые параметры, необходимые для запуска макроса. Например, скажем, Macro1 – это имя по умолчанию для записанного макроса. Он может выполнять различные задачи. Эта строка выполняет это:

```
oExcel.Run("Macro1")
```

Во многих случаях не составит труда создать ту же функциональность с помощью Автоматизации. Однако бывают ситуации, когда перезапись существующего макроса в качестве Автоматизации может создать проблему. В таких случаях Run предоставляет простое решение.

Другая распространенная ситуация – написание макроса с использованием кода Автоматизации. Например, вы можете написать макрос для сохранения книги Excel. Написание макроса Excel для сохранения книги позволяет вам воспользоваться свойством DisplayAlerts приложения Excel, которое принимает логическое значение и работает аналогично команде SET SAFETY в FoxPro. DisplayAlerts работает только в среде макроса и сбрасывается в значение True в конце макроса. Использование следующего макроса позволяет сохранить файл без использования всех обходных путей, описанных в главе 7 «Основы Excel».

Мы предполагаем, что вы умеете писать макросы Excel. Определите необходимый вам код и поместите его в файл. В этом примере для создания файла используются файловые функции низкого уровня, хотя в FoxPro существует множество других способов выполнить эту задачу. (В VFP 6 функция StrToFile() обеспечивает более удобный способ создания файлов.)

* Откройте файл и вставьте строки кода Excel.

```
m.MacroFileName = "C:\Temp\SaveMacro.Txt"
m.MFHandle = FCREATE(m.MacroFileName)
```

* Добавьте тело макроса.

```
= FPUTS(m.MFHandle, "' QuietSave Macro")
= FPUTS(m.MFHandle, "' ")
= FPUTS(m.MFHandle, "Sub QuietSave()")
= FPUTS(m.MFHandle, "' ")
= FPUTS(m.MFHandle, "  Application.DisplayAlerts = False")
= FPUTS(m.MFHandle, "  ActiveWorkbook.Save")
= FPUTS(m.MFHandle, "End Sub")
```

* Закройте открытый файл.

```
=FCLOSE(m.MFHandle)
```

После того как код макроса записан в файл, вы готовы перенести его в Excel. Используйте метод Add объекта коллекции Modules, чтобы добавить пустой макромодуль. Затем используйте метод InsertFile нового объекта Module, чтобы импортировать код макроса. Назовите объект Macro, и вы готовы его запустить:

* Добавьте новый модуль в коллекцию модулей.

```
oMacroModule = oExcelObject.Modules.Add()
```

* Загрузите файл, содержащий код макроса.

```
oMacroModule.InsertFile(m.MacroFileName)
```

* Назовите макрос

```
oMacroModule.Name = "MacroSave"
```

* Запустите макрос

```
oExcel.Application.Run("QuietSave")
```

Этот пример не обязательно является одобрением использования макроса QuietSave вместо обходных путей, описанных в главе 7 «Основы Excel». Это альтернативный метод, который может работать или не работать в среде вашего приложения. Но он иллюстрирует, как создавать макросы с помощью Автоматизации.

8.7 Добавление фигур

Подобно Word и PowerPoint, Excel может добавлять графику на лист. Excel (ну, на самом деле Office) имеет надежный набор объектов Shape, которые доступны любому приложению Office. В главе 5 «Промежуточный Word» рассказывается, как Word работает с объектом Shape, а в главе 10 «Основы PowerPoint» очень подробно рассказывается об объекте Shape для PowerPoint. Здесь мы рассмотрим основы добавления объекта Shape и отправим вас в главу PowerPoint для получения подробной информации.

Доступные фигуры, называемые автофигурами, предоставляют гораздо больше, чем просто базовые фигуры, такие как прямоугольники и круги. Существует

множество декоративных форм, таких как стрелки, звезды и баннеры. Также доступны символы блок-схемы и многие виды выносок. Любую из этих фигур можно добавить на лист с помощью метода AddShape коллекции Shapes. Автофигуры используются для создания диаграмм и иллюстраций или просто для украшения вашего рабочего листа.

Чтобы в интерактивном режиме добавить фигуру на рабочий лист, выберите в меню Вставить|Рисунок|Автофигуры, чтобы активировать панель инструментов «Автофигуры», показанную на рисунке 9. Кнопки представляют основные категории фигур, и нажатие на каждую из них открывает другое меню кнопок, в котором отображаются доступные фигуры для этой основной категории фигур.



Рисунок 9. Панель инструментов «Автофигуры». Эта панель инструментов появляется всякий раз, когда вы выбираете в меню Вставить|Рисунок|Автофигуры.

Чтобы добавить фигуру программным способом, используйте метод AddShape коллекции Shapes. Это синтаксис для добавления автофигуры:

```
oShape = oSheet.Shapes.AddShape(nType, nLeft, nTop, nWidth, ;
                                nHeight)
```

Первый параметр – это числовая константа, указывающая одну из 140 доступных фигур. В таблице 2 перечислены некоторые доступные константы. Обратите внимание, что их префикс – «mso», что означает, что они доступны для всех приложений Office, а не только для PowerPoint. Следующие два параметра определяют верхний левый угол прямоугольного поля, содержащего фигуру (конечно, в пунктах). Последние два параметра определяют ширину и высоту объекта в пунктах. Эта прямоугольная рамка, содержащая фигуру, называется *ограничивающей рамкой*.

Следующие строки кода добавляют стрелку в текущую электронную таблицу. Она расположена на полдюйма сверху и слева, имеет длину один дюйм и высоту полдюйма. Результаты см. на рисунке 10.

```
#DEFINE msoShapeRightArrow 33
oSheet.Shapes.AddShape(msoShapeRightArrow, 36, 36, 72, 36)
```

Таблица 2. Выборка констант AutoShape и их значений.

Константы Shape	Значение
msoShape5PointStar	92
msoShapeArc	25
msoShapeBalloon	137
msoShapeCube	14
msoShapeDownArrow	36
msoShapeLeftArrow	34
msoShapeLineCallout1	109
msoShapeNoSymbol	19
msoShapeOval	9

msoShapeParallelogram	2
msoShapeRectangle	1
msoShapeRightArrow	33
msoShapeRoundedRectangle	5
msoShapeUpArrow	35

Каждый объект Shape имеет ряд свойств, некоторые из которых содержат объекты форматирования. В таблице 3 показаны наиболее важные свойства формы.

Таблица 3. Свойства и объекты Shape.

Свойство	Тип	Описание
AutoShapeType	Numeric	Одна из констант AutoShape, некоторые из которых перечислены в таблице 2.
Height	Numeric	Высота фигуры в пикселях.
Width	Numeric	Ширина фигуры в пикселях
Left	Numeric	Положение левой части фигуры от левой стороны листа в пикселях.
Top	Numeric	Положение верхней части фигуры относительно верхнего края листа в пикселях.
TextFrame	Object	Ссылка на объект TextFrame. Среди его свойств и методов есть свойство «Character», которое указывает на объект «Character». Свойство Text объекта символов содержит текст в рамке, а его свойство Font ссылается на объект Font для управления внешним видом текста.
HorizontalFlip	Logical	Истинно, если фигура перевернута по горизонтали.
VerticalFlip	Logical	Истинно, если фигура перевернута вертикально.
Fill	Object	Ссылка на объект FillFormat.
Line	Object	Ссылка на объект LineFormat.

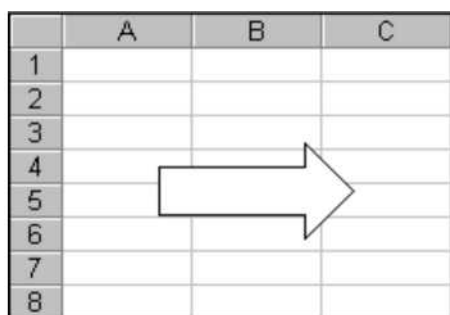


Рисунок 10. Добавление автофигуры. Метод AddShape коллекции фигур добавляет на лист одну из более чем 140 автофигур Office.

Объект TextFrame содержит свойства для форматирования текста (если таковой имеется) в фигуре. Самым важным свойством является свойство Character, которое ссылается на объект Character. Свойство Text объекта Character содержит текст, а

его свойство Font обращается к объекту Font для управления внешним видом текста. Следующий код добавляет текстовую строку и форматирует шрифт:

```
WITH oSheet.Shapes[1].TextFrame.Characters
    .Text = "HERE"
    WITH .Font
        .Color = RGB(255, 0, 0) &&красный
        .Name = "Times New Roman"
        .Size = "12"
    ENDWITH
ENDWITH
```

Вот интересная ситуация: при форматировании границ ячеек, объектов «Диаграмма» (Chart) и других элементов в Excel объект «Граница» (Border) управляет линиями границ, а объект «Внутренний» (Interior) управляет свойствами заливки. Однако фигуры являются объектами Office и вместо них используют объекты LineFormat и FillFormat. К счастью, они похожи. К сожалению, они не одинаковы.

Объекты Line хранят свойства цвета, стиля, прозрачности, узора, веса и стрелок в объекте LineFormat. Используйте свойство Line объекта Shape для доступа к объекту LineFormat. В таблице 4 показаны многие доступные свойства объекта LineFormat.

Цвета линии задаются с помощью свойств ForeColor и BackColor. Эти свойства указывают на объект ColorFormat. Объект ColorFormat имеет только два свойства: одно, необходимое здесь, – это свойство RGB (второе – свойство SchemeColor; оно будет рассмотрено позже). См. «Форматирование границы фигуры» в главе 10, где приведены примеры использования этих свойств.

Объект FillFormat имеет множество свойств и довольно сложен. Чтобы залить фигуру цветом, используйте объект ForeColor FillFormat. (Напоминание: на объект FillFormat ссылается свойство Fill объекта Shape.)

```
oSheet.Shapes[1].Fill.ForeColor.RGB = RGB(192,192,192) && Светло-серый
```

Дополнительную информацию об объекте FillFormat см. в разделе «Предпосылки» главы 11, где объект FillFormat обсуждается подробно.

Таблица 4. Свойства объекта LineFormat.

Свойство	Тип	Описание
ForeColor	Object	Цвет линии с использованием объекта ColorFormat.
BackColor	Object	Цвет фона узорной линии с использованием объекта ColorFormat. Это вторичный цвет линии с узором, который игнорируется, если узоры не используются.
DashStyle	Numeric	Стиль штриха линии. Использует одну из следующих констант: msoLineDash 4 msoLineDashDot 5 msoLineDashDotDot 6 msoLineLongDash 7 msoLineLongDashDot 8 msoLineRoundDot 3 msoLineSolid 1 msoLineSquareDot 2
Pattern	Numeric	Узор, примененный к линии. В качестве фона рисунка

		используется цветной фон. Используйте одну из многих шаблонных констант, некоторые из которых перечислены здесь:																												
		<table><tr><td>msoPattern50Percent</td><td>7</td><td>msoPatternLargeConfetti</td><td>33</td></tr><tr><td>msoPatternLargeGrid</td><td>34</td><td>msoPatternLightDownwardDiagonal</td><td>21</td></tr><tr><td>msoPatternLightHorizontal</td><td>19</td><td>msoPatternLightUpwardDiagonal</td><td>22</td></tr><tr><td>msoPatternLightVertical</td><td>20</td><td>msoPatternPlaid</td><td>42</td></tr><tr><td>msoPatternSmallGrid</td><td>23</td><td>msoPatternSolidDiamond</td><td>39</td></tr><tr><td>msoPatternWideDownwardDiagonal</td><td>25</td><td>msoPatternWideUpwardDiagonal</td><td>26</td></tr><tr><td>msoPatternZigZag</td><td>38</td><td></td><td></td></tr></table>	msoPattern50Percent	7	msoPatternLargeConfetti	33	msoPatternLargeGrid	34	msoPatternLightDownwardDiagonal	21	msoPatternLightHorizontal	19	msoPatternLightUpwardDiagonal	22	msoPatternLightVertical	20	msoPatternPlaid	42	msoPatternSmallGrid	23	msoPatternSolidDiamond	39	msoPatternWideDownwardDiagonal	25	msoPatternWideUpwardDiagonal	26	msoPatternZigZag	38		
msoPattern50Percent	7	msoPatternLargeConfetti	33																											
msoPatternLargeGrid	34	msoPatternLightDownwardDiagonal	21																											
msoPatternLightHorizontal	19	msoPatternLightUpwardDiagonal	22																											
msoPatternLightVertical	20	msoPatternPlaid	42																											
msoPatternSmallGrid	23	msoPatternSolidDiamond	39																											
msoPatternWideDownwardDiagonal	25	msoPatternWideUpwardDiagonal	26																											
msoPatternZigZag	38																													
Style	Numeric	Стиль линии, который может выглядеть как несколько линий.																												
		<table><tr><td>msoLineSingle</td><td>1</td><td>msoLineThickBetweenThin</td><td>5</td></tr><tr><td>msoLineThickThin</td><td>4</td><td>msoLineThinThick</td><td>3</td></tr><tr><td>msoLineThinThin</td><td>2</td><td></td><td></td></tr></table>	msoLineSingle	1	msoLineThickBetweenThin	5	msoLineThickThin	4	msoLineThinThick	3	msoLineThinThin	2																		
msoLineSingle	1	msoLineThickBetweenThin	5																											
msoLineThickThin	4	msoLineThinThick	3																											
msoLineThinThin	2																													
Transparency	Numeric	Степень прозрачности линии. Значение варьируется от 0,0 (непрозрачный) до 1,0 (полностью прозрачный).																												
Weight	Numeric	Толщина линии в пунктах.																												
Visible	Logical	Указывает, видна ли линия.																												

8.8 Добавление изображений

Изображения добавляются в коллекцию Shapes с помощью метода AddPicture. Синтаксис следующий:

```
oSheet.Shapes.AddPicture(cFileName, lLinkToFile,
                          lSaveWithDocument, nLeft, nTop, nWidth, nHeight)
```

cFileName	Character	Полное имя файла изображения.
lLinkToFile	Logical	Указывает, следует ли связать изображение с файлом, из которого оно было создано, или создать копию внутри документа.
lSaveWithDocument	Logical	Указывает, следует ли сохранить связанное изображение вместе с файлом или сохранить в документе только информацию о ссылке. Этот аргумент должен иметь значение True, если lLinkToFile имеет значение False.
nLeft, nTop	Numeric	Положение (в пунктах) верхнего левого угла изображения, измеренное от верхнего левого угла электронной таблицы.
nWidth, nHeight	Numeric	Ширина и высота изображения в пунктах.

В следующем примере на лист добавляется логотип TasTrade. Обязательно укажите полный путь, поскольку Excel ничего не знает о настройках VFP по умолчанию.

```
LogoFile = _SAMPLES + "TasTrade\Bitmaps\TTradeSm.bmp"
oSheet.Shapes.AddPicture(LogoFile, .T., .T., 32, 32, 72, 72)
```

Когда вы пытаетесь определить ширину и высоту изображения, потребуется немного поиграть. Ни один из параметров не является необязательным, поэтому вам необходимо знать ширину и высоту любого добавляемого изображения.

8.9 Собираем все вместе



В листинге 3 показан пример программы для отдела закупок Tasmanian Traders. Они хотели бы продавать малиновый сироп Mama's Megacorp, но Mama's Megacorp хочет получить «маркетинговый гонорар» в размере 500 долларов, прежде чем тасманские торговцы смогут продавать продукт Mama. Поэтому отдел закупок попросил проанализировать ситуацию. Они хотят знать, сколько единиц малинового сиропа Mama's им нужно продать, прежде чем они достигнут уровня безубыточности, и они хотят увидеть разбивку продаж всех приправ по годам, чтобы они могли выяснить, является ли количество, необходимое для достижения безубыточности, в пределах обычных продаж всех приправ. Это должно выглядеть хорошо, потому что вице-президент по закупкам Tasmanian Traders возьмет этот отчет с собой, когда встретится с представителями Mama's Megacorp для завершения сделки.

Вот решение. Используйте сводную таблицу, чтобы предоставить разбивку продаж, а затем используйте метод GoalSeek, чтобы найти показатель безубыточности. После того, как количество будет рассчитано, пометьте его яркой звездочкой, чтобы привлечь внимание к необходимому количеству. Затем добавьте логотип TasTrade. На рисунке 11 показан отчет, который не только будет удовлетворять требованиям к необходимой информации, но и будет иметь визуальную привлекательность.

Листинг 3 доступен как XLSample2.PRG в файлах для загрузки для разработчиков, доступных на сайте www.hentzenwerke.com.

Листинг 3. Аналитическая таблица тасманских торговцев для нового продукта. В этом примере показано создание сводной таблицы с использованием поиска цели и добавление изображений и фигур. На рисунке 11 показаны результаты.

```
* XLSample2.prg
* © 2000, Tamar E. Granor and Della Martin
* From: Microsoft Office Automation with Visual FoxPro
* Hentzenwerke Publishing. www.hentzenwerke.com
```

```
* Это пример «Собираем все вместе» из главы 8.
* Удалите все существующие ссылки на серверы.
* Это предотвращает потерю памяти оставшимися экземплярами.
```

```
RELEASE ALL LIKE o*
```

```
* В демонстрационных целях сделайте определенные объекты
* доступными после выполнения этой программы.
```

```
PUBLIC oExcel, oBook, oSheet2, oPivotTable
```

```
#DEFINE xlDatabase
```

```
1
```

```
#DEFINE xlDataField      4
#DEFINE xlEdgeBottom     9
#DEFINE xlContinuous     1
#DEFINE msoShape24PointStar 95
#DEFINE autoIn2Pts       72
#DEFINE xlHAlignCenter   -4108
#DEFINE xlVAlignCenter   -4108
```

```
CLOSE DATA
```

```
OPEN DATABASE (_SAMPLES + "\TasTrade\Data\TasTrade")
```

```
SET ENGINEBEHAVIOR 70
```

```
SELECT Customer.Company_Name, ;
       Customer.Country, ;
       CMONTH(Orders.Order_Date) AS Order_Month, ;
       YEAR(Orders.Order_Date) AS Order_Year, ;
       Products.Product_Name, ;
       Order_Line_Items.Quantity, ;
       Order_Line_Items.Unit_Price * Order_Line_Items.Quantity ;
       AS Total_Price;
FROM Orders, Order_Line_Items, Customer, Products ;
WHERE Order_Line_Items.Order_ID = Orders.Order_ID ;
      AND Orders.Customer_ID = Customer.Customer_ID ;
      AND Order_Line_Items.Product_ID = Products.Product_ID ;
      AND Products.Category_ID = "      2" ;
INTO CURSOR Pivot
```

```
COPY TO (CURDIR() + "Pivot") TYPE XL5
```

```
oBook = GETOBJECT(CURDIR() + "Pivot.XLS")
```

```
* Откройте книгу и найдите наилучшее расположение всех столбцов.
* Убедитесь, что есть два листа:
* сводный для данных и еще один для "презентации".
```

```
WITH oBook
  oExcel = .Application
  .Application.Visible = .T.
  .Windows[1].Activate()
  .Sheets[1].Range("A1:G310").Columns.AutoFit()
  oSheet2 = .Sheets.Add()
ENDWITH
```

```
* На листе презентации добавьте логотип и заголовок.
```

```
WITH oSheet2
```

```
  * Добавьте логотип TasTrade.
```

```
  LogoFile = _SAMPLES + "\TasTrade\Bitmaps\TTradeSm.bmp"
  .Shapes.AddPicture(LogoFile, .T., .T., ;
    0.25 * autoIn2Pts, 0.25 * autoIn2Pts, ;
    1.00 * autoIn2Pts, 1.00 * autoIn2Pts)
```

```
  * Назовите рабочий лист.
```

```

WITH .Range["C2"]
    .Value = "Analysis for Mama's Raspberry Syrup"
    .Font.Size = 18
    .Font.Bold = .T.
ENDWITH
ENDWITH

* Создайте новую сводную таблицу.
oSourceData = oBook.Sheets[2].Range["A1:G" + ;
    ALLTRIM(STR(RECCOUNT("Pivot") + 1))]
oDestination = oSheet2.Range["C4"]
oPivotTable = oBook.Sheets[1].PivotTableWizard(xlDatabase, ;
    oSourceData, oDestination, "Analysis", .T., .T.)

WITH oPivotTable
    .AddFields("Product_Name", "Order_Year")
    .PivotFields["Quantity"].Orientation = xlDataField

    * Сделайте заголовки красивыми
    .PivotFields["Product_Name"].Name = "Product Name"
    .PivotFields["Order_Year"].Name = "Year"
    .PivotFields["Quantity"].Name = "Quantity Sold"

ENDWITH

WITH oSheet2

    * Отформатируйте столбцы немного шире.
    .Columns[4].ColumnWidth = 9
    .Columns[5].ColumnWidth = 9
    .Columns[6].ColumnWidth = 9
    .Columns[7].ColumnWidth = 9
    .Columns[8].ColumnWidth = 9

    * Найдите точку безубыточности с помощью Goal Seek
    * Заголовков
    .Range["C21"].Value = "Анализ безубыточности"
    .Range["C21"].Font.Bold = .T.

    * Доход
    .Range["C22"].Value = "Цена за единицу"
    .Range["D22"].Name = "UnitPrice"
    .Range["D22"].Value = 10
    .Range["D22"].NumberFormat = "$###.00"

    .Range["C23"].Value = "Проданных единиц"
    .Range["D23"].Name = "UnitsSold"
    .Range["D23"].Value = 100
    .Range["D23"].NumberFormat = "###.0"

    .Range("C23:D23").Borders(xlEdgeBottom).LineStyle= xlContinuous

    .Range["C24"].Value = "Общий доход"

```



```

.Range["D24"].Name = "TotalRevenue"
.Range["D24"].Value = "= UnitPrice * UnitsSold"
.Range["D24"].NumberFormat = "$##,###.00"

* Расходы
.Range["C26"].Value = "Стоимость за единицу"
.Range["D26"].Name = "UnitCost"
.Range["D26"].Value = 7.23

.Range["C27"].Value = "Маркетинговый сбор"
.Range["D27"].Name = "MarketingFee"
.Range["D27"].Value = 500

.Range("C27:D27").Borders(xlEdgeBottom).LineStyle= xlContinuous

.Range["C28"].Value = "Суммарные затраты"
.Range["D28"].Name = "TotalCost"
.Range["D28"].Value = "= (UnitCost * UnitsSold) + MarketingFee"
.Range["D26:D30"].NumberFormat = "$##,###.00"

* Выгода
.Range["C30"].Value = "Выгода"
.Range["D30"].Name = "Profit"
.Range["D30"].Value = "= TotalRevenue - TotalCost"

* Выполните поиск цели
.Range["Profit"].GoalSeek(0, .Range("UnitsSold"))

* Добавьте фигуру с текстом, указывающим, сколько нужно
* продать, чтобы достичь безубыточности.
oShape = .Shapes.AddShape(msoShape24PointStar, ;
    4.8 * autoIn2Pts, 3.5 * autoIn2Pts, ;
    2.0 * autoIn2Pts, 2.0 * autoIn2Pts)
cQuantity = ALLTRIM(STR(.Range["UnitsSold"].Value))

WITH oShape.TextFrame
    WITH .Characters
        .Text = "Продайте " + cQuantity + ;
            " чтобы выйти на уровень безубыточности"
        .Font.Color = RGB(255, 0, 0) && красный
        .Font.Size = 12
        .Font.Bold = .T.
    ENDWITH
    .HorizontalAlignment = xlHAlignCenter
    .VerticalAlignment = xlVAlignCenter
ENDWITH

ENDWITH

```

	A	B	C	D	E	F	G	H
1								
2			Analysis for Mama's Raspberry Syrup					
3								
4			Quantity Sold	Year				
5			Product Name	1992	1993	1994	1995	Grand Total
6			Aniseed Syrup	48	120	219	34	421
7			Camembert Pierrot	303	543	761	366	1993
8			Chef Anton's Gumbo Mix	38	169	49	120	376
9			Genen Shouyu	10	78	97		185
10			Grandma's Boysenberry Spread	23	36	156	109	324
11			Gula Malacca	50	260	372	51	733
12			Louisiana Fiery Hot Pepper Sauce	90	195	506	64	855
13			Louisiana Hot Spiced Okra	24	30	208	1	263
14			Northwoods Cranberry Sauce	92	152	134	98	476
15			Original Frankfurter grüne Soße	62	235	439	244	980
16			Sirup d'érable	55	38	410	177	680
17			Veggie-spread	34	229	89	147	499
18			Grand Total	829	2085	3440	1431	7785
19								
20								
21			Break-Even Analysis					
22			Price per Unit	\$10.00				
23			Units Sold	180.5				
24			Total Revenue	\$1,805.05				
25								
26			Cost per Unit	\$7.23				
27			Marketing Fee	\$500.00				
28			Total Costs	\$1,805.05				
29								
30			Profit	\$.00				
31								



Рисунок 11. Аналитическая таблица TasTrade для нового продукта. Функция «Поиск цели» обеспечивает анализ безубыточности, результаты которого выделены в фигуре. Требуемое количество единиц можно сравнить со сводной таблицей, содержащей историю продаж всех приправ. И растровое изображение делает его красивым.

В этой главе рассматриваются некоторые основные инструменты анализа, доступные в Excel. Мы коснулись наиболее часто используемых инструментов анализа, чтобы дать вам представление о методах работы в Excel с помощью Автоматизации. Существуют дополнительные инструменты, которые требуют аналогичных методов. В следующей главе рассматривается еще один вид инструмента анализа: диаграммы. Excel имеет мощный механизм построения диаграмм, позволяющий отображать результаты анализа данных, выполненного в Excel.

Глава 9. Графический механизм Excel

Поскольку в FoxPro отсутствуют встроенные возможности построения графиков, Excel может вмешаться и предоставить потрясающий механизм построения графиков.

Об использовании Microsoft Graph в приложениях VFP написано много. Графический движок Excel основан на Microsoft Graph, но у него есть одно преимущество, которого нет в Graph: документация. (На самом деле почти любой графический код Excel также работает в Graph, но это один из тех недокументированных секретов, на которые нельзя рассчитывать.) То, что вы не можете найти в печатной или онлайн-документации и справке, обычно можно узнать путем записи макроса.

9.1 Объектная модель

В каждой книге есть коллекция Charts, содержащая все диаграммы активной книги. Объект Chart содержит свойства, которые управляют форматом и данными в диаграмме. Объект SeriesCollection хранит несколько объектов Series, по одному для каждой серии данных (один набор данных для построения графика) на диаграмме. Каждый объект Series имеет свойство Values, в котором хранится диапазон, содержащий значения для построения графика. Объект Series также имеет множество свойств для управления внешним видом рядов на графике, например тип диаграммы (одна серия может быть гистограммой, а следующая — линейной диаграммой), цвет, границы и т. д.

Объект Chart имеет свойства для доступа к объектам, которые являются глобальными для диаграммы. Например, существуют объекты для форматирования нескольких объектов Axis («Оси»), а также Floor («Пол») и Walls «Стены») PlotArea. ChartArea (область вокруг диаграммы) форматируется отдельно от PlotArea (внутренняя часть диаграммы, ограниченная снаружи осями). Наконец, есть свойства для доступа к объектам Legend («Легенда») и ChartTitle («Заголовок диаграммы»).

Это диаграмма или лист?

Excel предлагает два способа размещения диаграмм в книге. Вы можете сохранить диаграмму как объект на листе или сохранить ее как отдельный лист (также известный как *лист диаграммы*). Хотя оба типа хранятся как объекты Chart, они хранятся в разных коллекциях, и каждая коллекция принадлежит отдельному объекту.

Доступ к листам диаграмм осуществляется через коллекцию Charts, а поскольку они тоже являются рабочими листами, к ним также можно получить доступ через коллекцию Sheets (мы рекомендуем всегда использовать коллекцию Charts, поскольку изменение форматирования диаграммы на том, что выглядит как лист, довольно запутанно). Коллекция Charts доступна как через объект Application, так и через объект Workbook.

Встроенные диаграммы имеют собственную коллекцию ChartObjects. Эта коллекция принадлежит объекту Worksheet.

Важно понимать, что листы диаграмм принадлежат рабочей книге, а встроенные диаграммы — рабочему листу. Однако оба типа диаграмм хранятся как один и тот же объект Chart и имеют одни и те же свойства и методы, доступные для форматирования диаграммы.

Независимо от того, является ли это внедренной диаграммой или листом диаграммы, если он выбран (либо лист диаграммы является ActiveSheet, либо

внедренная диаграмма имеет маркеры «изменения размера»), свойство ActiveChart объекта Workbook ссылается на диаграмму.

9.2 Настройка данных

Чтобы построить график данных, они должны находиться на листе Excel (или в серии листов). В разделе «Обработка значений и формул» главы 7 обсуждается механизм добавления данных на листы. Мы не будем повторять это здесь, но расскажем вам, как настроить рабочий лист, чтобы ваши данные можно было легко преобразовать в график.

Во-первых, нам нужно изучить, как Excel работает с данными. Диаграммы Excel работают с последовательностями данных, называемыми *рядами данных*. Каждый ряд данных представляет собой последовательность точек, нанесенных на график с одинаковыми атрибутами (например, цветом, узором, символом, стилем линии, стилем столбца, что соответствует выбранному стилю диаграммы). На линейном графике на одном графике можно нанести множество линий; каждая линия представляет собой ряд данных. На гистограмме с тремя группами столбцов с цветовой кодировкой каждый цвет представляет собой ряд данных. Круговая диаграмма может отображать только один ряд данных.

Excel также использует диапазоны для обозначения оси категорий и для обозначения рядов данных в легенде. Вы можете указать каждый ряд данных отдельно, что полезно, когда один лист данных используется для создания множества графиков. Однако Excel может принять один прямоугольный диапазон и использовать его для построения диаграммы. Если вы знакомы с мастером диаграмм Excel, вы, вероятно, уже знаете, что по умолчанию он использует первую строку и столбец для заголовков, а затем использует оставшиеся столбцы (или строки) для рядов данных. Понимание того, как Excel использует один диапазон для диаграммы, значительно облегчит вам жизнь; передать один диапазон гораздо проще, чем явно задать полдюжины диапазонов.

Настроить этот единый диапазон легко. Используйте столбец A для меток категорий. Содержимое каждой ячейки – это фактические метки, используемые на оси X, поэтому убедитесь, что они не слишком загадочны и не слишком длинные. Используйте последующие столбцы для рядов данных. Поместите метку каждого ряда в первую строку; данные начинаются со строки 2. Заголовки категорий в столбце A также должны начинаться со строки 2. На рисунке 1 показан образец электронной таблицы, готовый к построению графика. Код для создания этой таблицы показан в листинге 1 (XLGDataSetup.PRG в файлах для загрузки разработчика, доступных по адресу www.hentzenwerke.com).

	A	B	C	D	E	F
1		1992	1993	1994	1995	1996
2	January		\$1,038.10	\$1,632.30	\$3,530.98	
3	February		\$850.40	\$1,506.90	\$4,723.60	
4	March		\$1,383.90	\$2,581.42	\$5,381.84	
5	April		\$1,424.70	\$2,667.85		
6	May	\$730.10	\$929.00	\$1,693.97		\$19.00
7	June	\$782.20	\$1,361.10	\$2,042.98		
8	July	\$908.00	\$1,485.20	\$2,462.44		
9	August	\$736.30	\$1,254.70	\$2,860.28		
10	September	\$907.10	\$1,633.80	\$2,497.63		
11	October	\$1,808.40	\$2,223.60	\$2,210.94		
12	November	\$919.90	\$1,673.20	\$4,123.10		
13	December	\$865.30	\$1,994.80	\$4,848.48		

Рисунок 1. Электронная таблица, отформатированная для графика в виде единого диапазона. Значения в столбце A становятся метками оси X; текст в строке 1 становится текстом, обозначающим каждую серию в легенде для серии данных, хранящейся в столбце.

Листинг 1. Код для заполнения графических данных, показанных на рисунке 1. Этот код использует ежемесячные данные о продажах из образца базы данных TasTrade.

```
* XLGDataSetup.PRG
* © 2000, Tamar E. Granor and Della Martin
* From: Microsoft Office Automation with Visual FoxPro
* Hentzenwerke Publishing. www.hentzenwerke.com

* Устанавливает ежемесячные данные о продажах по годам
* для отображения на графике.

* Удалите все существующие ссылки на серверы.
* Это предотвращает потерю памяти оставшимися экземплярами.
RELEASE ALL LIKE o*

* В демонстрационных целях сделайте oExcel и oBook
* доступными после выполнения этой программы.
PUBLIC oExcel, oBook

#DEFINE xlColumns 2
#DEFINE xlAutoFill 4

* Откройте представление «Сводка продаж», которое содержит
* сводку цен за единицу продукции за каждый месяц и год.
CLOSE ALL
OPEN DATABASE (_SAMPLES + "\TASTRADE\DATA\Tastrade")
USE Customer IN 0
SELECT 0
USE "Sales Summary" ALIAS SalesSummary

* Добавьте книгу, используя настройки по умолчанию.
oExcel = CREATEOBJECT("Excel.Application")
oExcel.Visible = .T.
oBook = oExcel.Workbooks.Add()

WITH oBook.Sheets[1]

    * Впишите месяцы в столбец A.
    * Это метки по оси категорий (x).
    .Range("A2").Value = "Январь" && "January"
    .Range("A2:A13").DataSeries(xlColumns, xlAutoFill, 1, 1)

    * Прокрутите представление. Создайте для каждого года
    * столбец и убедитесь, что есть заголовки.
    CurrentColumn = "A"
    CurrentYear = "X"
    SCAN
        IF CurrentYear <> LEFT(SalesSummary.Exp_1, 4)

            * Сохраните текущий год.
            CurrentYear = LEFT(SalesSummary.Exp_1, 4)

            * Увеличьте букву столбца
            * (примечание: при такой логике не превышайте 25 лет!)
```

```

CurrentColumn = CHR(ASC(CurrentColumn) + 1)

* Установите заголовок.
.Range(CurrentColumn + "1").Value = CurrentYear
ENDIF

* Вычислить текущую строку (прибавить 1 к значению месяца);
* убедитесь, что это строка
CurrentRow = ALLTRIM(STR(VAL(RIGHT(SalesSummary.Exp_1, 2)) ;
+ 1))

* Установите значение ячейки.
.Range(CurrentColumn + CurrentRow).Value = ;
SalesSummary.Sum_Unit_Price

ENDSCAN

* Расширьте столбцы, чтобы были видны все значения.
* Преобразование CurrentColumn из буквенного символа в число.
FOR I = 1 TO ASC(CurrentColumn) - 63
.Columns[I].ColumnWidth = 12
NEXT I
ENDWITH

```

Что, если вам нужны категории в каждом столбце, а каждая строка представляет собой ряд? Вы можете сделать это. Когда вы создаете график с помощью метода ChartWizard (см. раздел «Создание графика» далее в этой главе), существует параметр, указывающий, расположены ли ряды в строках или в столбцах.

Имена диапазонов — ваши друзья

Пройдет немного времени, и ваш клиент запросит серию графиков, построенных на основе одних и тех же данных. Используя пример данных, показанный на рисунке 1, клиент может захотеть получить график для каждого года, а затем сравнить все годы на одном графике. Хотя вы можете создать отдельный рабочий лист только с необходимыми данными для графика (что приводит к избыточности данных), самый простой способ — поместить все данные на один рабочий лист, а затем использовать имена диапазонов для каждой серии и для меток осей категорий. Зачем диапазон имен? «Имена категорий» и «Год1994» легче читать и отлаживать, чем «Лист1!A2:A13» и «Лист1!D2:D13».

Имена хранятся в коллекции Names, доступной из объектов Application, Workbook и Worksheet. Используйте свойство Name объекта Range, чтобы назвать диапазон. Используя данные, показанные на рисунке 1, следующий код показывает имена примеров вместе с их диапазонами. Как и имена полей VFP, имена диапазонов должны начинаться с буквы; отсюда и название «Year1992», а не «1992».

```

WITH oExcel.Sheets[1]
.Range("A1:A13").Name = "CategoryNames"
.Range("B1:B13").Name = "Year1992"
.Range("C1:C13").Name = "Year1993"
.Range("D1:D13").Name = "Year1994"
.Range("E1:E13").Name = "Year1995"
.Range("F1:F13").Name = "Year1996"
ENDWITH

```

9.3 Создание диаграммы

Здесь усердно работает полиморфизм: используйте метод Add, чтобы добавить диаграмму. На этот раз есть небольшой поворот: вам нужно решить, какой метод Add коллекции использовать.

Если вы хотите добавить встроенную диаграмму, используйте метод Add коллекции ChartObjects. Он принимает четыре параметра: левый, верхний, высоту и ширину (в пунктах) диаграммы. Выполнение следующей команды вставляет объект диаграммы под пример данных:

```
oChart = oExcel.Sheets[1].ChartObjects.Add(45.7, 173.2, 319.5, 190.5)
```

Рисунок 2 показывает размещение пустого контейнера ChartObject. Не волнуйтесь, это действительно диаграмма. Метод ChartWizard позволяет легко установить наиболее распространенные свойства за один вызов метода, или вы можете установить их вручную. Мы предполагаем, что вы можете считать этот пустой график «функцией», поскольку он не принимал за вас решения о типе графика, поэтому отменить нечего.



Рисунок 2. Добавление встроенной диаграммы из метода ChartObjects.Add. Новая диаграмма пуста; Excel не делал никаких предположений по этому поводу. Используйте метод ChartWizard, чтобы легко заполнить его, или установите все свойства вручную.

Как мы определились со значениями, которые нужно передать? Немного грубой силы в командном окне. Мы создали общедоступную переменную oExcel, а затем запустили программу XLGDataSetup. Затем мы активировали Excel (с помощью Alt-Tab) и добавили диаграмму с помощью кнопки «Мастер диаграмм», а затем расположили полученный график там, где мы хотели.

Затем мы вернулись в VFP и запросили свойства Left, Top, Height и Width объекта ChartObject. Следующие команды дадут вам нужные цифры.

```
? oExcel.Sheets[1].ChartObjects[1].Left  
? oExcel.Sheets[1].ChartObjects[1].Top  
? oExcel.Sheets[1].ChartObjects[1].Height
```



```
? oExcel.Sheets[1].ChartObjects[1].Width
```

Если вы хотите добавить лист диаграммы, используйте метод Add коллекции Charts. Этому параметру не нужны параметры. Он помещает новый лист диаграммы перед текущим активным листом. Лист диаграммы называется Диаграмман (Chartn), где *n* – следующий доступный номер диаграммы.

```
oChart = oBook.Charts.Add()
```

Add также может принимать параметры для размещения нового листа диаграммы. Передайте объект Worksheet в качестве первого параметра, чтобы разместить диаграмму перед листом, или передайте объект Worksheet в качестве второго параметра, чтобы разместить её после указанного листа. Предупреждение: передача *имен листов* в качестве первого или второго параметра приводит к возникновению этой ошибки: «Код исключения OLE IDispatch 0 из Microsoft Excel: невозможно получить свойство Add класса Sheets». Обязательно передайте *объект Worksheet*.

Свойство ActiveChart

Если для добавления внедренной диаграммы используется только метод Add, результирующая диаграмма не выбирается. Чтобы сделать её доступной через свойство ActiveChart книги, вам необходимо использовать метод Activate диаграммы:

```
oChart.Activate()
```

Теперь диаграмма активна и доступна из свойства ActiveChart книги.

Иногда мы обнаруживали, что методы объекта Chart, особенно для встроенных диаграмм, завершаются с ошибкой «Неизвестное имя», если вызываются с полным именем ChartObject (например, Excel.Workbooks[1].Sheets["Sheet1"].ChartObjects[1] или любая переменная, для которой установлена полная ссылка), но они, похоже, работают нормально при вызове из свойства ActiveChart, когда та же самая диаграмма определенно активна (например, oExcel.ActiveWorkbook.ActiveChart). Мы не совсем понимаем, почему это происходит, но мы рады найти обходной путь, когда некоторые методы просто не хотят работать.

Выходим к мастеру

Существует единственный метод ChartWizard, который позволяет быстро форматировать диаграмму. Если вы знакомы с интерактивным методом построения диаграммы в Excel, параметры тоже будут знакомы, поскольку они во многом аналогичны вкладкам диалогового окна мастера диаграмм.

```
oChart.ChartWizard( [oSourceRange] [, nChartType] [, nChartFormat]
                    [, nPlotBy] [, nCategoryLabels]
                    [, nSeriesLabels]
                    [, lHasLegend] [, cTitle] [, cCategoryTitle]
                    [, cValueTitle] [, cExtraTitle] )
```

oSourceRange **Object** Объект Range, содержащий исходные данные для диаграммы.

nChartType **Numeric** Тип диаграммы. Используйте одну из следующих констант:

xlArea	1	xl3DArea	-4098
xlBar	2	xl3DBar	-4099

xlColumn	3	xl3DColumn	-4100
xlLine	4	xl3DLine	-4101
xlPie	5	xl3DPie	-4102
xlRadar	-4151	xl3DSurface	-4103
xlXYScatter	-4169	xlDoughnut	-4120
xlCombination	-4111		

Смотрите текст для получения дополнительной информации.

nChartFormat	Numeric	Вариант используемого типа диаграммы (например, если nChartType – xlBar, вы можете выбрать составную или кластеризованную полосу). Используйте число от 1 до любого доступного формата. Дополнительную информацию смотрите в тексте.
nPlotBy	Numeric	Указывает, хранятся ли ряды данных в строках или столбцах. Используйте одну из этих двух констант: xlRows 1 xlColumns 2 Значение по умолчанию основано на форме диапазона. Если строк больше, чем столбцов, значением по умолчанию является xlRows. Если столбцов больше, чем строк, значением по умолчанию является xlColumns. Если количество строк и столбцов одинаковое, используется xlRows.
nCategoryLabels	Numeric	Указывает, сколько строк (при построении по строкам) или столбцов (при построении по столбцам) использовать в качестве меток на оси категорий (X).
nSeriesLabels	Numeric	Указывает, сколько столбцов (при построении по строкам) или строк (при построении по столбцам) нужно использовать для обозначения ряда. Обычно метки появляются в легенде.
lHasLegend	Logical	Указывает, есть ли на диаграмме легенда (.T.) или нет (.F.).
cTitle	Character	Строка символов, используемая для названия диаграммы. По умолчанию она располагается по центру чуть выше области графика.
cCategoryTitle	Character	Строка символов, используемая для названия оси категории (X).
cValueTitle	Character	Строка символов, используемая для названия оси значений (Y).
cExtraTitle	Character	Строка символов, используемая для названия второй оси значений на 2D-диаграммах (если используются две отдельные оси значений) или в качестве названия оси ряда (Z) на 3D-диаграммах.

Вся эта мощь в одном методе имеет свою цену: объяснение всех параметров занимает немало страниц. Объясняя каждый из параметров, мы также расскажем вам, какие свойства объекта Chart задаются методом ChartWizard и доступны ли

какие-либо альтернативные методы для установки этих параметров. Эти свойства и методы обсуждаются в остальных разделах этой главы, поэтому мы отметим раздел, на который можно ссылаться для полного обсуждения этих свойств.

Исходный диапазон

Это объект Range, в котором находятся данные. Для встроенных диаграмм это может быть, но не обязательно, на одном листе. Очевидно, что данные для листов диаграмм находятся на других листах. Чтобы передать диапазон, используемый для заполнения встроенной диаграммы, показанной на рисунке 2, передайте следующий объект в качестве параметра oSourceRange:

```
oBook.Sheets["Лист1"].Range("A1:F13")
```

Обратите внимание, что в этот диапазон мы включили ячейки, содержащие метки категорий и значений. Если мастеру диаграмм не передаются никакие другие параметры, он предполагает, что строки или столбцы не используются в качестве меток, поэтому на графике отображаются 13 категорий вместо 12 месяцев и шесть рядов вместо пяти лет. Если вы решите включить метки категорий и значений, обязательно включите параметры nCategoryLabels и nSeriesLabels.

Несмежные диапазоны

Было бы неплохо, если бы все графики можно было построить на основе смежных диапазонов данных. Однако вы обнаружите, что бывают случаи, когда вам нужно использовать один лист для создания нескольких графиков или поместить данные из нескольких листов в один график. В любом случае ряды данных не являются смежными.

К счастью, Excel предоставляет возможность использовать несмежные данные. Все что необходимо – это перечислить диапазоны через запятую. Обязательно сначала добавьте строку с меткой категории. В нашем примере давайте построим график с месяцами в качестве названий категорий и отобразим 1993 и 1994 годы в виде рядов данных. Есть два способа сделать это. Используя адреса ячеек, код выглядит следующим образом:

```
oBook.Sheets["Лист1"].Range("A1:A13, C1:C13, D1:D13")
```

Используя имена диапазонов (см. раздел «Имена диапазонов – ваши друзья» ранее в этой главе), вы получаете следующий более читаемый код:

```
oBook.Sheets["Лист1"].Range("CategoryNames, Year1993, Year1994")
```



Обязательно присваивайте имена диапазонам, используя свойство Name объекта Range, а не метод Add коллекции Names. Мы не знаем почему, но диапазон, добавленный с помощью Names.Add, похоже, не распознается методом ChartWizard. Вы получаете ошибку OLE: 0x800a03ec, Неизвестный код состояния COM. Честно говоря, это не единственный способ получить эту ошибку. Вы также можете вызвать эту ошибку, передав неправильно написанное или несуществующее имя диапазона.

Обратите внимание, что для каждого диапазона (включая диапазон «CategoryNames») диапазон включает строку для меток рядов. При передаче диапазона, включающего заголовки, убедитесь, что параметры nCategoryLabels и nSeriesLabels (пятый и шестой параметры) имеют значение 1 (или больше, если используется больше строк/столбцов).

Другие исходные свойства и методы

Ряды данных хранятся в объекте коллекции SeriesCollection, в котором хранится объект Series для каждого ряда данных. Метод Add SeriesCollection принимает пять параметров. Это исходный диапазон (так же, как ChartWizard), числовое значение PlotBy (так же, как параметр nPlotBy ChartWizard), логическое значение, которое является истинным, если первая строка или столбец содержит метки рядов, другое логическое значение, которое является истинным, если первый столбец или строка содержит метки категорий и, наконец, логическое значение, определяющее, заменяются ли метки категорий или нет. Выполнение следующей команды аналогично добавлению исходного диапазона через ChartWizard (с использованием параметров oSourceRange, nPlotBy, nSeriesLabels и nCategoryLabels):

```
#DEFINE xlColumns 2
oExcel.ActiveChart.SeriesCollection.Add("Лист1!A1:F13", ;
                                         xlColumns, .T., .T.)
```

Объекты Series хранят много информации о форматировании. Более подробно это описано далее в этой главе в разделе «Форматирование компонентов».

Другой альтернативой, новой для Excel 2000, является метод SetDataSource объекта Chart. Этот метод принимает два параметра: исходный диапазон и числовое значение PlotBy. Следующее эквивалентно методу SeriesCollection.Add и методу ChartWizard:

```
#DEFINE xlColumns 2
oExcel.ActiveChart.SetSourceData(oExcel.Sheets[1].Range(
                                "Лист1!A1:F13"), xlColumns)
```

Типы и форматы диаграмм

Второй и третий параметры ChartWizard работают вместе. Первым из них является nChartType – числовое значение, соответствующее типу диаграммы, например диаграмме с областями, круговой диаграмме, линейной диаграмме или гистограмме. Вторым из этих параметров, nChartFormat, является подтипом или тонкой настройкой основного типа диаграммы. Например, гистограмма может представлять собой гистограмму с кластерами или гистограмму с накоплением. На рисунке 3 показано, как эти два параметра работают в интерактивном мастере диаграмм. В списке слева показан тип диаграммы (с миниатюрой, которая поможет вам), а подтип диаграммы показан справа с более крупными значками, которые помогут вам точно выбрать, какую диаграмму использовать.

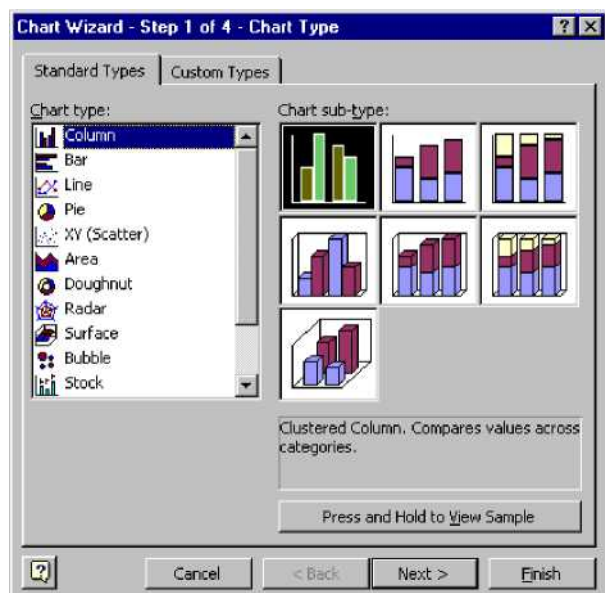


Рисунок 3. Типы и подтипы диаграмм, как видно из мастера диаграмм Excel. Это иллюстрирует, как параметры `nChartType` (соответствует списку типов диаграмм) и `nChartFormat` (соответствует кнопкам подтипа справа) работают вместе.

Есть только одна проблема с использованием рисунка 3 в качестве образца: это одно из немногих диалоговых окон, в котором не указаны параметры, используемые в соответствующем методе! Мы показали рисунок 3, чтобы проиллюстрировать концепцию типа и формата диаграммы, но не полагайтесь на то, что диалоговое окно даст вам подсказки относительно используемых значений.

Параметр `nChartType` использует ряд значений, каждое из которых имеет собственное имя константы VBA. К сожалению, констант для параметра `nChartFormat` нет. Фактически, они даже не соответствуют подтипам диаграмм, показанным на рисунке 3, поскольку метод `ChartWizard` отделяет трехмерные графики от двухмерных, а также смешивает и сопоставляет типы. Приложение В показывает полный список доступных значений; В таблице 1 показаны наиболее распространенные значения.

Таблица 1. Пример параметров `nChartType`. «Значение `ChartFormat`» – это столбец, который будет использоваться в мастере диаграмм; «Значение `ChartType`» – это результирующее значение свойства `ChartType` объекта `Chart`. Приложение В содержит полный список.

Описание формата диаграммы	Значение <code>ChartFormat</code>	Значение <code>ChartType</code>
Для типа диаграммы: <code>xlArea</code> (с областями)	1	
Диаграмма с областями с накоплением	1	76
Диаграмма с областями с накоплением с черными линиями сетки	4	76
Для типа диаграммы: <code>xlBar</code> (линейчатая)	2	
Линейчатая диаграмма	1	57
Линейчатая диаграмма с накоплением	3	58
Линейчатая диаграмма с 0 перекрытий и 0 шириной зазора	8	57

Описание формата диаграммы	Значение ChartFormat	Значение ChartType
Для типа диаграммы: xlColumn (гистограмма)	3	
Кластеризованная колонна с шириной зазора, установленной на 150	1	51
Обычная гистограмма с 0 перекрытий и 0 шириной зазора	8	51
Нормированная гистограмма с последовательными линиями	10	53
Для типа диаграммы: xlLine (график)	4	
Линия с маркерами	1	65
Линия без маркеров данных	2	4
Только маркеры данных (без соединительных линий)	3	65
Сглаженная линия без маркеров	10	4
Для типа диаграммы: xIPie (круговая)	5	
Круговая диаграмма без надписей	1	5
Круговая диаграмма с надписями, с выделением первого клина	2	5
Разрезанная круговая диаграмма	4	69
Для типа диаграммы: xl3DArea (объемный вариант диаграммы с областями)	-4098	
Объемная диаграмма с областями с накоплением	1	78
Объемная диаграмма с областями с накоплением с метками серий	2	78
Объемная диаграмма с областями, вертикальные линии сетки	7	-4098
Для типа диаграммы: xl3DBar (объемный вариант линейчатой диаграммы)	-4099	
Объемная линейчатая диаграмма	1	60
Объемная линейчатая диаграмма с накоплением	2	61
Для типа диаграммы: xl3DColumn (объемный вариант гистограммы)	-4100	
Объемная гистограмма	1	54
Объемная гистограмма с накоплением	2	55
Объемная нормированная гистограмма	3	56
Для типа диаграммы: xl3DPie (объемный вариант круговой диаграммы)	-4102	
Объемная круговая диаграмма без надписей	1	-4102

Описание формата диаграммы	Значение ChartFormat	Значение ChartType
Разрезанная объемная круговая диаграмма без надписей	4	70

Другие свойства типа диаграммы

Параметры nChartType и nChartFormat задают свойство ChartType объекта Chart. Чтобы убедиться, что жизнь не слишком проста, в Excel есть 73 числовых значения, которые плохо связаны (по крайней мере математически) с комбинациями параметров nChartType и nChartFormat. Вы можете увидеть результирующие значения, перечисленные в столбце «Значение ChartType» в таблице 1. Мы обсудим их далее более подробно в этой главе в разделе «Типы диаграмм».

Эти два параметра не только задают свойство ChartType, но также задают многие свойства различных объектов Axis, включая ScaleType, HasMajorGridlines и HasRadarAxisLabels (если это лепестковая диаграмма). Также задаются свойства серии, включая AxisGroup, ChartType, Explosion, HasDataLabels, HasErrorBars и различные свойства Marker.

Дополнительную информацию об этих и других связанных свойствах см. далее в этой главе в соответствующих частях раздела «Форматирование компонентов».

PlotBy – выбор строк или столбцов

Параметр nPlotBy определяет, используются ли строки или столбцы в качестве рядов данных. Доступны два значения: xlRows (1) и xlColumns (2). Значение по умолчанию зависит от формы диапазона: если столбцов больше, чем строк, по умолчанию используются столбцы; в противном случае по умолчанию используются строки. Из-за такой «движущейся цели» значения по умолчанию мы предпочитаем всегда задавать этот параметр, а не предполагать что-либо.

Другие свойства PlotBy

Это правильный параметр, поскольку он без каких-либо догадок задает только одно свойство, и оно даже имеет то же имя! Свойство PlotBy объекта Chart использует одни и те же два значения: xlRows (1) и xlColumns (2).

Ярлыки категорий и рядов

Параметры nCategoryLabels и nSeriesLabels определяют, сколько строк или столбцов используется в исходном диапазоне для обозначения категорий и рядов. Параметр nPlotBy определяет, являются ли метки категорий строками или столбцами. Если nPlotBy имеет значение xlRows, то метки категорий находятся в первых n строках, а метки рядов – в первых n столбцах. И наоборот, если nPlotBy имеет значение xlColumns, то метки категорий находятся в первых n столбцах, а метки рядов – в первых n строках.

Обычно значение nCategoryLabels и nSeriesLabels равно 0 или 1. Однако вы можете использовать два или более столбца/строки. Вам следует поэкспериментировать с визуальными результатами; наличие двух строк для каждой категории может быть слишком загруженным, если используется много категорий. В любом случае, в зависимости от места, Excel может опустить вторую строку.

Другие свойства метки

Метки категорий хранятся в виде диапазона в свойстве XValues объекта Series. Метки рядов хранятся в свойстве Name объекта Series. Подробнее об этом говорится в разделе «Ряды данных» далее в этой главе.

Есть ли у него легенда?

Параметр IHasLegend определяет, имеет ли результирующая диаграмма легенду (также известную как «ключ»). Все, что он делает, это делает легенду видимой (или нет). Передайте логическое значение, указывающее, должна ли диаграмма иметь легенду.

Другие свойства и объекты легенды

Этот параметр задает свойство HasLegend объекта Chart, которое является логическим значением. Вы можете запросить это значение и получить логическое значение (в отличие от многих, казалось бы, логических значений). Если значение истинно, значения легенды задаются с помощью объекта Legend объекта Chart, который подробно обсуждается в разделе «Легенды» далее в этой главе.

Параметры заголовка

Последние четыре параметра cTitle, cCategoryTitle, cValueTitle и cExtraTitle представляют собой просто строки символов, которые размещаются на диаграмме как заголовки. Параметр cTitle используется в качестве заголовка диаграммы и располагается по центру верхней части диаграммы. Параметры cCategoryTitle и cValueTitle являются заголовками осей и располагаются по центру соответствующей оси. cExtraTitle используется только в двух случаях: если на двухмерной диаграмме есть вторая ось, или если диаграмма является трехмерной, а для третьей оси требуется заголовок. См. пример ChartWizard в следующем разделе, где показано, где размещаются заголовки.

Другие свойства заголовка

Параметр cTitle устанавливает для свойства HasTitle диаграммы значение True и делает объект ChartTitle доступным (через свойство ChartTitle). Остальные параметры заголовка — это заголовки осей, которые устанавливают для свойства HasTitle соответствующего объекта Axis значение True и делают объект AxisTitle доступным (через свойство AxisTitle). Они подробно обсуждаются в разделе «Заголовки» далее в этой главе.

Наконец, пример волшебства

После всех объяснений различных параметров пришло время посмотреть, как эта штука работает на самом деле. В качестве примера мы построим гистограмму с кластеризацией из всего диапазона данных примера, построив ряды по столбцам и используя один столбец и строку для меток. Легенда рассчитана, и мы будем использовать примеры заголовков, чтобы показать, где находятся различные заголовки. Готовый? Вот оно:

```
#DEFINE xlColumn          3
#DEFINE xlColumns         2
#DEFINE autoColumnFormat  4
#DEFINE autoOneCategoryLabel 1
#DEFINE autoOneSeriesLabel 1
#DEFINE autoHasLegend     .T.
```

```
oSourceRange = oExcel.Sheets[1].Range("A1:F13")
oExcel.ActiveChart.ChartWizard(oSourceRange, xlColumn, ;
```

```

autoColumnFormat, xlColumns, autoOneCategoryLabel, ;
autoOneSeriesLabel, autoHasLegend, ;
"Название диаграммы", "Название категории", ;
"Название значения", "Дополнительный заголовок")

```

Рисунок 4 показывает результаты метода ChartWizard. Как видите, в этом методе заключено много возможностей. Изучая полученный график, вы видите множество особенностей метода ChartWizard. На красиво отформатированной диаграмме есть заголовки, метки осей и легенда. Если есть что-то, что вы хотите улучшить, вы, безусловно, можете это сделать. Существует множество свойств и методов, которые помогут вам уточнить формат, см. раздел «Форматирование компонентов» далее в этой главе.

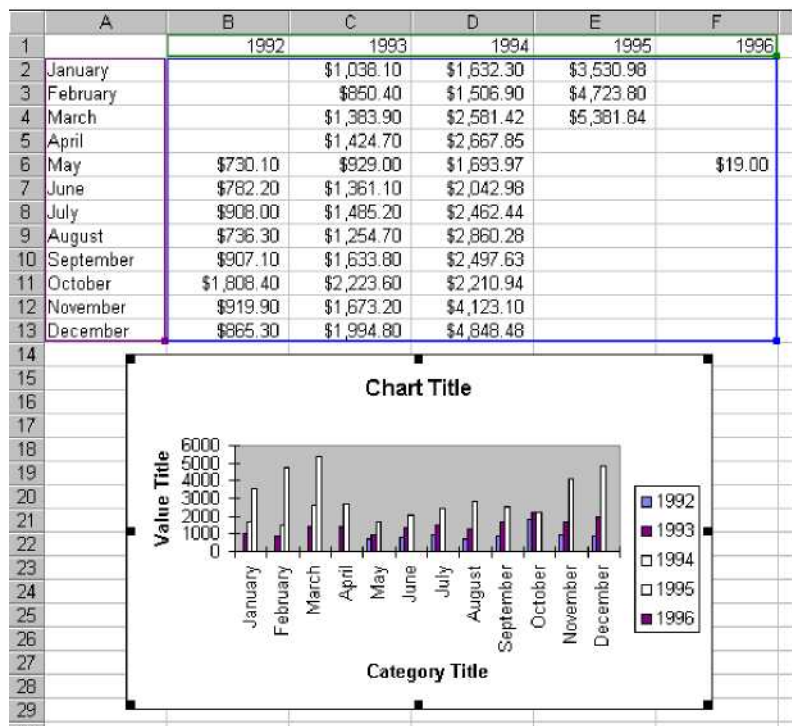


Рисунок 4. Результаты метода ChartWizard. Хотя список параметров у метода ChartWizard длинный, он обладает большими возможностями для форматирования диаграммы.

Существует альтернатива использованию ChartWizard. Вы можете управлять каждым из объектов отдельно. Это потребует больше кода, но вы получите больше контроля. Мы обнаружили, что гибридный подход работает лучше всего: используйте ChartWizard, чтобы приблизиться к желаемому результату, а затем уточните форматирование. В следующих разделах мы обсудим различные объекты, содержащиеся в диаграмме, и способы их уточнения. Затем мы рассмотрим код для создания этой диаграммы вручную без ChartWizard.

9.4 Анатомия диаграммы

Теперь, когда у нас есть диаграмма, мы можем поговорить о ее частях. Диаграмму составляет так много объектов, что терминология быстро запутывается.

Область диаграммы – это вся область диаграммы. Для встроенных диаграмм это область, ограниченная рамкой. Когда она выбрана, она имеет восемь маркеров изменения размера, как показано на рисунке 4. Для листов диаграмм областью диаграммы является весь лист. Основное различие между листами диаграмм и встроенными диаграммами заключается в том, что область диаграммы на встроенной диаграмме можно перемещать и изменять ее размер (с помощью

свойств Top, Left, Height и Width), в то время как размер и положение листа диаграммы фиксируются на границы листа. Область диаграммы – единственный объект, на который влияет то, является ли диаграмма листом диаграммы или встроенной диаграммой. Объект Chart хранит объект ChartArea в свойстве ChartArea.

Площадь графика – это площадь, ограниченная осями. Это серая область, показанная на рисунке 4. Доступ к объекту PlotArea осуществляется из свойства PlotArea объекта Chart.

На диаграммах доступно несколько типов осей. На рисунке 4 показаны две наиболее распространенные: *ось категорий*, также известная как X или горизонтальная ось, и *основная ось значений*, известная как Y или вертикальная ось. Excel предусматривает две оси значений; если бы использовалась *вторичная ось значений*, она отображалась бы в правой части графика. Для некоторых трехмерных диаграмм существует ось *рядов*, также известная как ось Z. Доступ к коллекции Axes осуществляется через метод Axes объекта Chart.

Легенда – это поле справа от области графика. Она показывает образец форматирования для каждого ряда вместе с метками рядов. Доступ к объекту Legend осуществляется через свойство Legend объекта Chart.

Ряды данных – это точки данных, нанесенные на график в одном и том же формате. Каждый ряд хранится в объекте SeriesCollection. Объект Series хранит диапазоны данных для данных, отображаемых на графике, метки категорий и метки рядов. Он также хранит все параметры форматирования, такие как параметры цвета и форматирование, специфичное для типа диаграммы (столбцы максимумов и минимумов, маркеры данных, типы столбцов и т. д.). Каждый ряд может иметь свой тип диаграммы; например, вы можете иметь один ряд в виде линии, следующий в виде диаграммы с областями, а третий в виде столбчатой диаграммы. Поскольку во всех рядах используются одни и те же оси, вы можете смешивать и сопоставлять типы диаграмм. (Другими словами, одна диаграмма может содержать несколько разных типов диаграмм, но не планируйте помещать круговые диаграммы в одну диаграмму в качестве линий и столбцов.)

Внутри каждого ряда находится набор *точек данных*. Каждую точку можно пометить и/или отформатировать отдельно. Форматирование точки привлекает внимание к этому элементу данных, например, её можно использовать для определения максимума (или минимума) ряда. Предупреждение: если выделить слишком много точек данных, диаграммы могут запутать зрителя. Если вы выделяете на диаграмме более пары точек, убедитесь, что эти точки действительно нуждаются в выделении. Получите доступ к коллекции Points из метода Points объекта Series.

9.5 Форматирование компонентов

Метод GraphWizard не позволяет настроить график. Многие клиенты знакомы с графиками, которые может создавать Excel, и просят реализовать многие из этих функций. Они попросят вас переместить легенду на другую сторону, увеличить шрифт на метках осей, добавить линии сетки, изменить цвета (чаще всего клиенты *любят настаивать* на изменении цвета) и так далее.

Следующие темы обычно организованы так, чтобы соответствовать объектам и коллекциям, которые являются частью диаграммы. Некоторые объекты, например SeriesCollection, доступны всегда. Другие, например объекты Title, доступны только в том случае, если для определенного свойства, например HasTitle, установлено значение True.

Независимо от того, создаете ли вы диаграмму с нуля или изменяете диаграмму, созданную с помощью ChartWizard, вы найдете все, что вам нужно, в следующих разделах. Оставшуюся часть этой главы мы организовали так, чтобы начать с обсуждения свойств и методов объекта Chart, а затем поработаем над подчиненными объектами объекта Chart.

9.6 Объект Chart

Мы рассмотрели многие основы объекта Chart: как его добавить и как использовать метод ChartWizard. Теперь мы можем рассмотреть объект более детально. Многие свойства и методы объекта Chart ссылаются на другие объекты и коллекции. Свойства ChartArea и PlotArea ссылаются на объекты ChartArea и PlotArea и всегда доступны.

Некоторые объекты не всегда доступны. Заголовки диаграмм, легенды и оси являются необязательными компонентами. Объект Chart имеет свойства, позволяющие определить, доступны ли эти объекты. Три основных свойства, которые обеспечивают доступ к дополнительным компонентам:

- **HasAxis.** Если истина, метод Axes доступен для доступа к коллекции Axes.
- **HasLegend.** Если истина, свойство Legend доступно для доступа к объекту Legend.
- **HasTitle.** Если истина, свойство ChartTitle доступно для доступа к объекту ChartTitle.

Есть несколько свойств, которые не имеют доступа к объектам. Эти глобальные свойства, показанные в таблице 2, влияют на всю диаграмму.

Таблица 2. Свойства объекта диаграммы. Эти свойства не ссылаются на другие объекты; они влияют на внешний вид диаграммы.

Свойство	Тип	Описание												
ChartType	Numeric	Дополнительную информацию см. в следующем разделе «Типы диаграмм».												
BarShape	Numeric	Определяет форму полос или столбцов в трехмерной линейчатой или трехмерной столбчатой диаграмме. <table><tr><td>xlBox</td><td>0</td><td>xlCylinder</td><td>3</td></tr><tr><td>xlConeToMax</td><td>5</td><td>xlPyramidToMax</td><td>2</td></tr><tr><td>xlConeToPoint</td><td>4</td><td>xlPyramidToPoint</td><td>1</td></tr></table> Дополнительную информацию см. в следующем разделе «Типы диаграмм».	xlBox	0	xlCylinder	3	xlConeToMax	5	xlPyramidToMax	2	xlConeToPoint	4	xlPyramidToPoint	1
xlBox	0	xlCylinder	3											
xlConeToMax	5	xlPyramidToMax	2											
xlConeToPoint	4	xlPyramidToPoint	1											
DisplayBlanksAs	Numeric	Определяет способ отображения пустых ячеек на диаграмме. Использует следующие значения: <table><tr><td>xlNotPlotted</td><td>1</td><td>Не отображайте точки данных (например, ряд линий показывает пробел для пустых данных).</td></tr><tr><td>xlInterpolated</td><td>3</td><td>Определите точку данных по двум точкам данных с каждой стороны (например, ряд линий рисуется от точки перед пустым значением до точки после).</td></tr></table>	xlNotPlotted	1	Не отображайте точки данных (например, ряд линий показывает пробел для пустых данных).	xlInterpolated	3	Определите точку данных по двум точкам данных с каждой стороны (например, ряд линий рисуется от точки перед пустым значением до точки после).						
xlNotPlotted	1	Не отображайте точки данных (например, ряд линий показывает пробел для пустых данных).												
xlInterpolated	3	Определите точку данных по двум точкам данных с каждой стороны (например, ряд линий рисуется от точки перед пустым значением до точки после).												

		xlZero 2 Отобразите точки данных как ноль (например, ряд строк опускается до нуля для пустых данных).
Name	Character	Указывает имя диаграммы. Для листов диаграмм это имя используется для обозначения вкладки листа. Как для листов диаграмм, так и для встроенных диаграмм имя используется для доступа к объектам Chart или ChartObject. По умолчанию для листов диаграмм используется «Диаграмма <i>n</i> », где <i>n</i> – следующий доступный номер диаграммы. По умолчанию для встроенных листов используется «<SheetName> Chart <i>n</i> » (обратите внимание на пробелы), где <SheetName> – это имя текущего рабочего листа (и наследуется, если имя листа изменяется, как и эта часть имени диаграммы) и <i>n</i> – следующий доступный номер объекта Chart.
PlotVisibleOnly	Logical	Указывает, отображаются ли видимые и скрытые ячейки. Значение True для отображения только видимых ячеек, значение False для отображения как видимых, так и скрытых ячеек. Это эффективно только в том случае, если на листе включены функции защиты.
Visible	Logical	Указывает, видна ли диаграмма.

Несколько методов обеспечивают некоторые интересные функции; они обсуждаются в соответствующих разделах в конце этой главы. Это GetChartElement, PrintOut, PrintPreview, SaveAs и Export. Метод SetBackgroundPicture обсуждается с объектом ChartArea, поскольку он влияет на форматирование области диаграммы.

Типы диаграмм

Мы немного коснулись типов диаграмм в разделе «Мастер диаграмм». Метод ChartWizard использует два параметра: тип диаграммы и формат диаграммы, которые в совокупности определяют внешний вид диаграммы. Результаты этой комбинации сохраняются как одно число в свойстве ChartType объекта Chart (некоторые форматы также могут устанавливать некоторые свойства объектов Axis и Series).

К сожалению, нам неизвестен алгоритм преобразования параметров ChartWizard в значения ChartType, поэтому мы перечислили значения в таблице 1 и приложении В.

Для каждого значения в Excel имеется определенная константа. Чтобы сэкономить место в таблице 1, мы указали только значение. Дополнительную информацию можно получить в Обзорщике объектов. Некоторые из констант: xl3DColumn (-4100), xl3DColumnClustered (54), xlBarClustered (57), xlConeBarClustered (102), xlCylinderColClustered (92) и xlLineMarkers (65).

Каковы эти постоянные конуса и цилиндра? На самом деле, вы также можете добавить константы пирамиды. Эти диаграммы недоступны для мастера диаграмм. В данном случае вам необходимо вручную установить тип графика, чтобы полосы (или столбцы) отображались в виде конусов, цилиндров или пирамид, примеры которых показаны на рисунке 5. Итак, чтобы задать график для столбчатого кластера цилиндров, выполните следующую команду:

```
#DEFINE xlCylinderColClustered 92
oExcel.ActiveChart.ChartType = xlCylinderColClustered
```

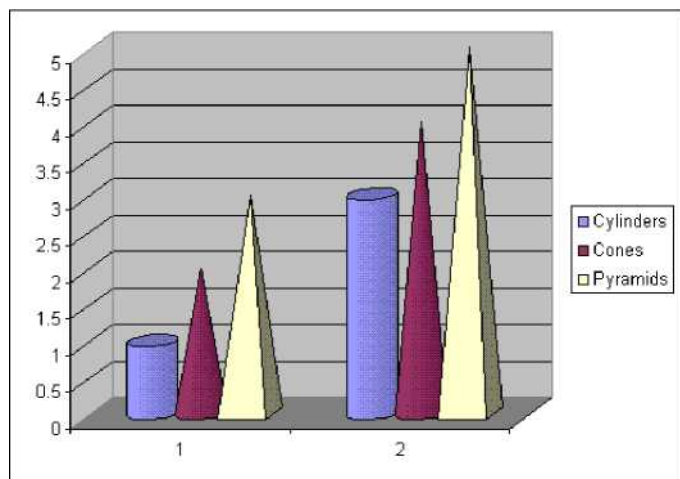


Рисунок 5. Примеры различных форм для трехмерных гистограмм и столбчатых диаграмм. Цилиндры, конусы и пирамиды сделают ваши презентации интереснее.

На самом деле, если вы изучите Приложение Б, вы обнаружите, что представлены только 33 из 73 значений ChartType. Мастер диаграмм отлично подходит для настройки наиболее распространенных типов диаграмм, но для некоторых необычных графиков вам придется вручную установить тип диаграммы.

Другой способ получить формы цилиндра, конуса и пирамиды в гистограммах и столбчатых диаграммах – задать ChartType как трехмерную линейчатую или столбчатую диаграмму, а затем установить свойство BarShape. Когда свойство BarShape установлено, оно обновляет свойство ChartType, чтобы отразить новое значение ChartType.

Свойства объекта Chart доступные только для трехмерных диаграмм

Некоторые объекты и свойства доступны только в том случае, если диаграмма имеет трехмерные свойства. Свойства Floor и Walls ссылаются на объекты Floor и Walls соответственно. Walls – это стороны трехмерной диаграммы, а Floor – нижняя часть. Эти объекты похожи и содержат два основных свойства для форматирования объектов. Свойство Border и свойство Interior, описанные далее в этой главе в разделах «Объект Border» и «Объект Interior», соответственно управляют всем форматированием. Обычно вам нужно установить их, используя те же свойства, что и для PlotArea.

Свойство DepthPercent хранит глубину диаграммы (вдоль оси Z) в процентах от ширины диаграммы. Допустимые значения находятся в диапазоне от 20 до 2000. Свойство HeightPercent хранит высоту диаграммы в процентах от ширины диаграммы. Допустимые значения находятся в диапазоне от 5 до 500 процентов.

Два свойства задают угол просмотра диаграммы. Elevation – это высота, с которой вы видите диаграмму в градусах. На рисунке 6 показаны два графика: один с высотой, установленной на 10, а другой с максимальным значением 90. Для большинства типов диаграмм значение по умолчанию составляет 15 градусов. Допустимый диапазон свойства Elevation – от -90 до 90, за исключением гистограмм, которые могут находиться в диапазоне от 0 до 44.

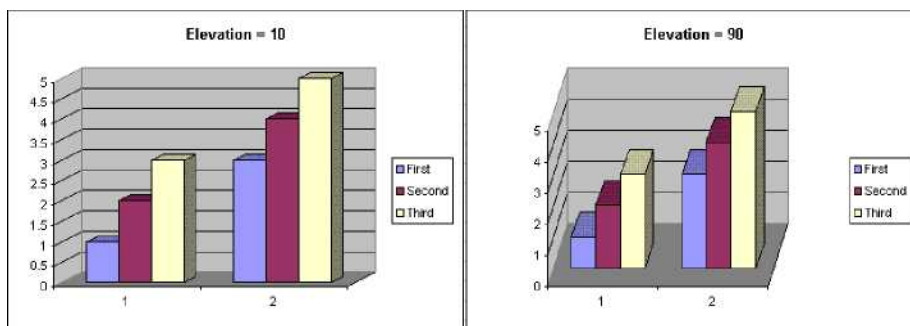


Рисунок 6. Иллюстрация свойства Elevation. Elevation - это высота, с которой вы видите диаграмму. На диаграмме слева для свойства Elevation установлено значение 10, что соответствует уровню глаз. На диаграмме справа показана максимальная высота, или 90, если смотреть на диаграмму вниз. Минимальное значение – -90 или если смотреть снизу диаграммы.

Свойство Rotation определяет, насколько диаграмма повернута влево или вправо (вокруг оси Y) в градусах. Допустимые значения находятся в диапазоне от 0 до 360, за исключением гистограмм, которые также могут находиться в диапазоне от 0 до 44. Значение по умолчанию – 20 для всех диаграмм. На рисунке 7 показаны два графика с разным вращением.

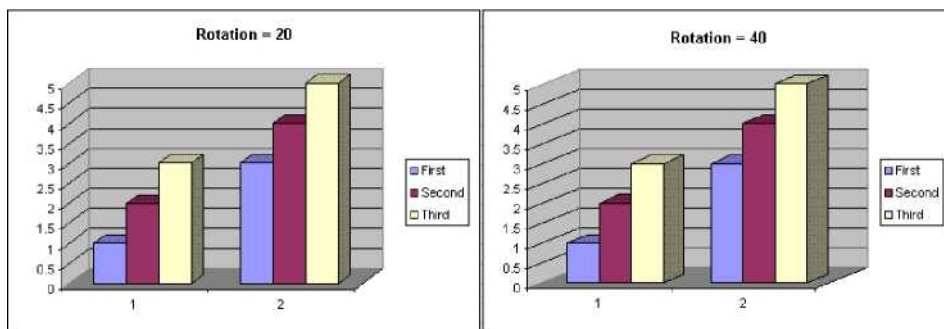


Рисунок 7. Вращение графика. На диаграмме слева показан поворот по умолчанию на 20 градусов, а на диаграмме справа – на 40 градусов.

Диаграммы, показанные на рисунке 7, построены в стиле, называемом изометрической проекцией. Этот стиль трехмерного рисования используется для строительных чертежей, поскольку точные измерения можно выполнить из любой точки диаграммы. Это контролируется логическим свойством RightAngleAxes; если оно равно True, используется изометрическая проекция, которая характеризуется расположением осей под прямым углом друг к другу. Если оно имеет значение False, вы можете установить свойство Perspective, которое принимает значение от 0 до 100. Перспектива – это еще один стиль трехмерного рисования, в котором линии, параллельные в реальности, например горизонтальные линии сетки, не рисуются параллельно и встречаются в одной точке. называется точкой схода. Этот метод рисования используется для иллюстрации, и измерения неточны, поскольку объекты на переднем плане обязательно больше, чем объекты на заднем плане. Перспектива – это техника, которая делает 3D-иллюстрации более реалистичными. Когда вы устанавливаете значение 0, точка схода находится далеко на расстоянии, поэтому линии выглядят параллельными. Если установлено максимальное значение 100, точка схода находится близко к графику, что сильно искажает график и переоценивает большой конец графика. Значение по умолчанию – 30. На рисунке 8 показаны два графика: для одного свойство Perspective установлено на 25, а для другого – на 100. Сравните эти графики с графиками, показанными на рисунке 7, у которых для свойства RightAngleAxes установлено значение True, и используется изометрическая проекция.

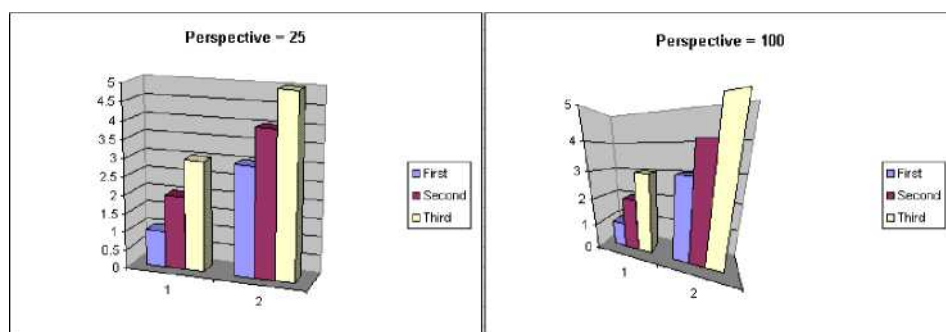


Рисунок 8. Разница во взглядах. Использование перспективы усиливает иллюзию трехмерности, как показано на диаграмме слева (сравните с квадратным видом на рисунке 7). Диаграмма справа показывает максимальную перспективу, которая дает совсем другой эффект.

Область диаграммы

ChartArea – это область, содержащая всю диаграмму. Как упоминалось ранее, если диаграмма является встроенной, у вас есть доступ к свойствам Top, Left, Height и Width, чтобы задать размер и расположение встроенной диаграммы. Эти свойства недоступны в листах диаграмм, поскольку размер диаграммы позволяет занимать весь лист; эти свойства бессмысленны в этой ситуации.

Есть одно свойство, которое не ссылается на объект. Это свойство AutoScaleFont. Установите для этого параметра значение False, если вы не хотите, чтобы текст менял размер при изменении размера диаграммы. По умолчанию True.

Три свойства влияют на объекты, которые форматируют всю область диаграммы. Это свойство Border (ссылается на объект Border), свойство Interior (ссылается на объект Interior) и свойство Font (ссылается на объект Font). Эти три объекта обычно используются для форматирования многих объектов диаграммы (например, PlotArea), а объекты Font и Border даже используются в других областях Microsoft Office.

Объект Border

В главе 7 «Основы Excel» мы рассмотрели коллекцию Borders, которая используется для форматирования отдельных сторон каждой ячейки в диапазоне. Диапазоны работают с коллекцией объектов Borders, по одному для каждой из восьми различных границ. Все остальные объекты работают с одной рамкой, которая одинаково влияет на все стороны объекта. ChartArea, PlotArea и другие обычно имеют прямоугольные границы, причем все четыре стороны имеют одинаковый формат.

Все объекты Border имеют одинаковые свойства; они перечислены в Таблице 3.

Таблица 3. Свойства границы. Вокруг многих объектов можно размещать границы, а их внешний вид можно изменять с помощью этих свойств.

Свойство	Тип	Описание
Color	RGB Color	Цвет границы.
LineStyle	Numeric	Числовое значение, соответствующее предустановленному стилю линии.
		xlContinuous 1 xlDot -4118
		xlDash -4115 xlDouble -4119

		xlDashDot	4	xlLineStyleNone	-4142
		xlDashDotDot	5	xlSlantDashDot	13
Weight	Numeric	Ширина линии. Здесь используются константы, а не пункты.			
		xlHairline	1	xlMedium	-4138
		xlThin	2	xlThick	4

Чтобы удалить рамку вокруг диаграммы, используйте следующий код:

```
#DEFINE xlLineStyleNone -4142
oExcel.ActiveChart.ChartArea.Border.LineStyle = xlLineStyleNone
```

Помните, что вы можете выбрать необычный стиль линии или выбрать стиль непрерывной линии и установить толщину. Вы не можете совмещать и то, и другое.

Объект Interior

Объект Interior управляет тем, как выглядит внутренняя часть объекта. Чтобы изменить цвет, установите для свойства Color объекта Interior нужный цвет RGB. В данном примере выбран пастельный синий цвет:

```
oExcel.ActiveChart.ChartArea.Interior.Color = RGB(192,192,255)
```



Хотя вы можете указать значение RGB, Excel выберет ближайший цвет в своей палитре (посмотрите это в диалоговом окне «Формат ряда данных», которое можно открыть, щелкнув правой кнопкой мыши любой ряд данных и выбрав «Формат рядов данных» во всплывающем меню). Excel поддерживает одновременно только 56 цветов; Вы можете манипулировать палитрой, используя коллекцию Colors Workbook и изменяя один из 56 цветов на значение RGB. Только тогда вы сможете установить для объекта точное значение RGB.

Объект Interior также имеет свойство Pattern. Да, вы *можете* создавать шаблоны для области диаграммы, области графика, столбцов и полос и т. д., но используйте эту функцию с осторожностью! Узоры могут быть визуально подавляющими, особенно если в каждом столбце используется разный узор и цвет. Если диаграмма цветная, на выбор должно быть много цветов. Хорошо, вы *можете* использовать цвет и шаблон одного цвета для обозначения пар рядов, например фактический и прогнозируемый, где фактические значения представляют собой сплошные цвета, а столбцы прогноза представляют собой шаблон одного и того же цвета. Шаблоны действительно превосходны, когда вам нужно печатать в черно-белом режиме, и вы просто не можете полагаться на то, как принтер отображает цвета в оттенках серого.

Однако мы настоятельно не рекомендуем создавать шаблоны в области графика, особенно в области диаграммы. Узоры слишком малы для больших площадей и выглядят некрасиво.

Учитывая эти предостережения, если вы хотите создать узор на объекте, вот как это сделать. Задайте для свойства Color «фоновый» цвет узора. Задайте для свойства PatternColor цвет узора. Некоторые шаблоны, такие как Checker и Gray50, используют одинаковое количество Color и PatternColor, поэтому не имеет значения, какой из них вы выберете. Однако CrissCross состоит из тонких линий, поэтому установка для PatternColor значения красного, а Color – белого дает красные линии

на белом фоне. Изменение цвета создает иллюзию красных ромбов (белые линии на красном фоне). После того как вы установили свойства Color и PatternColor, вы можете установить для свойства Pattern желаемый узор. В таблице 4 перечислены константы шаблона.

В следующем примере кода для PlotArea задается красная сетка на белом фоне:

```
#DEFINE xlPatternGrid 15
```

```
&& Белый
```

```
oExcel.ActiveChart.PlotArea.Interior.Color = RGB(255,255,255)
```

```
&& Красный
```

```
oExcel.ActiveChart.PlotArea.Interior.PatternColor = RGB(255,0,0)
```

```
oExcel.ActiveChart.PlotArea.Interior.Pattern = xlPatternGrid
```

Это, вероятно, не получит никаких визуальных наград, но эффективно иллюстрирует формирование узора объекта Interior.

Таблица 4. Константы шаблона Excel. Их можно использовать для создания узоров на многих объектах. Однако будьте осторожны и сведите количество рисунков к минимуму, чтобы избежать визуального эффекта «записки о выкупе».

Константа шаблона	Значение	Константа шаблона	Значение
xlPatternAutomatic	-4105	xlPatternHorizontal	-4128
xlPatternChecker	9	xlPatternLightDown	13
xlPatternCrissCross	16	xlPatternLightHorizontal	11
xlPatternDown	-4121	xlPatternLightUp	14
xlPatternGray16	17	xlPatternLightVertical	12
xlPatternGray25	-4124	xlPatternNone	-4142
xlPatternGray50	-4125	xlPatternSemiGray75	10
xlPatternGray75	-4126	xlPatternSolid	1
xlPatternGray8	18	xlPatternUp	-4162
xlPatternGrid	15	xlPatternVertical	-4166

Объект Font

Если вы читали эту книгу последовательно, то к настоящему времени вы уже хорошо знакомы с объектом Font. Это обсуждается в главе «Основы Word» (глава 4) и еще раз в главе «Основы Excel» (глава 7).

Здесь мы рассмотрим основные моменты и отправим вас к теме «Шрифты» в главе 7 для получения подробной информации (в таблице 1 главы 7 перечислены часто используемые свойства шрифтов в Excel).

Если объект содержит объект Font, свойства шрифта наследуются всеми дочерними объектами. Например, когда вы устанавливаете свойства объекта Font ChartArea, все объекты, содержащиеся в ChartArea, наследуют шрифт, если только вы явно не задали свойства шрифта дочернего объекта. Установите свойства шрифта так же, как вы устанавливаете их где-либо еще:

```
oExcel.ActiveChart.ChartArea.Font.Name = "Times New Roman"
```

```
oExcel.ActiveChart.ChartArea.Font.Bold = .T.
```

```
oExcel.ActiveChart.Legend.Font.Name = "Arial"  
oExcel.ActiveChart.Legend.Font.Bold = .F.
```

В этом примере весь текст на диаграмме выделяется полужирным шрифтом Times New Roman, за исключением текста в легенде, для которого выделен нежирный шрифт Arial.

Объект PlotArea

Объект PlotArea является дочерним по отношению к объекту Chart и представляет область, ограниченную осями. Вы можете изменить размер этой области, увеличив или уменьшив ее (и ограничивающие ее оси). PlotArea автоматически оптимизируется для этой диаграммы, и обычно ее не нужно изменять. Но если вы пытаетесь сделать так, чтобы области графика двух или более диаграмм совпадали по размеру, вам придется поиграть с размером PlotArea.

Объект PlotArea имеет стандартные свойства Top, Left, Width и Height. Он также имеет свойства InsideTop, InsideLeft, InsideWidth и InsideHeight. Объект PlotArea фактически имеет две прямоугольные области. Внутренняя область – это область, которая по умолчанию окрашена в серый цвет и ограничена снизу осью категорий, по бокам – осями значений, а сверху – максимальным значением. Вся область графика – это область, ограниченная прямоугольником от самой левой и самой верхней точки текста основной оси значений до самой правой и самой нижней точки вторичной оси значений и текста оси категорий. Ладно, здесь картинка стоит тысячи слов. Если вы в интерактивном режиме выберете PlotArea, а затем попытаетесь переместить ее, Excel отметит обе области пунктирными линиями при ее перемещении, как показано на рисунке 9.

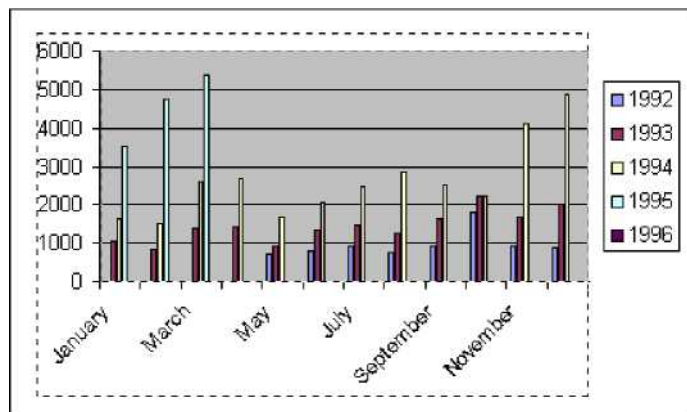


Рисунок 9. Перемещение PlotArea, которое показывает границы внешней и внутренней областей. Внешние области контролируются свойствами Top, Left, Height и Width, а внутренние области – свойствами InsideTop, InsideLeft, InsideHeight и InsideWidth. Свойства внутренней области доступны только для чтения.

Единственная проблема заключается в том, что свойства Inside доступны только для чтения. Чтобы изменить любое из значений Inside, вам необходимо увеличивать или уменьшать свойства Height или Width до тех пор, пока значения InsideHeight и/или InsideWidth не станут подходящими.

К счастью, размер области графика по умолчанию подходит для большинства диаграмм, и вам не придется с ним возиться. Что вы хотите изменить, так это свойства Interior и Border, которые работают так же, как их аналоги ChartArea. См. разделы «Объект Interior» и «Объект Border» ранее в этой главе.

Ряд данных

Объекты ряда данных хранят ссылки на данные для построения графика, а также многочисленные свойства для каждого ряда. Каждый объект Series хранится в объекте коллекции SeriesCollection. Прежде чем мы перейдем к самому интересному (поиграем с цветами и другими функциями форматирования), нам нужно понять, как объект Series хранит и управляет свойствами данных и объектами.

Как хранятся данные и метки

Когда объекты Series задаются с помощью ChartWizard или метода SetSourceData, каждый столбец или строка (в зависимости от параметра nPlotBy) становится объектом Series. Вы также можете добавить ряд (либо в новую диаграмму, либо в существующую диаграмму, даже если она была создана с помощью ChartWizard) с помощью метода Add SeriesCollection.

Синтаксис следующий:

```
oExcel.ActiveChart.SeriesCollection.Add( oSourceRange [, nPlotBy]
    [, lSeriesLabels] [, lCategoryLabels] [, lReplaceCategories] )
```

oSourceRange	Object	Объект Range с новыми данными для добавления. Вы можете добавить несколько рядов одновременно, если исходный диапазон содержит несколько рядов.
nPlotBy	Numeric	Указывает, хранятся ли ряды данных в строках или столбцах. Используйте одну из двух констант: xlRows 1 xlColumns 2 По умолчанию используется xlColumns.
lSeriesLabels	Logical	Указывает, содержит ли первая строка или столбец метки рядов. Если этот аргумент опущен, Excel пытается определить его на основе содержимого первой строки или столбца – предполагается, что меткой являются любые нечисловые данные. Лично мы предпочитаем передавать этот параметр.
lCategoryLabels	Logical	Указывает, содержит ли первая строка или столбец метки категорий. Если этот аргумент опущен, Excel пытается определить его на основе содержимого первой строки или столбца. Символьные данные и данные даты становятся метками, а числовые данные считаются данными для построения графика.
lReplaceCategories	Logical	Указывает, следует ли заменить метки категорий, которые уже существуют на диаграмме, метками категорий в исходном диапазоне. По умолчанию – False.

Например, начиная с данных на рисунке 4, если мы добавим столбец данных, представляющий 1997 год, в столбец G, мы можем обновить диаграмму следующим образом:

```
#DEFINE xlColumns 2
oExcel.ActiveChart.SeriesCollection.Add(oExcel.Sheets[1].Range( ;
    "G1:G13"), xlColumns, .T., .F., .F.)
```

Чтобы использовать метод Add вместо метода ChartWizard для помещения данных в диаграмму, добавленную с помощью ChartsObjects.Add, используйте следующее:

```
oExcel.ActiveChart.SeriesCollection.Add( ;  
    oExcel.Sheets[1].Range("A1:F13") )
```

Часть данных каждого ряда хранится в свойстве Values, из которого вы можете изменить диапазон рядов. В VFP вам нужно установить его в объект Range; из VB вы можете установить его либо как объект Range, либо как массив точек. Когда вы читаете это свойство Values из VFP, вы получаете содержимое первой ячейки, а не объект Range или строку символов, представляющую объект Range. По всей вероятности, это связано с несовместимостью массивов VFP и VB.

Свойство XValues SeriesCollection хранит диапазон, используемый для значений оси X (категории). Однако они не сохраняются в нормализованном виде. *Каждый* объект Series в коллекции хранит значение XValues. Но, похоже, используется только значение первого объекта в коллекции (oChart.SeriesCollection[1]). Как и для свойства Values, XValues имеет значение Range (из VFP; VB также допускает массив точек), но возвращает только первую ячейку.

Свойство Name SeriesCollection хранит значение метки серии. Вы устанавливаете для этого диапазон (обычно представляющий одну ячейку) или строку символов. Это очень удобная команда, поскольку вы можете отформатировать диаграмму, придав ей читаемые имена, а данные в электронной таблице будут сокращены, чтобы разместиться в столбцах.

Форматирование Series

Теперь мы переходим к самому интересному. Сложности хранения данных бледнеют по сравнению с манипулированием их внешним видом (по крайней мере, по нашему мнению).

Два свойства уже знакомы: ChartType и BarShape. Свойства ChartType и BarShape объекта Chart задают форматирование для всей диаграммы, а свойства объекта Series изменяют формат только для этого конкретного ряда. В следующем примере показано, как изменить свойства для каждого ряда. Мы настроим несколько разных рядов, чтобы можно было работать над некоторыми другими свойствами, влияющими на определенные типы диаграмм.

```
#DEFINE xlArea          1  
#DEFINE xlLineMarkers 65
```

```
oExcel.ActiveChart.SeriesCollection[1].ChartType = xlArea  
oExcel.ActiveChart.SeriesCollection[4].ChartType = xlLineMarkers  
oExcel.ActiveChart.SeriesCollection[5].ChartType = xlLineMarkers
```

Рисунок 10 показывает результаты кода. Обычно эту диаграмму не форматируют с использованием разных типов диаграмм, поскольку каждый ряд представляет одни и те же данные. Вы можете использовать разные типы диаграмм, если отображаете прибыль, показатели продаж и прогнозируемые цели. Но мы поработаем над манипулированием этими данными, чтобы проиллюстрировать концепции рядов.

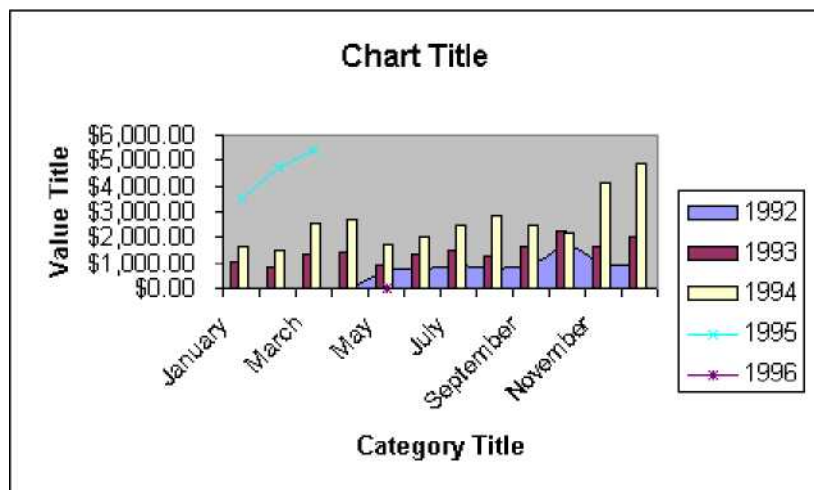


Рисунок 10. Каждый ряд может представлять собой отдельный тип диаграммы. Нет смысла форматировать этот конкретный набор данных таким образом, но мы можем использовать данные для иллюстрации концепций.

Есть еще два объекта, которые к настоящему моменту вам должны быть знакомы. Объекты `Border` и `Interior` форматируют каждый ряд. Линейные объекты, такие как линия, используют свойства `Border` для форматирования самой линии; объект `Interior` недоступен для линий. Другие объекты используют свойства `Border` и `Interior` для установки границы (по умолчанию она черная), а также цвета и узора панели. Подробности см. в разделах «Объект `Border`» и «Объект `Interior`» ранее в этой главе.

Метки данных

Метки данных – это текстовые объекты, которые отмечают каждую точку ряда. Эти свойства и методы доступны объектам `Chart`, `Series`, and `Point`. Включите их с помощью свойства `HasDataLabels`:

```
oExcel.ActiveChart.SeriesCollection[3].HasDataLabels = .T.
```

Если `HasDataLabels` имеет значение `.T.`, Excel помечает каждую точку значением точки данных. Excel отлично справляется с центрированием метки данных над точкой и установкой размера шрифта на удобочитаемый. Однако он не пытается ничего сделать с перекрывающимися надписями (если бы это было сделано, то надписи не были бы центрированы на вычисленном расстоянии над точкой).

Включение меток дает нам очень загроможденный вид, как показано на **рисунке 11**. Существует несколько способов обойти эту проблему, но ни один из них не является надежным в автоматизированной среде, поскольку сложно запросить метки данных, чтобы определить, перекрываются ли они.

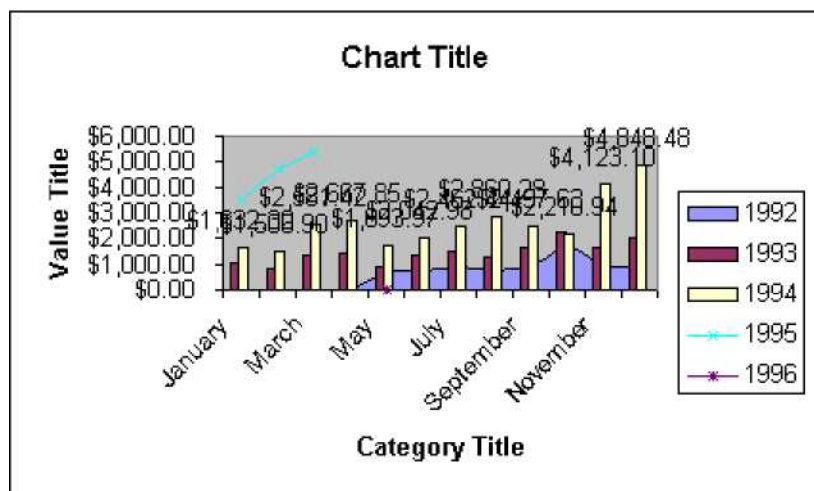


Рисунок 11. Включение меток данных может создать очень загроможденный вид. Есть несколько обходных путей: уменьшите шрифт меток или включите их только для выбранных точек.

Один из способов – изменить размер шрифта. Шрифты меньшего размера не будут так сильно перекрываться. Проблема здесь в том, что вы можете сделать его настолько маленьким, что размер шрифта станет нечитаемым. Свойство `DataLabels` позволяет обращаться ко всей коллекции меток данных или выбирать отдельные метки для форматирования. Вот пример изменения свойств шрифта для всего ряда меток данных:

```
WITH oExcel.ActiveChart.SeriesCollection[3].DataLabels.Font
    .Name = "Times New Roman"
    .Size = 6
ENDWITH
```

Выбрать одну метку из коллекции так же просто, как обратиться к ней по индексу. Например, это устанавливает для третьей метки данных размер шрифта шесть:

```
oExcel.ActiveChart.SeriesCollection[3].DataLabels[3].Font.Size = 6
```

Существует больше вариантов маркировки, чем просто значение в точке. Вы можете использовать метод `ApplyDataLabels` объекта `SeriesCollection`. Он имеет четыре параметра, но наиболее важным является первый. Вы можете передать одно из значений, приведенных в таблице 5, чтобы задать текст метки.

Таблица 5. Значения для метода `ApplyDataLabels`. Обратите внимание, что некоторые метки доступны только для определенных типов диаграмм.

Константа шаблона	Значение	Примечание
<code>xlDataLabelsShowNone</code>	-4142	Никаких меток с данными.
<code>xlDataLabelsShowValue</code>	2	Показывает значение.
<code>xlDataLabelsShowPercent</code>	3	Только для круговых и кольцевых диаграмм.
<code>xlDataLabelsShowLabel</code>	4	Показывает метку категории.
<code>xlDataLabelsShowLabelAndPercent</code>	5	Только для круговых и кольцевых диаграмм.
<code>xlDataLabelsShowBubbleSizes</code>	6	Только для пузырьковых диаграмм.

На тот случай, если этих параметров недостаточно, вы можете установить текст отдельных меток, задав для свойства `Text` объекта `DataLabel` любое желаемое значение:

```
oExcel.ActiveChart.SeriesCollection[3].DataLabels[1].Text = "Тест"
```

Предыдущий код устанавливает метку «Тест». Установка текстового значения в пустую строку фактически удаляет метку.

Полосы ошибок

Полосы ошибок показывают потенциальные суммы ошибок для каждого значения в ряду. Они доступны для двумерных областных, гистограмм, столбчатых, линейных, XY (точечных) и пузырьковых диаграмм. Для установки значений используется метод `ErrorBar` объекта `Series`.

```
.ActiveChart.Series[n].ErrorBar( [nDirection] [, nInclude]
    [, nType] [, nErrorAmount] [,nCustomNegAmount] )
```

nDirection	Numeric	Направление всегда вдоль оси Y, за исключением точечных диаграмм, где для указания направления полос погрешностей можно использовать xlX (-4168) или xlY (1).										
nInclude	Numeric	Значение, указывающее, какие половины шкалы ошибок следует включить. Выберите один из следующих вариантов: <table><tr><td>xlErrorBarIncludePlusValues</td><td>2</td></tr><tr><td>xlErrorBarIncludeMinusValues</td><td>3</td></tr><tr><td>xlErrorBarIncludeNone</td><td>-4142</td></tr><tr><td>xlErrorBarIncludeBoth</td><td>1</td></tr></table> По умолчанию используется xlErrorBarIncludeBoth.	xlErrorBarIncludePlusValues	2	xlErrorBarIncludeMinusValues	3	xlErrorBarIncludeNone	-4142	xlErrorBarIncludeBoth	1		
xlErrorBarIncludePlusValues	2											
xlErrorBarIncludeMinusValues	3											
xlErrorBarIncludeNone	-4142											
xlErrorBarIncludeBoth	1											
nType	Numeric	Значение, указывающее, как рассчитываются значения полосы погрешностей. Выберите один из следующих вариантов: <table><tr><td>xlErrorBarTypeFixedValue</td><td>1</td></tr><tr><td>xlErrorBarTypePercent</td><td>2</td></tr><tr><td>xlErrorBarTypeStDev</td><td>-4155</td></tr><tr><td>xlErrorBarTypeStError</td><td>4</td></tr><tr><td>xlErrorBarTypeCustom</td><td>-4114</td></tr></table> По умолчанию используется xlErrorBarTypeFixedValue.	xlErrorBarTypeFixedValue	1	xlErrorBarTypePercent	2	xlErrorBarTypeStDev	-4155	xlErrorBarTypeStError	4	xlErrorBarTypeCustom	-4114
xlErrorBarTypeFixedValue	1											
xlErrorBarTypePercent	2											
xlErrorBarTypeStDev	-4155											
xlErrorBarTypeStError	4											
xlErrorBarTypeCustom	-4114											
nErrorAmount	Numeric	Величина ошибки в единицах, выраженная в nType. Если параметр nType имеет значение xlErrorBarTypeCustom, это значение положительной ошибки в виде фиксированного значения.										
nCustomNegAmount	Numeric	Используется только в том случае, если параметр nType имеет значение xlErrorBarTypeCustom. Это значение отрицательной ошибки, как фиксированное значение.										

Свойство ErrorBars (обратите внимание, что метод имеет единственное число, а свойство – множественное число) ссылается на объект ErrorBar со свойствами для дополнительного форматирования. Первое – это свойство Border; да, он ссылается на тот же объект Border, чтобы задать форматирование строк панели ошибок (см. раздел «Объект Border» ранее в этой главе).

Вторым свойством объекта ErrorBar является свойство EndStyle, которое определяет, имеют ли полосы погрешностей границу (перпендикулярную линию в конце полосы ошибок) или нет. Имеет две настройки: xlCap (1) и xlNoCap (2).

Форматирование маркеров данных (линейные, лепестковые и точечные диаграммы)

Линейные, лепестковые и точечные диаграммы могут использовать определенные символы для нанесения точек, которые на жаргоне Excel называются *маркерами данных*. Когда вы изначально выбираете формат с маркерами данных, вы получаете набор маркеров по умолчанию. В первом ряду получается шестиконечная звезда (например, «х» с дополнительной вертикальной линией), во втором ряду – «х», в третьем – треугольник, в четвертом – квадрат, в пятом – ромб и так далее. Поскольку линии отличаются только цветом, некоторым это может показаться слишком перегруженным. Лично нам кажется, что первые два символа немного напоминают колючую проволоку; мы предпочитаем другие символы.

Объект Series (а также объект Points) использует свойство MarkerStyle для установки формы маркера. В Таблице 6 показан список всех констант маркеров, а также их значения и формы.

Таблица 6. Доступные стили маркеров. Маркеры используются на двумерных линейных диаграммах и точечных диаграммах.

Константа MarkerStyle	Значение	Картинка
xlMarkerStyleNone	-4142	
xlMarkerStyleAutomatic	-4105	варьируется
xlMarkerStyleCircle	8	●
xlMarkerStyleDash	-4115	-
xlMarkerStyleDiamond	2	◆
xlMarkerStyleDot	-4118	▪
xlMarkerStylePlus	9	+
xlMarkerStyleSquare	1	■
xlMarkerStyleStar	5	*
xlMarkerStyleTriangle	3	▲
xlMarkerStyleX	-4168	×

Вы можете изменить размер маркера с помощью свойства MarkerSize. Нам нравится тот факт, что это свойство устанавливает размер в пунктах, а не на основе ряда констант!

Вы также можете изменить цвет маркера. Вы можете отдельно установить цвета переднего плана и фона, что позволит вам создать множество эффектов. MarkerForegroundColor управляет цветом границы, а MarkerBackgroundColor – цветом заливки. Установите для каждого значение цвета RGB. Установка этих значений не меняет цвет линий, соединяющих маркеры (если они есть). Например, давайте заменим маркер данных 1995 года на открытый темно-синий ромб, а одиночный маркер данных 1996 года – на красный квадрат.

```
#DEFINE xlMarkerStyleDiamond 2
#DEFINE xlMarkerStyleSquare 1
#DEFINE ColorRed RGB(255, 0, 0)
#DEFINE ColorWhite RGB(255,255,255)
```

```
#DEFINE ColorDkBlue RGB( 0, 0,128)
```

* Отформатируйте данные 1995 года в открытый темно-синий ромб:

```
WITH oExcel.ActiveChart.SeriesCollection[4]
    .MarkerStyle = xlMarkerStyleDiamond
    .MarkerForegroundColor = ColorDkBlue
    .MarkerBackgroundColor = ColorWhite
ENDWITH
```

* Отформатируйте данные 1996 года в виде красного квадрата на белом фоне:

```
WITH oExcel.ActiveChart.SeriesCollection[5]
    .MarkerStyle = xlMarkerStyleSquare
    .MarkerForegroundColor = ColorRed
ENDWITH
```

Рисунок 12 показывает результаты форматирования маркеров. Хотя книга и черно-белая, можно разглядеть, что границы ромбов темнее линии, а закрашенный квадрат виден и имеет другой цвет, чем линия.

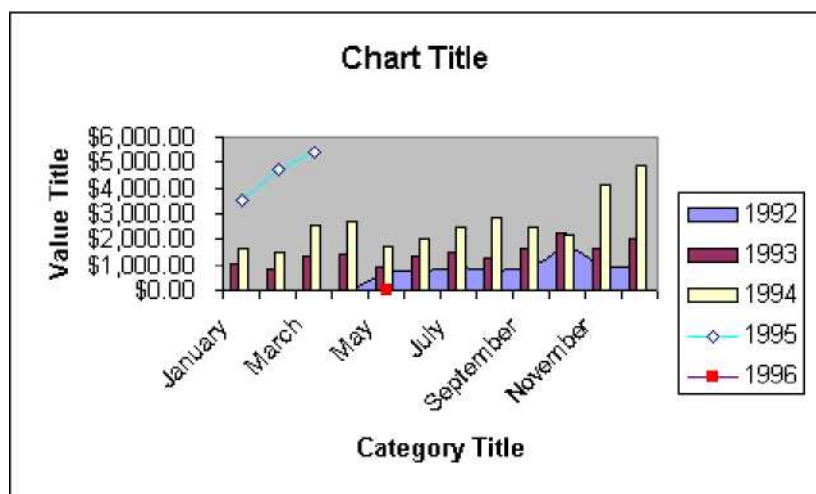


Рисунок 12. Результаты изменения маркеров данных. Обратите внимание, что маркеры можно раскрашивать отдельно от линии.

Сглаживание линий на графиках

По умолчанию линии на диаграмме представляют собой прямые линии, соединяющие каждую точку. Вы можете использовать свойство `Smooth` объекта `Series`, чтобы применить сглаживание кривой. Установите для этого свойства значение `True`, чтобы сгладить линию, или значение `False`, чтобы линии оставались прямыми. Это свойство применяется только к линейным диаграммам и точечным диаграммам.

Форматирование круговых диаграмм

Круговые диаграммы имеют уникальные особенности. В этом примере мы создаем круговую диаграмму рядом с диаграммой, созданной для предыдущих примеров, с помощью метода `ChartWizard`.

```
#DEFINE xlPie 5
#DEFINE autoPieNoLabels 1
#DEFINE xlColumns 2
#DEFINE autoOneSeriesLabel 1
```

```
#DEFINE autoOneCategoryLabel 1

oRange = oExcel.Sheets[1].Range("CategoryNames; Year1994")

oChart = oExcel.Sheets[1].ChartObjects.Add(372.7, 173.2, 319.5, 190.5)
oChart.Activate()
oExcel.ActiveChart.ChartWizard(oRange, xlPie, autoPieNoLabels, ;
    xlColumns, autoOneSeriesLabel, autoOneCategoryLabel)
```

Полученный график показан на **рисунке 13**. Это немного невзрачно. Легенда тоже надоедает, так как часть последней записи отрезана. К счастью, Excel предоставляет нам ряд возможностей форматирования.

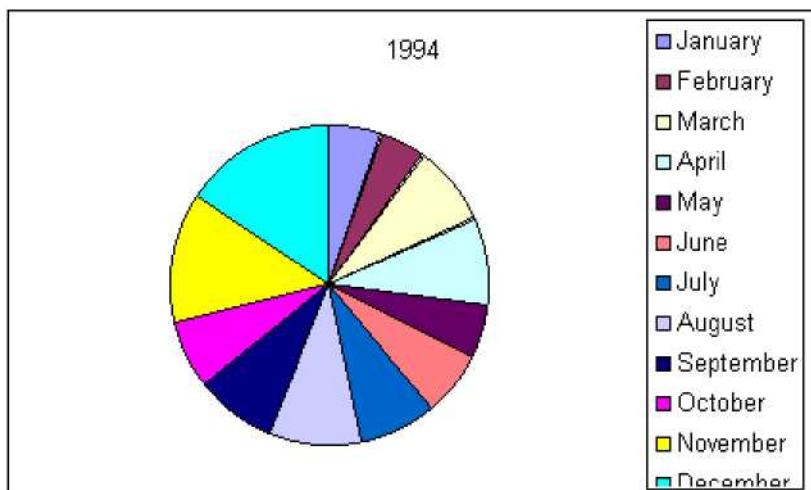


Рисунок 13. Круговая диаграмма по умолчанию, созданная ChartWizard. В Excel есть несколько вариантов форматирования, позволяющих украсить эту диаграмму.

Мы уже рассмотрели добавление меток данных с помощью метода ApplyDataLabels объекта Series (см. раздел «Метки данных» ранее в этой главе). Давайте посмотрим на значения, доступные только для круговых диаграмм (и кольцевых диаграмм): xlDataLabelsShowPercent (3) и xlDataLabelsShowLabelAndPercent (5). Отображение и метки, и процента может привести к перекрытию меток, поскольку в этом случае секторов слишком много, поэтому отображение только процента будет хорошим выбором.

```
#DEFINE xlDataLabelsShowPercent 3
oExcel.ActiveChart.SeriesCollection[1].ApplyDataLabels( ;
    xlDataLabelsShowPercent)
```

Другая возможность – эффект разрезания, когда один (или все) сектора выходят из центра. Свойство Explosion применяется к объекту Series (где все фрагменты расчленены) и объекту Points (где расчленяется только этот фрагмент). Значение свойства Explosion – это число, обозначающее расстояние, на которое срез перемещается от центра. Поэкспериментируйте немного, чтобы определить наилучшее значение, поскольку оно является относительной величиной. Один – совсем немного, что-то в диапазоне 5-20 выглядит хорошо, а 50 разделяет кусочки, возможно, слишком сильно (на этой конкретной диаграмме ваш опыт может отличаться).

```
oExcel.ActiveChart.SeriesCollection[1].Explosion = 15
```

Результаты форматирования меток и взрыва показаны на **рисунке 14**.

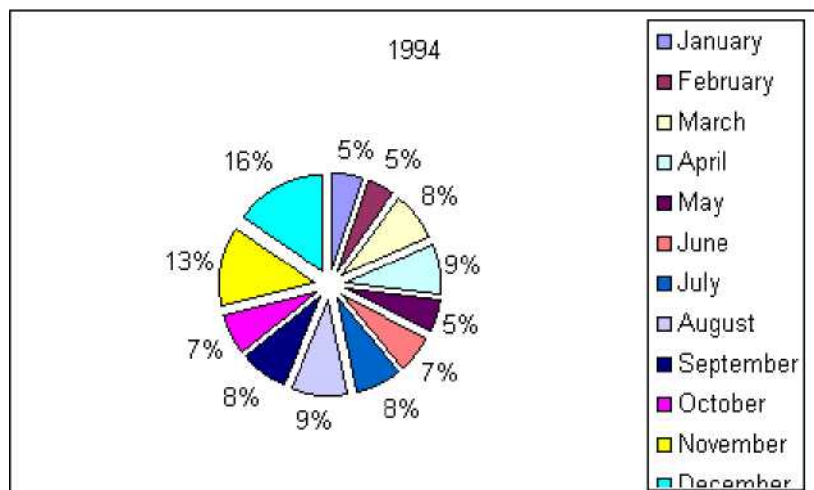


Рисунок 14. Добавлена круговая диаграмма с некоторым форматированием. Показываются процентные метки, и вы также можете добавить метку категории (хотя она будет перекрываться). Расчленение всех фрагментов позволяет по-новому взглянуть на круговую диаграмму.

Форматирование отдельных точек

В предыдущих разделах вы видели ссылку на возможность форматирования отдельных точек. Почти все функции форматирования, доступные для объекта Series, доступны для одной точки. Доступ к точке осуществляется с помощью метода Points объекта Series. Вы можете использовать эту функцию для изменения цвета и/или стиля маркера, полосы, столбца или среза, чтобы привлечь внимание к этой точке данных. Вы также можете добавить метки данных к одной или двум точкам, чтобы снова привлечь внимание к этой точке. Вы можете вручную установить значения меток данных, что особенно хорошо работает, если вы хотите добавить метки, которые не являются частью ваших данных в электронной таблице.

В примере круговой диаграммы, возможно, клиент хочет выделить фрагмент декабря меткой категории и разобрать его (но не остальное). В следующем примере сначала удаляется разворачивание из ряда, а затем устанавливается форматирование для декабрьского фрагмента:

```
#DEFINE xlDataLabelsShowLabelAndPercent 5

* Сбросьте взрыв ряда на 0
oExcel.ActiveChart.SeriesCollection[1].Explosion = 0

* Форматировать одну точку
WITH oExcel.ActiveChart.SeriesCollection[1].Points[12]
    .ApplyDataLabels(xlDataLabelsShowLabelAndPercent)
    .Explosion = 15
ENDWITH
```

Рисунок 15 показывает результаты форматирования одной точки.

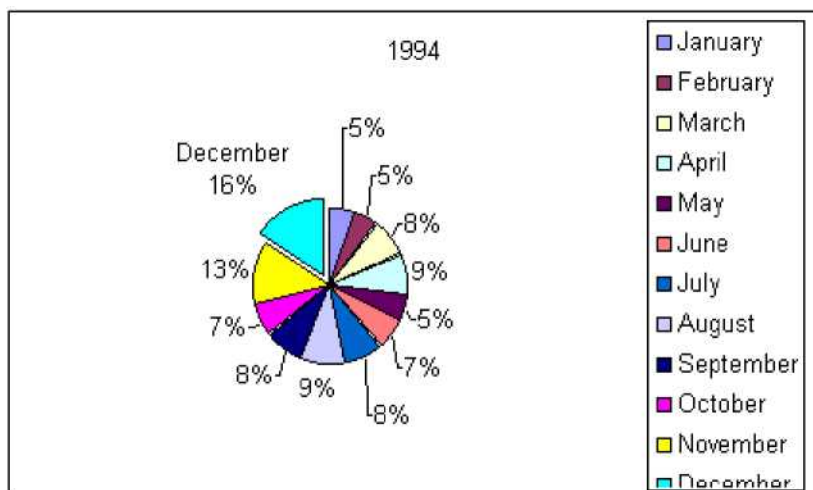


Рисунок 15. Вы можете отформатировать одну точку. Обратите внимание, что декабрьский срез расчленен и имеет другой стиль меток данных.

Объект легенды

Легенда круговой диаграммы беспокоит нас с самого первого примера. Доступ к объекту Legend осуществляется через свойство Legend объекта Chart. ChartObject также имеет свойство HasLegend; если для него установлено значение False, свойство Legend (и, следовательно, объект Legend) недоступно. Чтобы быстро отключить легенду, просто установите для HasLegend значение False:

```
oExcel.ActiveChart.HasLegend = .F.
```

Это работает, но у него есть неприятная привычка не пересчитывать PlotArea, чтобы использовать пространство, ранее занимаемое легендой. Вы также можете использовать метод ChartWizard, чтобы отключить легенду, повторно указав параметр ChartFormat (второй) и параметр HasLegend (седьмой), как показано ниже:

```
#DEFINE xLPie 5
oExcel.ActiveChart.ChartWizard(,xLPie,,,,.F.)
```

Обычно это приводит к перерасчету площади участка. Однако теперь легенды нет, и пользователю предстоит выяснить, что означают все эти фрагменты. Возможно, нам стоит вернуть легенду и придумать, как сделать ее лучше. Установка для HasLegend значения True пересчитывает область графика, поэтому нам не нужно беспокоиться о запоминании параметров ChartWizard.

```
oExcel.ActiveChart.HasLegend = .T.
```

Свойства шрифта можно изменить с помощью свойства Font Legend, которое обращается к объекту Font (описано в предыдущем разделе этой главы «Объект Font»). Вы можете немного уменьшить шрифт, а затем подогнать его по размеру.

```
oExcel.ActiveChart.Legend.Font.Size = ;
oExcel.ActiveChart.Legend.Font.Size - 2
```

Рисунок 16 показывает, что изменение размера шрифта сработало. Вы также можете управлять названием шрифта, атрибутами полужирного и курсива и т. д. Это тот же объект Font, который используется во всей модели Excel.

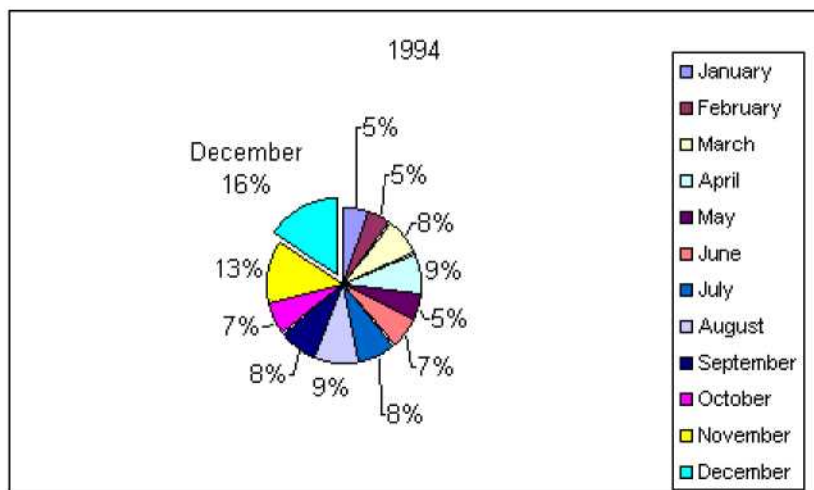


Рисунок 16. Эффекты глобального изменения размера шрифта легенды. Уменьшив размер, мы сделали его подходящим. Объект Legend столь же надежен, как и другие объекты диаграммы, и имеет множество свойств для изменения его формата.

Что еще можно сделать с Legend? Вы можете установить его свойства Border и Interior (см. разделы «Объект Border» и «Объект Interior» ранее в этой главе). Вы можете манипулировать свойствами Top и Left, чтобы перемещать его (Height и Width рассчитываются автоматически и поэтому доступны только для чтения). Более простой способ перемещения – использовать свойство Position, которое принимает одно из следующих значений: xlLegendPositionBottom (-4107), xlLegendPositionCorner (2), xlLegendPositionLeft (-4131), xlLegendPositionRight (-4152) или xlLegendPositionTop (-4160). При этом также пересчитывается высота и ширина, позволяя легенде растягиваться вдоль нижней части.

Вы также можете отформатировать свойства шрифта каждой записи легенды отдельно, обратившись к свойству LegendEntries, которое ссылается на коллекцию LegendEntries объектов LegendEntry. Вы не можете изменить текст (это можно сделать через объект Series или ячейку с меткой), но вы можете получить доступ к объекту шрифта, как показано ниже, который форматирует запись легенды декабря как жирный шрифт:

```
oExcel.ActiveChart.Legend.LegendEntries[12].Font.Bold = .T.
```

Каждый объект LegendEntry имеет объект LegendKey, который представляет собой небольшой образец, идентифицирующий ряд. Он имеет объект Border и объект Interior, а также другие свойства форматирования, используемые объектом Series. Рекомендуем задать форматирование в объекте Series; однако есть одно исключение. Мы столкнулись с проблемой форматирования объекта Series с совершенно пустым диапазоном данных. Это может произойти, если вы создаете данные для стандартной диаграммы, а данные для ряда не появляются, например, если вы строите график ежемесячных продаж по клиентам для каждой категории продуктов, а клиент не заказывает товары, скажем, в категориях «Напитки» или «Кондитерские изделия». Доступ к объекту Series дает ошибку, поскольку диапазон данных пуст. Вы можете безопасно отформатировать объект LegendKey, чтобы настроить внешний вид ключа легенды, чтобы легенда выглядела так же, как другие графики.

Оси координат

Метод Axes объекта Chart возвращает объект Axis из коллекции Axes. Каждый объект Axis на диаграмме содержится в коллекции Axes. Метод Axes принимает два

параметра: тип оси и группу осей. Типом оси является одно из следующих значений: xlCategory (1), xlSeriesAxis (3) или xlValue (2). Группа осей может быть либо xlPrimary (1), либо xlSecondary (2). Вторичные оси доступны только на двумерных диаграммах и только в том случае, если для свойства AxisGroup объекта Series установлено значение 2. Вторичная ось позволяет отображать вместе два отдельных диапазона данных. Например, основная ось может представлять собой стоимость продаж, а вторичная ось отображает процент от общей суммы. В таблице 7 показаны свойства, доступные для каждого объекта Axis.

Таблица 7. Свойства объекта Axis. Эти свойства обеспечивают большую гибкость форматирования осей.

Свойство	Тип	Описание
AxisBetweenCategories	Logical	Применяется только к оси категорий. Истинно, если ось значений пересекает ось категорий между категориями (лучше всего подходит для гистограмм); Ложь, если она пересекает категорию (хорошо для линий).
Crosses	Numeric	Определяет точку, где другие оси пересекают эту ось. Может иметь первое значение, используя xlMinimum (4), или последнее значение, используя xlMaximum (2).
CrossesAt	Numeric	Применяется только к оси значений. Целое число, представляющее относительное значение на оси значений, где пересекается ось категорий.
Border	Object	Ссылается на объект Border, который отвечает за форматирование линии оси. См. раздел «Объект Border» ранее в этой главе.
ScaleType	Numeric	Применяется только к оси значений. Значениями являются либо xlLinear (-4132), либо xlLogarithmic (-4133).
HasMajorGridlines, HasMinorGridlines	Logical	Истинно, если ось имеет основные или второстепенные линии сетки соответственно. Только основные оси могут иметь линии сетки.
MajorGridlines, MinorGridlines	Object	Если соответствующее свойство HasMajorGridlines или HasMinorGridlines имеет значение True, это свойство ссылается на объект Gridlines. См. раздел «Линии сетки» далее в этой главе.
MaximumScaleIsAuto, MinimumScaleIsAuto	Logical	Применяется только к оси значений. Логическое свойство, указывающее, рассчитывается ли Excel число минимальных и максимальных значений. Если значение равно False, установите свойства MaximumScale или MinimalScale.
MaximumScale, MinimumScale	Numeric	Применяется только к оси значений. Если MaximumScaleIsAuto или MinimalScaleIsAuto имеет значение False, эти свойства содержат значения наибольшего/наименьшего значения на оси.

Свойство	Тип	Описание
MajorUnitIsAuto, MinorUnitIsAuto	Logical	Применяется только к оси значений. Указывает, рассчитывается ли число единиц между делениями в Excel. Если значение False, установите свойства MajorUnit/MinorUnit.
MajorUnit, MinorUnit	Numeric	Применяется только к оси значений. Если MajorAxisIsAuto или MinorAxisIsAuto имеет значение False, эти свойства содержат значение между основными и второстепенными делениями; в противном случае это свойство содержит .T.
TickmarkSpacing	Numeric	Применяется только к осям рядов и категорий. Определяет, сколько категорий или рядов находится между делениями. Используйте свойства MajorUnit или MinorUnit, чтобы задать ось значений.
MajorTickMark, MinorTickMark	Numeric	Указывает расположение отметки. Используйте одно из следующих: xlTickMarkNone -4142 xlTickMarkInside 2 xlTickMarkOutside 3 xlTickMarkCross 4
TickLabels	Object	Ссылается на объект TickLabels. В этом объекте задаются свойства текста для отметки. См. следующий раздел «Объект TickLabels».
TickLabelPosition	Numeric	Описывает положение меток на оси. Значения: xlTickLabelPositionNone -4142 Подавляет метки. xlTickLabelPositionLow -4134 Ближайшая к оси. xlTickLabelPositionHigh -4127 На другой стороне PlotArea. xlTickLabelPositionNextTo Axis 4 Ближайшая к оси.
TickLabelSpacing	Numeric	Определяет количество категорий или рядов между метками делений. Невозможно установить на оси значений.

Объект TickLabels

Объект TickLabels управляет форматом меток на оси. Это дает вам почти полный контроль над метками. В таблице 8 приведены важные свойства объекта TickLabels.

Таблица 8. Свойства объекта TickLabels. TickLabels – это текст, который отображается вдоль оси.

Свойство	Тип	Описание
Alignment	Numeric	Значение, указывающее, как метка совпадает с осью. Выберите xlHAlignCenter (-4108), xlHAlignLeft (-4131) или xlHAlignRight (-4152).
Font	Object	Свойство, которое обращается к объекту Font для форматирования текста по оси. См. раздел «Объект Font» ранее в этой главе.
NumberFormat	Character	Строка кодов, которая используется для форматирования чисел, во многом аналогично свойству VFP InputMask. Дополнительную информацию см. в разделе «Форматирование значений» главы 7.
Orientation	Numeric	Целочисленное значение, представляющее ориентацию текста в диапазоне от -90 до 90. Альтернативно можно использовать константу: xlTickLabelOrientationAutomatic -4105 xlTickLabelOrientationDownward -4170 xlTickLabelOrientationHorizontal -4128 xlTickLabelOrientationUpward -4171 xlTickLabelOrientationVertical -4166

Линии сетки

Объект Gridlines доступен только для основных осей. Его единственное соответствующее свойство – это свойство Border, обеспечивающее доступ к объекту Border (см. раздел «Объект Border» ранее в этой главе). Объект Border содержит форматирование линий сетки.

Заголовки

Диаграммы имеют два типа объектов заголовка: объекты ChartTitle и объекты AxisTitle. Их структуры схожи. Доступ к объекту ChartTitle осуществляется через свойство ChartTitle объекта Chart, и он используется только в том случае, если свойство HasTitle объекта Chart имеет значение True. Доступ к AxisTitle осуществляется через свойство AxisTitle объекта Axis, и оно используется только в том случае, если свойство HasTitle объекта Axis имеет значение True. В таблице 9 показаны их свойства.

Таблица 9. Свойства объектов ChartTitle и AxisTitle.

Свойство	Тип	Описание
Border	Object	Ссылается на объект Border. См. раздел «Объект Border» ранее в этой главе. По умолчанию заголовки не имеют границ.
Caption, Text	Character	Эти два идентичных свойства содержат текст заголовка.
Font	Object	Ссылается на объект Font. См. раздел «Объект Font» ранее в этой главе.

Свойство	Тип	Описание
HorizontalAlignment	Numeric	Определяет горизонтальное выравнивание заголовка. Используйте одно из следующих: xlHAlignCenter -4108 xlHAlignLeft -4131 xlHAlignDistributed -4117 xlHAlignRight -4152 xlHAlignJustify -4130
VerticalAlignment	Numeric	Определяет вертикальное выравнивание заголовка. Используйте одно из следующих: xlVAlignBottom -4107 xlVAlignJustify -4130 xlVAlignCenter -4108 xlVAlignTop -4160 xlVAlignDistributed -4117
Left, Top	Numeric	Можно настроить точное расположение заголовка.
Orientation	Numeric	Определяет ориентацию текста в диапазоне от -90 до 90. Или можно использовать константу: xlDownward -4170 xlHorizontal -4128 xlUpward -4171 xlVertical -4166
Shadow	Logical	Указывает, есть ли тень в поле заголовка.
Interior	Object	Ссылается на объект Interior, который управляет цветом поля заголовка. См. раздел «Объект Interior» ранее в этой главе.

9.7 Собираем все вместе



Листинг 2 показывает программу (XLSample3.PRG в файлах для загрузки для разработчиков, доступных на сайте www.hentzenwerke.com), которая использует многие функции, описанные в этой главе. Первая диаграмма представляет собой столбчатую диаграмму, в которой используются данные, полученные в **листинге 1**. Поскольку данные за 1996 год неполны, в диапазоне эти данные отсутствуют. Чтобы проиллюстрировать добавление ряда к графику, столбец «Среднее» создается и заполняется с помощью функции СРЗНАЧ (AVERAGE) в Excel. Затем ряд средних значений форматируется как линейная диаграмма. Каждый из остальных столбцов отформатирован в определенный цвет, оптимизированный для печати (поэтому они хорошо выглядят при печати в этой книге). Метки осей также форматируются. Эта диаграмма показана на **рисунке 17**.

Вторая диаграмма представляет собой трехмерную круговую диаграмму, представляющую собой лист диаграммы, а не встроенную диаграмму. Она использует некоторые свойства 3D-форматирования и изменяет форматирование некоторых заголовков и меток данных, а также цвет секторов круговой диаграммы. В качестве цвета был выбран серый, чтобы он хорошо печатался. Круговая диаграмма показана на **рисунке 18**.

Листинг 2. Графики годовых продаж тасманских торговцев. Этот код охватывает многие темы, рассмотренные в этой главе, такие как создание графиков, работа с несколькими типами диаграмм внутри диаграммы, добавление серий,

форматирование отдельных серий и многое другое.

```
* XLSample3.PRG
* © 2000, Tamar E. Granor and Della Martin
* From: Microsoft Office Automation with Visual FoxPro
* Hentzenwerke Publishing. www.hentzenwerke.com

* Это пример «Собираем все вместе» из главы 9.
* Устанавливает ежемесячные данные о продажах по годам
* для отображения на графике.

* Удалите все существующие ссылки на серверы.
* Это предотвращает потерю памяти оставшимися экземплярами.
RELEASE ALL LIKE o*
```

```
* В демонстрационных целях сделайте определенные объекты
* доступными после выполнения этой программы.
PUBLIC oExcel, oBook, oSheet, oChart
```

```
#DEFINE xlColumn          3
#DEFINE xl3DPie           -4102
#DEFINE xlColumns         2
#DEFINE xlLegendPositionBottom -4107
#DEFINE xlLineStyleNone   -4142
#DEFINE xlCategory        1
#DEFINE xlValue            2
#DEFINE xlPrimary          1
#DEFINE xlFillDefault      0
#DEFINE xlLineMarkers      65
#DEFINE xlMarkerStyleDiamond 2
#DEFINE autoColumnFormat   4
#DEFINE autoPieFormat      7
#DEFINE autoOneSeriesLabel 1
#DEFINE autoOneCategoryLabel 1
#DEFINE autoHasLegend      .T.
#DEFINE autoNotHasLegend   .F.
#DEFINE rgbWhite           RGB(255, 255, 255)
#DEFINE rgbLtGray         RGB(192, 192, 192)
#DEFINE rgbMedGray        RGB(128, 128, 128)
#DEFINE rgbDkGray         RGB( 64,  64,  64)
#DEFINE rgbBlack          RGB(  0,  0,  0)
```

```
* --- Создайте книгу и добавьте данные
DO XLGDataSetup      && Листинг 1
```

```
* Добавьте имена диапазонов
WITH oExcel.Sheets[1]
    .RANGE("A1:A13").NAME = "CategoryNames"
    .RANGE("B1:B13").NAME = "Year1992"
    .RANGE("C1:C13").NAME = "Year1993"
    .RANGE("D1:D13").NAME = "Year1994"
    .RANGE("E1:E13").NAME = "Year1995"
    .RANGE("F1:F13").NAME = "Year1996"
ENDWITH
```

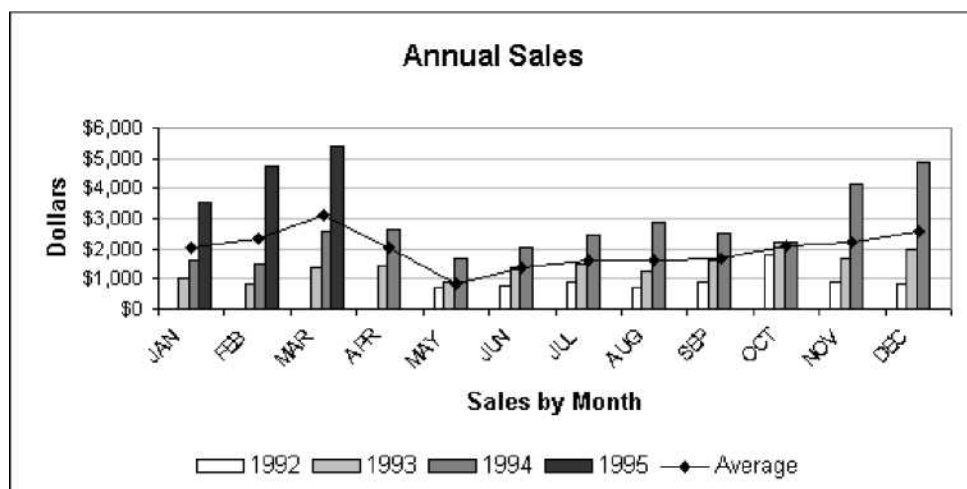


Рисунок 17. График годовых продаж. На этой диаграмме показано форматирование осей с использованием различных типов диаграмм по сериям, перемещением легенды и форматированием каждой серии по отдельности.

* --- Создайте первую диаграмму

```
oChart = oExcel.Sheets[1].ChartObjects.ADD(0, 175, 400, 200)
oChart.ACTIVATE()
```

* Включите только названия категорий и годы до 1995 г.

```
oSourceRange = oExcel.Sheets[1].Range("A1:E13")
oExcel.ActiveChart.ChartWizard(oSourceRange, xlColumn, ;
    autoColumnFormat, xlColumns, autoOneCategoryLabel, ;
    autoOneSeriesLabel, autoHasLegend, ;
    "Годовые продажи", "Продажи по месяцам", "Рубли")
```

```
WITH oExcel.ActiveChart
```

* Переместите легенду вниз и удалите рамку.

```
.Legend.Position = xlLegendPositionBottom
.Legend.Border.LineStyle = xlLineStyleNone
```

* Отформатируйте оси.

* На оси категорий установите размер немного меньше,

* и поместите этикетки под наклоном 45 градусов.

```
.Axes(xlCategory, xlPrimary).TickLabels.Orientation = 45
.Axes(xlCategory, xlPrimary).TickLabels.Font.Size = 8
```

* На оси значений установите размер немного меньше,

* отформатируйте значения в денежном выражении и включите

* «Линии сетки».

```
.Axes(xlValue, xlPrimary).TickLabels.Font.Size = 8
.Axes(xlValue, xlPrimary).TickLabels.NumberFormat = "$#,###; ;$0"
.Axes(xlValue, xlPrimary).HasMajorGridlines = .T.
.Axes(xlValue, xlPrimary).MajorGridlines.Border.Color = rgbLtGray
```

* Сократите названия категорий, чтобы освободить место для

* диаграммы

```
FOR I = 1 TO 12
```



```

    WITH oExcel.ActiveSheet.Range("A" + ALLTRIM(STR(I + 1)))
        .Value = UPPER(LEFT(.Value, 3))
    ENDWITH
ENDFOR

```

```

* Центрировать ось внутри ChartArea.
.Legend.Left = (.ChartArea.Width - .Legend.Width) / 2

```

```

ENDWITH

```

```

* Добавьте столбец средних значений.

```

```

WITH oExcel.Sheets[1]

```

```

    * Вставьте метку ряда.

```

```

    .Range("G1").Value = "Среднее"

```

```

    * Insert the AVERAGE formula into each cell

```

```

    * Вставьте формулу СРЗНАЧ в каждую ячейку.

```

```

    FOR I = 2 TO 13

```

```

        cI = ALLTRIM(STR(I))

```

```

        .Range("G" + cI).Value = "=AVERAGE(B" + cI + ":F" + cI + ")"

```

```

    * .Range("G" + cI).Value = "=СРЗНАЧ(B" + cI + ":F" + cI + ")"

```

```

    ENDFOR

```

```

    * Назовите диапазон

```

```

    .Range("G1:G13").Name = "Average"

```

```

ENDWITH

```

```

WITH oExcel.ActiveChart

```

```

    * Добавьте ряд "Среднее" и отформатируйте его как черную линию

```

```

    * с помощью ромбовидных маркеров (также черных).

```

```

    .SeriesCollection.Add( oExcel.Sheets[1].Range("G1:G13"), ;

```

```

        xlColumns, .T., .F., .F.)

```

```

    WITH .SeriesCollection[5]

```

```

        .ChartType = xlLineMarkers

```

```

        .MarkerStyle = xlMarkerStyleDiamond

```

```

        .MarkerForegroundColor = rgbBlack

```

```

        .MarkerBackgroundColor = rgbBlack

```

```

        .Border.Color = rgbBlack

```

```

    ENDWITH

```

```

    * Измените цвета полос на оттенки серого.

```

```

    .SeriesCollection[1].Interior.Color = rgbWhite

```

```

    .SeriesCollection[2].Interior.Color = rgbLtGray

```

```

    .SeriesCollection[3].Interior.Color = rgbMedGray

```

```

    .SeriesCollection[4].Interior.Color = rgbDkGray

```

```

    * Форматирование внутреннего цвета области графика.

```

```

    .PlotArea.Interior.Color = rgbWhite

```

```

ENDWITH

```

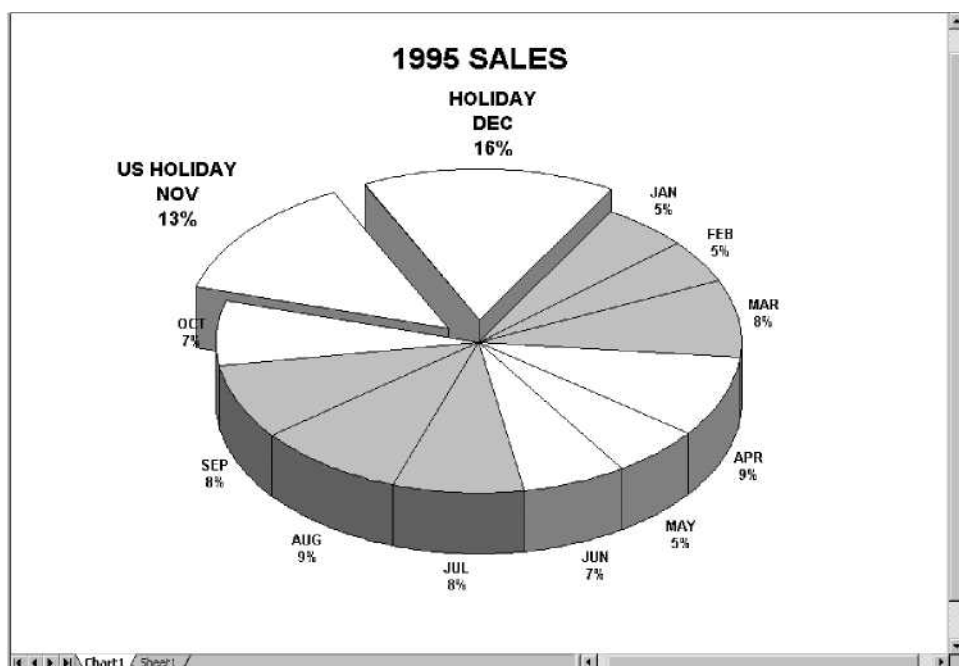


Рисунок 18. Трехмерная круговая диаграмма продаж 1994 года. На этой диаграмме демонстрируются некоторые функции трехмерных диаграмм, такие как разнесение, вращение и возвышение, а также работа с несмежными диапазонами данных и форматирование заголовков и меток данных.

* --- Создайте вторую диаграмму. На этот раз это лист диаграммы.

```
oChart = oExcel.Charts.Add()
```

```
oChart.ACTIVATE()
```

* Используйте только названия категорий и данные за 1994 год.

* Обратите внимание, что лист диаграммы был добавлен перед листом

* данных, что перемещает лист данных во второй экземпляр в

* коллекции листов.

```
oSourceRange = oExcel.Sheets[2].Range("CategoryNames; Year1994")
```

```
WITH oExcel.ActiveChart
```

* Создайте трехмерную круговую диаграмму.

```
.ChartWizard(oSourceRange, xl3DPie, autoPieFormat, ;  
             xlColumns, autoOneCategoryLabel, autoOneSeriesLabel, ;  
             autoNotHasLegend, "ПРОДАЖИ ЗА 1994 г")
```

* Установите высоту немного больше, чем по умолчанию.

```
.Elevation = 35
```

* Поверните диаграмму. Вращение круговой диаграммы меняет

* угол первого куска.

```
.Rotation = 30
```

* Увеличьте размер шрифта заголовка.

```
.ChartTitle.Font.Size = 22
```

```
WITH .SeriesCollection[1]
```

* Сделать все метки данных жирным шрифтом

```
.DataLabels.Font.Bold = .T.
```

* Отформатируйте ноябрьский сегмент, расчленив его,

```
* добавив собственную метку и изменив размер шрифта метки.
WITH .Points[11]
    .Explosion = 15
    .DataLabel.Text = "ПРАЗДНИК В США" + CHR(13) + ;
        .DataLabel.Text
    .DataLabel.Font.Size = .DataLabel.Font.Size + 4
ENDWITH

* Отформатируйте декабрьский сектор, расчленив его,
* добавив собственную метку и изменив размер шрифта метки.
WITH .Points[12]
    .Explosion = 15
    .DataLabel.Text = "ПРАЗДНИК" + CHR(13) + .DataLabel.Text
    .DataLabel.Font.Size = .DataLabel.Font.Size + 4
ENDWITH

ENDWITH

* Определите каждый квартал: Q1 и Q3 - серые, Q2 и Q4 - белые
FOR I = 1 TO 12
    IF I <= 3 OR (I >= 7 AND I <=9)
        .SeriesCollection[1].Points[I].Interior.Color = rgbLtGray
    ELSE
        .SeriesCollection[1].Points[I].Interior.Color = rgbWhite
    ENDIF
NEXT I

ENDWITH
```

Механизм построения диаграмм Excel очень надежен и предлагает множество вариантов форматирования. Вы удивите своих клиентов (и себя) количеством доступных вариантов.

Глава 10. Основы PowerPoint

Хотя большинство людей рассматривают PowerPoint как инструмент для создания мощных презентаций, опытный разработчик видит множество возможностей для визуально привлекательных вариантов вывода, которые недоступны в средстве создания отчетов FoxPro.

PowerPoint – отличный инструмент для создания визуальных презентаций, состоящих из слайдов, где каждый слайд содержит элементы текста, графики и даже мультимедиа. Богатые функции форматирования обеспечивают контроль над внешним видом слайда. Вы можете изменить цвет, шрифт и размер текстовых объектов, вплоть до изменения атрибутов каждого символа. Предоставляются многочисленные элементы фигур с множеством вариантов форматирования, таких как градиентная или текстурированная заливка фона, различные границы и возможность включать текст в фигуру.

Учитывая мощь и возможности современных компьютеров, а также стремление населения к высокотехнологичным визуальным презентациям, PowerPoint имеет возможность добавить навороты, которые нравятся сегодняшней аудитории. Объекты можно настроить на мигание, воспроизведение звуков, переход к другим слайдам в слайд-шоу, запуск макросов или даже запуск внешних программ, когда пользователь наводит на них указатель мыши или щелкает по ним. Фактически, вы можете установить другие действия для наведения курсора мыши, чем для щелчка мыши. Вы почувствуете себя голливудским продюсером, выбрав один из множества эффектов вытеснения, затухания и звуковых эффектов, которые будут активироваться для каждого слайда при переходе от одного к другому.

В PowerPoint также есть инструменты, помогающие докладчику, например сохранение заметок для каждого слайда. Эти заметки могут быть предназначены для выступающего или распечатаны для раздачи аудитории. Само слайд-шоу можно распечатать в нескольких форматах: от одной до девяти миниатюр на странице. Если проявите немного воображения, PowerPoint станет очень ценной альтернативой вывода для программ FoxPro.

10.1 Объектная модель PowerPoint

Как и в других программах Office, объектом верхнего уровня PowerPoint является объект Application, в котором хранятся настройки и параметры всего приложения (размер главного окна, активный принтер и т. д.). Объект Application также обеспечивает доступ ко всем другим объектам PowerPoint.

Основным объектом PowerPoint является объект Presentation, который представляет собой презентацию с одним слайд-шоу. Объект Presentation соответствует файлу PowerPoint (PPT); пользователи PowerPoint также называют файл PowerPoint «презентацией». (Термин «презентация» используется в файлах справки и другой документации по-разному; убедитесь, что вы понимаете контекст, в котором он используется, что может быть сложно в 2:30 ночи!)

Объект Application PowerPoint отслеживает все открытые объекты Presentation в коллекции Presentations. Каждый объект Presentation хранит некоторые характеристики презентации по умолчанию. Объект SlideMaster управляет внешним видом каждого объекта слайда по умолчанию и управляет такими функциями, как фон слайда, цветовая схема, стили текста, а также верхние и нижние колонтитулы. Также доступны объекты, которые устанавливают значения по умолчанию для форматов печати заметок и раздаточных материалов, называемые NotesMaster и HandoutMaster. Объект SlideShowSettings хранит такую информацию, как на каком слайде начинать и заканчивать, как продвигать слайды и следует ли запускать

непрерывный цикл. Свойство `ActivePresentation` приложения указывает на активный объект `Presentation`.

В каждой презентации есть коллекция `Slides`, содержащая объект `Slide` для каждого слайда презентации. Все, добавленное на слайд – текст, растровые изображения, фигуры, линии и т. д. – сохраняется в коллекции `Shapes`. Эта концепция на первый взгляд кажется немного неудобной, поскольку не кажется интуитивно понятным, что текст, линии и растровые изображения должны храниться в одной коллекции, разве средний разработчик баз данных не нормализует их в отдельные таблицы?

Будет полезно, если вы определите «форму» как нечто, имеющее размер (высоту, ширину), цвета (границы, фон, передний план; возможно, даже растровое изображение) и текстовый элемент (который не всегда используется). Так как же отличить все эти формы друг от друга? Свойство `Type` содержит число, описывающее форму. Вы также можете использовать свойство `Name`, чтобы прикрепить осмысленное имя, аналогично тому, как вы называете каждый элемент управления, помещенный в форму FoxPro.

Существует несколько объектов `Range`. Вы можете сгруппировать любое количество слайдов в объект коллекции `SlideRange` и любое количество фигур в объект коллекции `ShapeRange`. Ссылаясь на коллекцию слайдов или объектов, вы можете легко установить свойства всех элементов коллекции.

Справка PowerPoint Visual Basic содержит диаграмму объектной модели PowerPoint. (Подробную информацию о том, как найти эту справку и что делать, если вы не можете ее найти, см. в главе 2 «Серверы Office».) Рисунок «живой» – когда вы нажимаете на объект, вы попадаете в раздел справки для этого объекта. На рисунке 1 показана часть диаграммы объектной модели, описывающая объект `Presentation`.

Microsoft PowerPoint Objects

[See Also](#)

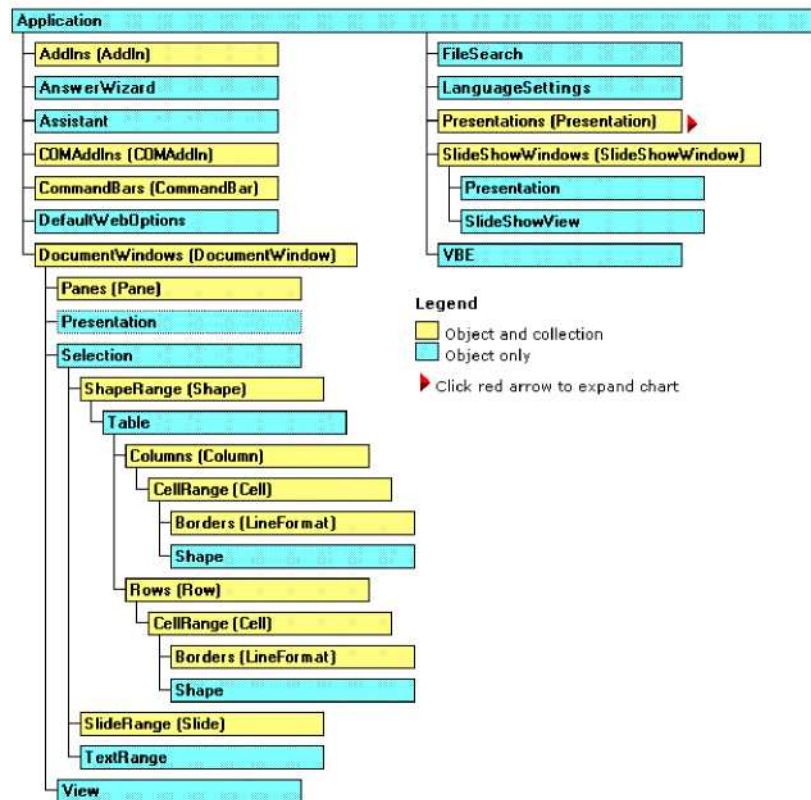


Рисунок 1. Объектная модель PowerPoint. Эта диаграмма доступна в файле справки и показывает иерархию доступных объектов PowerPoint.

10.2 Дежавю

Если вы прочитаете эту книгу от начала до конца, читая эту главу, вы скажете себе: «Боже, я уже читал это раньше!» (тем более, что это по сути тот же текст, что и в главе 7 «Основы Excel»). Office 2000 является объектно-ориентированным и полиморфным. Полиморфизм буквально означает «множество форм», а применительно к ООП это означает, что разные объекты имеют свойства и методы, которые ведут себя согласованно. Чтобы записать файл на диск, вы используете метод Save, независимо от того, работаете ли вы в Word, Excel, PowerPoint или Outlook. Это не означает, что команды Save выполняют одно и то же: при сохранении документа Word должны выполняться иные действия, чем при сохранении презентации PowerPoint. Для каждого могут даже быть разные параметры. Однако вы можете быть уверены, что команда Save сохранит вашу работу на диск.

Преимущество полиморфизма в том, что если вы знаете, как сделать что-то в одном инструменте, вы будете знать, как сделать это и в остальных. В Office это обычно так. Однако полиморфизм включает в себя способность методов принимать разные параметры, поскольку разные объекты совершенно разные. Поэтому вам необходимо знать, что синтаксис может меняться (и меняется) в разных приложениях Office.

Между приложениями Office так много общего, что мы могли бы написать в этой главе что-то вроде: «Просто используйте функцию CreateObject(), как описано в главе Word, но вместо этого используйте «PowerPoint.Application». Мы посчитали, что было бы лучше иметь полное объяснение каждого приложения, чтобы вам не приходилось листать книгу назад и вперед. Это также упрощает объяснение тонких синтаксических различий, существующих между приложениями. После того, как мы преодолеем основы открытия и сохранения файлов, мы перейдем к достаточному количеству специфичных для приложения функций, поэтому это покажется менее повторяющимся.

Преимущество полиморфизма в том, что если вы знаете, как сделать что-то в одном объекте, вы сможете сделать это и во всех из них. Однако объяснение схожих концепций для каждого объекта приводит к избыточности текста. Просто думайте об этом избыточном тексте как о «функции».

10.3 Как разобраться с приложением

Теперь, когда у вас есть некоторый опыт работы с объектной моделью PowerPoint, введите следующую команду в командное окно, чтобы создать экземпляр PowerPoint с помощью Автоматизации:

```
oPowerPoint = CreateObject("PowerPoint.Application")
```

Курсор возвращается обратно через секунду (он перейдет на диск, чтобы открыть PowerPoint, дайте ему наносекунду!). Теперь создан экземпляр PowerPoint. «Да, верно», – говорите вы, проверьте список задач, если не верите нам! Помните, что серверы Office создаются как невидимые. Итак, введите следующую команду, чтобы увидеть экземпляр:

```
oPowerPoint.Visible = .T.
```

Сейчас вы просматриваете приложение PowerPoint. Переменная oPowerPoint – это дескриптор объекта приложения PowerPoint. Объект Application представляет весь экземпляр PowerPoint. Он знает, какие презентации открыты через коллекцию Presentations, и знает, какая из них активна, через свойство ActivePresentation.

Управлять размером и расположением окна PowerPoint легко с помощью свойств

Top, Left, Height и Width. Все это измеряется в пунктах. Следующий код устанавливает верхнюю и левую границы примерно на 1 дюйм от края рабочего стола и делает его квадратным примерно на 4 дюйма (помните, что на дюйм приходится 72 пункта). Дюймы, используемые для отображения окон, являются виртуальными дюймами, что означает, что они приблизительно соответствуют реальному дюйму. Однако существуют различия в размерах дисплея, что влияет на размер окна. На дисплее размером более 20 дюймов размер окна составляет около 5 дюймов и 1,25 дюйма сверху и слева. На крошечном дисплее ноутбука оно может составлять всего 3,5 дюйма. Разрешение экрана учитывается в виртуальных дюймах, поэтому размер окна должен быть одинаковым на любом дисплее независимо от разрешения. В любом случае окно занимает одинаковые пропорции на любом экране, на котором оно отображается. Если вы следуете инструкциям в командном окне, прежде чем продолжить, убедитесь, что для окна PowerPoint установлено значение Normal (а не свернутое или развернутое).

```
WITH oPowerPoint
    .Top      = 72
    .Left     = 72
    .Width    = 288
    .Height   = 288
ENDWITH
```

Если вы продолжаете вводить эти команды в командном окне FoxPro, а окно PowerPoint было свернуто или развернуто, вы получите сообщение об ошибке: «Код исключения ошибки OLE IDispatch 0». PowerPoint не позволяет вам устанавливать эти свойства, если приложение минимизировано или максимизировано. Когда вы пишете код, вам необходимо проверить свойство WindowState, чтобы убедиться, что окно PowerPoint является «нормальным» (ни развернутым, ни свернутым). Свойство WindowState использует следующие внутренние константы (см. главу 2 «Офисные серверы» для объяснения констант VBA): ppWindowNormal (1), ppWindowMinimized (2) и ppWindowMaximized (3). Запустите следующий код, чтобы сделать окно «нормальным», прежде чем устанавливать высоту и ширину:

```
#DEFINE ppWindowNormal 1
WITH oPowerPoint
    IF .WindowState <> ppWindowNormal
        .WindowState = ppWindowNormal
    ENDIF
ENDWITH
```

Для ваших пользователей было бы неплохо изменить заголовок PowerPoint, чтобы они знали, почему внезапно появилось окно PowerPoint. Используйте свойство Caption, чтобы изменить заголовок окна.

```
oPowerPoint.Caption = "Автоматизировано на основе величайшего "+"  
"в мире приложения"
```

Метод Quit закрывает экземпляр PowerPoint. Если открыты какие-либо несохраненные презентации, пользователю предлагается разобраться с ними, и дальнейшая обработка приостанавливается до тех пор, пока пользователь не ответит на запросы. См. следующий раздел «Управление презентациями и слайдами» для получения информации о том, как сохранить презентации перед вызовом метода Quit. Этот метод не принимает никаких параметров. Это просто:

```
oPowerPoint.Quit
```

Метод выхода работает по-разному в различных приложениях Office. В Word вы можете передать ему параметр для автоматического сохранения изменений,

запроса пользователя на внесение изменений или выхода без сохранения изменений. Excel ведет себя как PowerPoint и сообщает пользователю, есть ли несохраненные изменения. Outlook просто закрывается без сохранения изменений. Имейте в виду, что, несмотря на полиморфизм, объектные модели Office иногда несовместимы!

10.4 Управление презентациями и слайдами

Когда вы открываете PowerPoint с помощью Автоматизации, он не открывается автоматически и не спрашивает пользователя, хочет ли он открыть существующую презентацию или создать новую. Это ваша работа. PowerPoint запускается и остается с красивым серым экраном (то есть, если вы установили для свойства Visible значение True), пока вы не скажете ему, что делать. Сначала вы создаете объект Presentation (используя метод Add коллекции Presentations), а затем добавляете объекты Slide (используя метод Add коллекции Slides).

Представление: объект Presentation

Коллекция Presentations представляет собой массив объектов Presentation. Думайте об объекте Presentation как о ссылке на один файл презентации (PPT).

Когда вы создаете экземпляр объекта приложения PowerPoint с помощью CreateObject(), он открывается без каких-либо активных презентаций. Чаще всего требуется добавить пустую презентацию. Используйте метод Add коллекции Presentations:

```
oPresentation = oPowerPoint.Presentations.Add()
```

oPresentation теперь является ссылкой на объект Presentation. Add принимает необязательный параметр, который определяет, видимо ли окно, содержащее новую презентацию. По умолчанию используется значение .T., поэтому окно по умолчанию становится видимым, пока объект приложения виден. Существует несколько способов доступа к новому объекту Presentation. Во-первых, это имя переменной, которое вы задали при добавлении презентации oPresentation. Объект Application имеет свойство ActivePresentation; используйте oPowerPoint.ActivePresentation, чтобы вернуть ссылку на активную презентацию. Внимание: если вы попытаетесь получить доступ к свойству ActivePresentation, когда ни одна презентация не открыта или приложение не отображается, произойдет ошибка. Прежде чем обращаться к ActivePresentation, проверьте значение PowerPoint.Presentations.Count и убедитесь, что оно больше нуля.

Как и другие коллекции, Presentations предлагает два способа доступа к своему содержимому. В первом используется индексный номер, соответствующий порядку открытия презентаций – в данном случае oPowerPoint.Presentation[1]. Во втором используется имя презентации (находится в свойстве Name). Например, на презентацию, сохраненную как «MyPresentation.PPT», можно сослаться как oPowerPoint.Presentation("MyPresentation.PPT"). Путь не включается, а имена презентаций не чувствительны к регистру.

Обратите внимание, что свойство Name для презентаций доступно только для чтения. При добавлении презентации имя – «ПрезентацияХ», где «Х» – число, соответствующее количеству файлов, созданных во время этого сеанса, но еще не сохраненных. После сохранения файла свойство Name отражает имя и расширение файла (но не путь) файла, сохраненного на диске. (Свойство FullName предоставляет имя и путь к сохраненному файлу, а свойство Path содержит только путь.)

Открытие существующей презентации

Чтобы открыть существующую презентацию, используйте метод `Open`. Синтаксис метода `Open` следующий:

```
oPowerPoint.Presentations.Open( cFileName, [lReadOnly],  
                                [lUntitled], [lWithWindow] )
```

cFileName	Character	Имя файла презентации. Обязательно укажите полный путь.
lReadOnly	Logical	.Т. чтобы открыть только для чтения. По умолчанию – чтение/запись (.F.). (Необязательный)
lUntitled	Logical	.Т. чтобы открыть без заголовка. По умолчанию – .F., а заголовком по умолчанию является имя файла. (Необязательный)
lWithWindow	Logical	.Т. открывает видимое окно, которое используется по умолчанию. .F. скрывает открытую презентацию. (Необязательный)

Итак, если вы дадите команду:

```
oWayCool = oPowerPoint.Presentations.Open("c:\" +  
      "My Documents\WayCool.PPT")
```

вы открываете презентацию `WayCool.PPT` в новом окне. По умолчанию заголовок окна – «`WayCool.PPT`», и вы можете ссылаться на этот объект презентации как `oPowerPoint.Presentations("WayCool.PPT")` или `oPowerPoint.ActivePresentation` или `oWayCool`.

Как отмечалось в главе 3 «`Visual FoxPro` как клиент Автоматизации», альтернативным методом открытия презентации является функция `GetObject()`. Ниже приведен синтаксис:

```
oPresentation = GetObject("c:\My Documents\WayCool.PPT")
```

При использовании только имени файла в качестве параметра `GetObject()` возвращает ссылку на объект `Presentation` (а не на объект `Application`, как это делает `CreateObject()`). Если `PowerPoint` не запущен (или запущен, но не виден), вам необходимо сделать приложение видимым, чтобы его увидеть:

```
oPresentation.Application.Visible = .T.
```

Презентация открыта и доступна для редактирования. Однако у неё еще нет окна, и поэтому она не отображается. Документ, открытый с помощью `GetObject()`, учитывает настройку свойства `Visible` приложения во время создания экземпляра объекта `Document`. Если приложение не было видимым (что произойдет, если вы используете синтаксис `GetObject()` для открытия документа без активного `PowerPoint`), вы должны сделать документ видимым, предоставив ему окно. Кстати, такое же поведение происходит при использовании метода `Open` с параметром `lWithWindow`, установленным в значение `False`. Вы должны использовать метод `NewWindow`, чтобы поместить презентацию в окно:

```
oPresentation.NewWindow()
```

Открытие презентаций без их отображения может быть использовано в ваших интересах. Если вы создадите презентацию, когда она не видна, вы увидите прирост производительности примерно на 30 процентов. Чтобы закрыть открытое окно, не закрывая презентацию, введите следующий код:

```
oPowerPoint.ActiveWindow.Close()
```

У вас по-прежнему есть доступ к презентации, но она не отображается в окне, поэтому ваша презентация создается значительно быстрее. Функция `GetObject()` имеет альтернативный синтаксис, который позволяет предотвратить многократное появление объекта. Если вы программируете Office 97, это может быть важной проблемой. Возможно, вы захотите использовать открытый экземпляр PowerPoint, чтобы не использовать больше памяти для другого экземпляра. Одним из изменений в Office 2000 является то, что существует только один экземпляр приложения; само приложение Office обеспечивает лучшее управление памятью. Однако синтаксис `GetObject()` по-прежнему работает в PowerPoint 2000 (и открывает приложение, если оно еще не открыто). Команда:

```
oPowerPoint = GetObject(, 'PowerPoint.Application')
```

открывает PowerPoint и возвращает ссылку на приложение. Обратите внимание, что синтаксис открытия имени файла возвращает ссылку на презентацию.

В оставшейся части этого раздела мы будем предполагать, что существует ссылка на открытую презентацию и эта ссылка называется `oPresentation`.

Добавление слайдов

Презентация PowerPoint состоит из слайдов. Отдельные слайды – это объекты `Slide`, принадлежащие коллекции `Slides` (точно так же, как объекты `Presentation`, принадлежащие коллекции `Presentations`). Добавьте объект `Slide` в презентацию с помощью метода `Add`:

```
oSlide = oPresentation.Slides.Add(nIndex, nLayout)
```

`nIndex` Numeric Индексный номер слайда.

`nLayout` Numeric Одна из констант `ppSlideLayout` для выбора одного из 29 предопределенных вариантов макета слайда.

Узнать, какие параметры доступны для параметра `nLayout`, легко, если вы понимаете, что макеты соответствуют диалоговому окну, которое появляется при добавлении слайда вручную в PowerPoint, как показано на рисунке 2. При нажатии на миниатюры макета слайда в правом нижнем углу меняется и примерно соответствует имени константы.

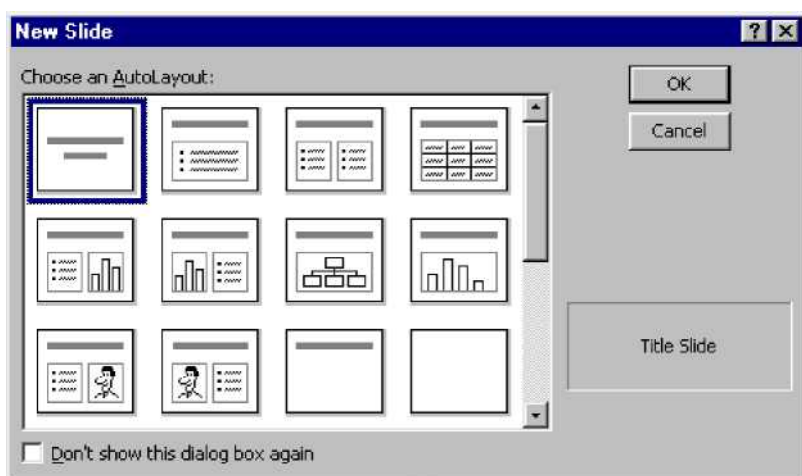


Рисунок 2. Диалоговое окно «Новый слайд» в PowerPoint. В этом диалоговом окне содержится визуальная ссылка на предопределенные макеты слайдов.

Константами некоторых наиболее распространенных стилей являются `ppLayoutTitle`

(1), ppLayoutText (2), ppLayoutTable (4) и ppLayoutBlank (12).

Индекс слайда представляет положение этого слайда в слайд-шоу. Индекс 1 вставляет новый слайд в начало презентации. Индекс 5 вставляет новый слайд после слайда 4 и увеличивает индексы следующих слайдов. Указание индекса, превышающего общее количество слайдов более чем на один, приводит к ошибке. Используйте oPresentation.Slides.Count, чтобы убедиться, что передаваемое значение не слишком велико.

```
#DEFINE ppLayoutTitle 1
#DEFINE ppLayoutBlank 12
```

* Добавьте титульный слайд в начало презентации.

```
oSlide = oPresentation.Slides.Add(1, ppLayoutTitle)
```

* Добавьте "пустой" слайд в конец презентации.

```
oSlide = oPresentation.Slides.Add(oPresentation.Slides.Count + 1, ;
    ppLayoutBlank)
```

Как и в случае с объектом Presentation, существует несколько способов доступа к слайду. Вы можете использовать переменную (oSlide в предыдущем примере). Вы также можете использовать свойство Slides, передав либо индекс, либо имя слайда, как в oPresentation.Slides[1] или oPresentation.Slides["Slide1"]. Обратите внимание, что свойство ActiveSlide отсутствует. Вы можете получить доступ к активному объекту слайда с помощью oPowerPoint.Windows[1].View.Slide, но только если объект Application виден. Кажется, не существует способа определить активный слайд, если объект Application невидим.

Когда вы добавляете слайд, его свойство Name по умолчанию содержит слово «Slide», за которым следует целое число, которое является следующим доступным номером слайда в слайд-шоу. Если вы пишете приложение, которое возвращается и редактирует слайды, разумно установить для свойства Name такое значение, которое будет легче запомнить.

Существует также объект коллекции SlideRange, который представляет диапазон слайдов. Согласно файлу справки, вы можете использовать метод Range коллекции слайдов для выбора серии слайдов. В справке говорится, что метод Range может принимать индексный номер, имя слайда или серию индексов и названий слайдов в массиве VBA. Однако массивы из FoxPro несовместимы с массивами VBA, поэтому работать они не будут. Вы можете передать в Range один индекс или одну строку символов, представляющую имя слайда, и работать с одним слайдом. Однако синтаксис:

```
oPresentation.Slides["Slide3"]
```

легче читать (и выполняется немного быстрее), чем:

```
oPresentation.Slides.Range("Slide3")
```

Так почему же мы подняли этот вопрос? Потому что средство записи макросов генерирует множество операторов, используя метод Range. Аналогичный метод Range существует и для фигур и заметок, и он часто используется средством записи макросов.

Поэтому, если вы генерируете код из средства записи макросов, обязательно удалите все ссылки на несколько слайдов, а также вы можете удалить ключевое слово метода Range.

Сохранение презентации

Быстро просмотрев методы объекта Presentation, мы подошли к Save. Инстинкт подсказывает нам использовать только oPresentation.Save, который работает, но не так, как вы думаете. Помните, что имя презентации по умолчанию – «ПрезентацияХ». PowerPoint сохраняет файл в каталоге, из которого был запущен PowerPoint, как «ПрезентацияХ.PPT». Возможно, это не *совсем* то, что хотели вы или ваши пользователи! Метод Save не принимает параметры.

Не очень интуитивное решение – использовать метод SaveAs. К счастью, эта информация задокументирована для метода Save, поэтому, если вы ее забудете, вы сможете найти ее в файле справки. Синтаксис метода SaveAs:

```
oPresentation.SaveAs(cFileName, [nFileFormat], [lEmbedFonts])
```

cFileName	Character	Полное имя файла презентации.	The fully
nFileFormat	Numeric	Используйте одну из констант ppSaveAs, чтобы выбрать один из поддерживаемых форматов. Если он не включен, презентация сохраняется в формате текущей презентации. (Необязательный)	Use one supported saved in
lEmbedFonts	Logical	Используйте .T. если вы хотите встроить информацию о шрифте в файл. По умолчанию – .F. (Необязательный)	Use .T. if file. The

Не забудьте использовать полное имя файла. Тот факт, что каталоги FoxPro по умолчанию установлены так, как вы хотите, и вы повторно вводите команды из VFP, не означает, что PowerPoint знает об этих значениях по умолчанию.

Имейте в виду, что PowerPoint ничего не знает о настройке SET SAFETY в FoxPro. Выполнение этой команды перезапишет существующий файл без какого-либо уведомления.

Существуют константы ppSaveAs для каждого поддерживаемого формата, который появляется в интерактивном диалоговом окне «Сохранить как». Office 2000 поддерживает 18 форматов, включая графические форматы (BMP, GIF и JPG), предыдущие версии PowerPoint, форматы обмена данными (RTF, MetaFile), HTML и другие (Office 97 поддерживает семь форматов, в основном предыдущие версии или RTF). Часто используемые константы и их значения: ppSaveAsPresentation (1), ppSaveAsHTML (12) (недоступно в Office 97), ppSaveAsShow (7) и ppSaveAsTemplate (5). Итак, как узнать, когда использовать SaveAs вместо Save? Когда вы открываете новый документ, его имя – «ПрезентацияХ». Проверьте свойство Name, чтобы узнать, имеет ли оно имя, которое вы хотите использовать, или оно похоже на имя по умолчанию. Возможно, вы захотите сравнить свойство Name с именем вашего файла (без пути) вместо показанного примера:

```
#DEFINE ppSaveAsPresentation 1

IF LEFT(oPresentation.Name, 11) == "Презентация"
    oPresentation.SaveAs("C:\MyDocs\MyPresentation.PPT", ;
        ppSaveAsPresentation)
ELSE
    oPresentation.Save()
ENDIF
```

Заккрытие презентации

Вы можете закрыть презентацию в любой момент, независимо от того, сохранена

она или нет, с помощью метода Close. Метод Close не принимает параметров.

```
oPresentation.Close()
```

Если вы закроете несохраненную презентацию, предупреждение не появится. Чтобы проверить, сохранена ли презентация перед ее закрытием, проверьте свойство Saved, которое содержит значение 0, если презентация изменилась с момента ее создания или последнего сохранения, и 1, если она не изменилась.

```
#DEFINE ppSaveAsPresentation 1
```

```
* Проверьте, сохранена ли презентация, прежде чем закрывать ее
IF oPresentation.Saved = 0
    * Лучше сохраните эту презентацию.
    * Убедитесь, что она сохранена с правильным именем
    IF LEFT(oPresentation.Name, 11) == "Презентация"
        oPresentation.SaveAs("C:\Users\User\Documents\" + ;
            "MyPresentation.PPT", ppSaveAsPresentation)
    ELSE
        oPresentation.Save()
    ENDIF
ENDIF
oPresentation.Close()
```

Закрытие объекта приложения

Чтобы завершить работу PowerPoint, вызовите метод Quit объекта приложения:

```
oPowerPoint.Quit()
```

Это разрывает соединение с объектом приложения PowerPoint. Все презентации закрываются (без сохранения), *если* в PowerPoint нет переменных, указывающих на ссылки на объекты. Если такая ссылка есть, ни одна из открытых презентаций не закрывается, а существующие ссылки на объекты действительны. Когда вы закрываете последнюю презентацию (или освобождаете все ссылки на объекты), все оставшиеся презентации закрываются.



Это один из случаев, когда в PowerPoint 2000 объектная модель немного изменилась. В PowerPoint 2000 может быть один и только один экземпляр объекта приложения PowerPoint; CreateObject() ссылается на уже запущенный экземпляр PowerPoint и создает на него еще одну ссылку. В PowerPoint 97 каждый вызов CreateObject() создает новый экземпляр приложения. Дополнительное отличие состоит в том, что PowerPoint 97 закрывается, не обращая внимания на наличие открытых переменных, указывающих на экземпляр PowerPoint.



Чтобы изучить поведение выхода из PowerPoint 2000, запустите код, показанный в листинге 1. Он сохраняется как PPTQuit.PRG в файлах загрузки для разработчиков, доступных по адресу www.hentzenwerke.com.

Листинг 1. Особенности выхода из PowerPoint 2000. Как и в Visual FoxPro, обязательно освободите все ссылки на объекты, чтобы успешно закрыть серверное приложение.

```
* PPTQuit.PRG
* Copyright (c)2000, Tamar Granor and Della Martin
```



```

* From _Automating Microsoft Office with Visual FoxPro_,
* Hentzenwerke Press. www.hentzenwerke.com
*
* Создайте экземпляр приложения.
oPowerPoint = CreateObject("PowerPoint.Application")
oPowerPoint.Visible = .T.

WITH oPowerPoint
    * Создайте первую презентацию.
    oPresentation1 = .Presentations.Add()

    * Создайте вторую презентацию.
    oPresentation2 = .Presentations.Add()
ENDWITH

= MESSAGEBOX("Нажмите Alt-Tab, чтобы перейти к PowerPoint и " + ;
    "вручную открыть новую презентацию.")

* Вы можете поставить здесь SUSPEND, чтобы можно было
* использовать Alt-Tab для запуска PowerPoint.
* Вручную откройте новую презентацию и посмотрите,
* что с ней произойдет, если у вас нет ссылки на объект.

= MESSAGEBOX("Нажмите ОК, чтобы закрыть Презентацию2")
oPresentation2.Close()

= MESSAGEBOX("Нажмите ОК, чтобы закрыть объект приложения." + ;
    CHR(13) + CHR(13) + ;
    "Обратите внимание, что ничего не происходит.")
oPowerPoint.Quit()

= MESSAGEBOX("Нажмите ОК, чтобы закрыть Презентацию1")
oPresentation1.Close()

= MESSAGEBOX("Приложение и любые презентации, открытые вами " + ;
    "вручную, по-прежнему доступны! Нажмите ОК, " + ;
    "чтобы освободить oPowerPoint.")
RELEASE oPowerPoint && Office 97 успешно завершает работу здесь.

= MESSAGEBOX("Оно все еще здесь! Теперь нажмите ОК, чтобы " + ;
    "освободить oPresentation1.")
RELEASE oPresentation1

= MESSAGEBOX("Оно все еще здесь! Теперь нажмите ОК, чтобы " + ;
    "освободить oPresentation2.")
RELEASE oPresentation2

= MESSAGEBOX("Его больше нет, как и презентаций, которые вы " + ;
    "открывали вручную." + CHR(13) + ;
    "Не забудьте освободить ВСЕ ваши переменные!")

```

Как и в случае с формами FoxPro, вам необходимо освободить все ссылки на объекты, прежде чем вы сможете уничтожить экземпляр основного объекта (только в PowerPoint 2000).

Сделать так, чтобы это выглядело хорошо для пользователей

Прежде чем перейти к добавлению контента на слайды, давайте рассмотрим некоторые свойства и методы, которые позволяют сделать процесс построения презентации более привлекательным для пользователя. Мы уже обсуждали параметры Visible, Left, Top, Width и Right. Они контролируют, видит ли пользователь PowerPoint и где на экране появляется окно PowerPoint. Если положение и размер окна не заданы в коде, окно PowerPoint отображается, где и как оно было у пользователя в последний раз. Его можно развернуть, свернуть или сделать нормальным в любом месте экрана.

Во время разработки нам хотелось бы, чтобы FoxPro располагался в обычном окне на всю высоту экрана и занимал около двух третей его ширины с одной стороны, а затем заставлял PowerPoint подниматься на две трети ширины с противоположной стороны. Таким образом, результаты выполнения кода легко увидеть.

Пользователям тоже поначалу нравится такая расстановка — за ней очень здорово смотреть. Однако вскоре им это надоедает, как раз в тот момент, когда они жалуются на производительность (верный признак того, что новизна себя изжила). Затем вы можете добавить функцию, позволяющую пользователю не видеть окончательную презентацию до конца. Это также работает примерно на 30 процентов быстрее (в зависимости от конфигурации оборудования и типов созданных слайдов), что заставляет вас выглядеть героем из-за того, что вы так чутко реагируете на их потребности.

Несмотря на то, что PowerPoint виден, есть причины поместить PowerPoint перед вашим приложением, особенно если пользователи пытаются наблюдать за ним во время сборки. Это достигается с помощью метода Presentation Activate:

```
oPowerPoint.Activate()
```

Этот метод выводит текущее окно презентации поверх всех запущенных приложений Windows. В этой ситуации вам также понадобится отображать слайды во время работы над ними. Используйте метод Select Slides, чтобы переместить слайд на передний план:

```
oPresentation.Slides[1].Select()
```

Примечание для пользователей PowerPoint 97: предыдущая строка приведет к ошибке. Вместо этого используйте следующую строку:

```
oPresentation.ActiveWindow.View.GoToSlide(2)
```

Когда вы покажете этот слайд? Вы запускаете метод Select сразу после его добавления? Если пользователи смотрят, это дает очень хорошую демонстрацию того, что именно происходит. Пользователи действительно могут оценить «тяжелую работу», которую выполняет компьютер. Однако, просмотрев его несколько раз, пользователям может немного наскучить, и они захотят, чтобы он работал быстрее. Чтобы повысить производительность (обычно примерно на 15-20 процентов), подождите, пока слайд не будет полностью отрисован, а затем вызовите метод Select. Добавление и форматирование фигур на скрытом слайде позволяет избежать трудоемкой перерисовки, и ваши слайды появляются на экране. Если ваши пользователи действительно жалуются, не делайте объект Application видимым до конца, а вместо этого используйте индикатор выполнения, чтобы указать относительное оставшееся время.

10.5 Работа с содержимым слайдов.

Создать презентацию, полную пустых слайдов, несложно. Следующим шагом будет добавление текста и фигур на слайды. Любой объект на слайде, будь то

изображение, заголовок, текст, фигура, объект OLE или что-то еще, становится одним объектом в коллекции Shapes. На первый взгляд это кажется странным способом создания коллекции, поскольку для многих разных типов элементов очевидно требуются разные наборы свойств. Если посмотреть немного глубже, если вы видите каждый из этих объектов как нечто, занимающее высоту и ширину в определенном положении и содержащее определенный тип контента с необязательным текстом, тогда каждый из этих странных объектов начинает выглядеть так, как будто он может принадлежать к одной и той же коллекции.

Как и в большинстве коллекций COM, вы можете ссылаться на конкретный объект Shape, используя его индекс в коллекции (например, `oSlide.Shapes[7]`) или его имя (`oSlide.Shapes["Rectangle 2"]`).

Имя фигуры генерируется при ее добавлении на слайд. Соглашение об именовании по умолчанию — это ключевое слово типа фигуры, за которым следует число, обозначающее количество фигур на слайде на момент добавления фигуры, плюс единица. Таким образом, если к пустому слайду добавляются линия, прямоугольник, текстовое поле и автофигура, их имена по умолчанию — «Line 2», «Rectangle 3», «Text Box 4» и «AutoShape 5» соответственно (вместе со встроенным пробелом). Свойство Name доступно для чтения и записи, поэтому у вас есть возможность их изменить. Если вы создаете приложение, которое редактирует сгенерированную презентацию, настоятельно рекомендуется изменить имена на более значимые. Однако если вы создаете презентацию от начала до конца и не имеете возможности ее редактировать, не тратьте время на изменение названий. Пользователь не имеет возможности видеть имена фигур в PowerPoint, поэтому кажется глупым платить небольшой штраф за производительность за изменение имен, когда они не будут использоваться.

Использование макетов слайдов

На выбор имеется 29 предустановленных макетов слайдов; вы должны выбрать один при добавлении слайда. Каждый формат имеет предопределенный набор фигур, называемых заполнителями, готовых к заполнению. В зависимости от формата существует от нуля до пяти заполнителей (все прямоугольные фигуры), уже предварительно заданных со свойствами по умолчанию. Это упрощает создание слайдов: просто заполните свойства и вперед!

Таблица 1 показывает различные макеты, включая имена констант и их значения (чтобы вы могли интерактивно играть с командами в командном окне), а также количество и тип фигур на слайде. Доступ к объектам Shape можно получить через коллекцию Shapes, где они являются первыми объектами Shape в коллекции. Вы также можете использовать коллекцию Placeholders для доступа к ним. У них есть тот же индексный номер, что и у коллекции Shapes, но любые другие фигуры, добавленные на слайд, не отображаются в коллекции Placeholders, а только в коллекции Shapes. Фигуры перечислены на слайде в порядке индекса. Например, если вы посмотрите на `ppLayoutChartAndText`, заголовок — `Shapes[1]` и `Placeholders[1]`, диаграмма — `Shapes[2]` и `Placeholders[2]`, а маркированный текст — это `Shapes[3]` и `Placeholders[3]`. Единственная разница между этими заполнителями и фигурами, которые вы добавляете самостоятельно, заключается в том, что они заранее определены с часто используемыми настройками по умолчанию. К сожалению, не существует констант для обозначения индекса различных заполнителей. Такие константы определить невозможно, поскольку маркированный текст может находиться на второй или третьей позиции в зависимости от используемого макета слайда. Просто будьте довольны использованием этой таблицы для ссылки на нужные вам значения индекса.

Таблица 1. Макеты слайдов PowerPoint. Предопределенные макеты содержат множество фигур, позволяющих легко создавать презентации без необходимости начинать с нуля.

Константа	Значение	Количество фигур	Формы предоставлены
ppLayoutBlank	12	0	Ничего
ppLayoutChart	8	2	Название, диаграмма
ppLayoutChartAndText	6	3	Название, диаграмма, маркированный текст
ppLayoutClipartAndText	10	3	Заголовок, картинка, маркированный текст
ppLayoutClipartAndVerticalText	26	3	Заголовок, картинка и маркированный текст по бокам
ppLayoutFourObjects	24	5	Заголовок, два сложенных друг на друга объекта рядом с двумя сложенными объектами
ppLayoutLargeObject	15	1	Объект OLE (без заголовка)
ppLayoutMediaClipAndText	18	3	Название, медиаклип, маркированный текст
ppLayoutMixed	-2		(Не выбираемый формат)
ppLayoutObject	16	2	Заголовок, объект OLE
ppLayoutObjectAndText	14	3	Заголовок, объект OLE, маркированный текст
ppLayoutObjectOverText	19	3	Заголовок, объект OLE над маркированным текстом
ppLayoutOrgchart	7	2	Должность, организационная структура
ppLayoutTable	4	2	Заголовок, прямоугольник таблицы
ppLayoutText	2	2	Название, маркированный текст
ppLayoutTextAndChart	5	3	Название, маркированный текст, диаграмма
ppLayoutTextAndClipart	9	3	Заголовок, маркированный текст, картинка
ppLayoutTextAndMediaClip	17	3	Заголовок, маркированный текст, медиаклип
ppLayoutTextAndObject	13	3	Заголовок, маркированный текст, объект OLE

ppLayoutTextAndTwoObjects	21	4	Заголовок, маркированный текст рядом с двумя расположенными друг над другом объектами OLE
ppLayoutTextOverObject	20	3	Заголовок, маркированный текст над объектом OLE
ppLayoutTitle	1	2	Заголовок, подзаголовок
ppLayoutTitleOnly	11	1	Заголовок (оставляет место для чего-то под ним)
ppLayoutTwoColumnText	3	3	Заголовок, маркированный текст в две колонки
ppLayoutTwoObjectsAndText	22	4	Заголовок, два расположенных друг над другом объекта OLE рядом с маркированным текстом
ppLayoutTwoObjectsOverText	23	4	Заголовок, два расположенных рядом объекта OLE поверх маркированного текста
ppLayoutVerticalText	25	2	Заголовок, маркированный текст сбоку
ppLayoutVerticalTitleAndText	27	2	Заголовок и маркированный текст сбоку (книжная ориентация)
ppLayoutVerticalTitleAndTextOverChart	28	3	Название сбоку, текст сбоку, диаграмма (книжная ориентация)

В этих макетах по умолчанию замечательно то, что фигуры полностью редактируются; PowerPoint предоставляет вам отличный вариант по умолчанию, и вы можете улучшить его по своему усмотрению. Это означает, что вы можете перемещать, изменять размер, добавлять или удалять любую из этих фигур.



PowerPoint определяет автомакеты, содержащие объекты OLE. Добавление объектов OLE в заполнители хорошо работает для пользователей интерактивных PowerPoint. Он не предназначен для Автоматизации заполнителей для принятия объекта OLE (с помощью макросов или Автоматизации). Это описано в статье базы знаний Microsoft № Q160252 для PowerPoint 97 и № Q222796 для PowerPoint 2000. Вы по-прежнему можете использовать объекты макета для хранения изображений или текста, но не можете изменить заполнители на объекты OLE. Необходимо добавить объект OLE с помощью метода AddMediaObject коллекции Shapes.

Формы, формы и еще раз формы

Теперь самое интересное: настройка свойств фигур, чтобы этот слайд о чем-то говорил и выглядел великолепно! Очевидное место для начала – титульный слайд. Это не только первый слайд презентации, но и фигуры, с которыми легче всего работать – обе они являются текстовыми.



Следующий код открывает новую презентацию и добавляет титульный слайд. На рисунке 3 показан результат. Вы получите наибольшую выгоду, если попыдаете сделать это в интерактивном режиме в командном окне, так как сможете поиграть со свойствами и посмотреть, как они изменяются. Просто не забудьте заменить константы их числовыми значениями, поскольку командное окно не использует автоматически #DEFINES. (За исключением, конечно, случая, когда вы вырезаете и вставляете код с #DEFINES, выделяете все команды, включая #DEFINES, а затем выполняете выделение. Дополнительные советы по интерактивному тестированию см. в главе 2 «Офисные серверы».)

```
#DEFINE ppLayoutTitle 1
oPowerPoint = CreateObject("PowerPoint.Application")
oPowerPoint.Visible = .T.
oPresentation = oPowerPoint.Presentations.Add()
oSlide = oPresentation.Slides.Add(1, ppLayoutTitle)
```

На титульном слайде есть два объекта: заголовок и подзаголовок. Таблица 1 сообщает нам, что первая фигура – это заголовок, а вторая – подзаголовок. Легко подумать, что мы можем просто установить свойство Caption фигуры и двигаться дальше, но это не так просто. Помните, что не каждая фигура имеет текст (зачем текст на звуковом объекте?). Таким образом, объектная модель Shape включает объект TextFrame для каждого объекта Shape. Объект TextFrame существует только в том случае, если свойство HasTextFrame объекта Shape имеет значение True. В этом случае текст виден на слайде, поэтому очевидно, что объект TextFrame существует (с другими типами объектов разумно протестировать HasTextFrame). Объект TextFrame содержит объект TextRange, который на самом деле имеет свойство Text, установленное таким образом:

```
oSlide.Shapes[1].TextFrame.TextRange.Text = "Демо-версия " + ;
    "Автоматизации PowerPoint"
oSlide.Shapes[2].TextFrame.TextRange.Text = "Найдите решение!"
```

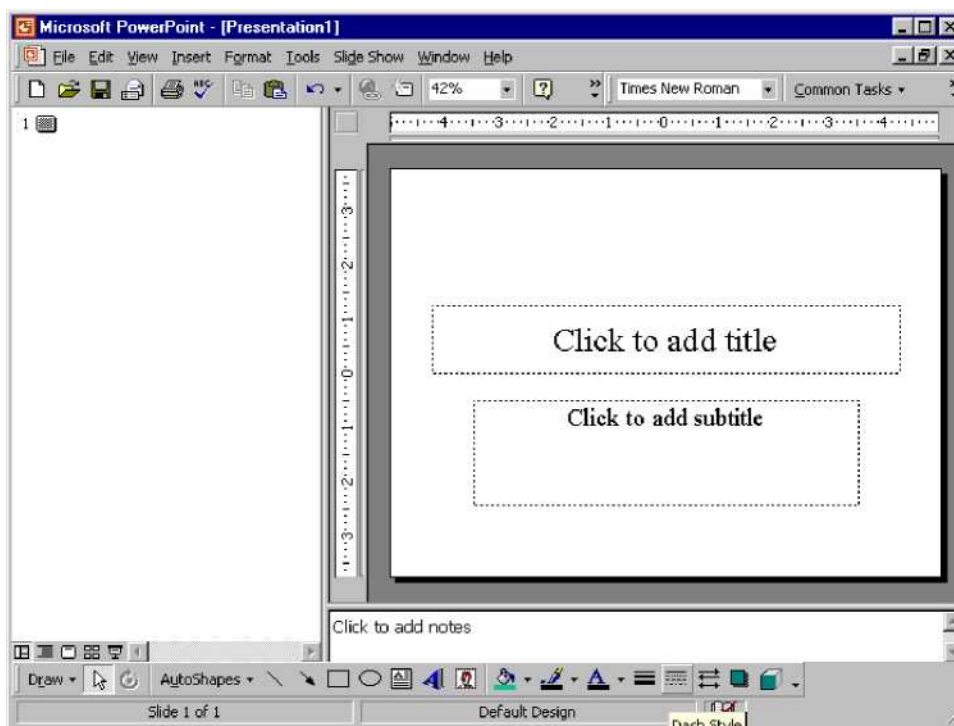


Рисунок 3. Добавляем титульный слайд. Обратите внимание на две фигуры, представляющие заголовок и подзаголовок.

Вуаля! Полный слайд и всего в шести строках кода! Следующий шаг – украсить этот простой слайд. Для этого необходимо лучшее понимание объекта Shape.

Объект Shape

Объект Shape – это ключевой объект для разработчика, автоматизирующего PowerPoint. Он сложнее, чем большинство других, но и более мощный.

Коллекция Shapes поддерживает несколько различных типов фигур. Каждый тип фигуры добавляется своим методом. Обсуждение их всех выходит за рамки этой книги. В справке есть полезная информация по этим вопросам, но в ней отсутствует полный список (кроме раскрывающегося списка «Смотри также», в котором одновременно отображаются только некоторые из них). Мы приводим полный список в Таблице 2 и обсуждаем те, которые вы, скорее всего, будете использовать. Чтобы поддерживать множество типов объектов, объект Shape использует связанные объекты для описания уникальных функций форматирования некоторых фигур. Они перечислены в столбце «Объекты уникального формата» таблицы 2.

Каждый объект Slide также имеет несколько стандартных объектов форматирования. В таблице 3 показаны объекты форматирования, применимые ко всем фигурам.

Таблица 2. Имена методов, используемые для добавления фигур. Для каждого вида формы существует свой особый метод. Некоторые типы используют дополнительные объекты для специального форматирования.

Имя метода	Описание	Уникальные объекты формата
AddCallout	Добавляет фигуру выноски (текстовое поле с линией-выноской, обычно используемой для указания и пояснения чего-либо на слайде).	CalloutFormat
AddComment	Добавляет прямоугольник с цветным фоном. Как выноска, но без линии выноски.	
AddConnector	Добавляет линию или кривую, соединяющую две другие фигуры.	ConnectorFormat
AddCurve	Добавляет кривую Безье из ряда точек.	ShapeNodes
AddLabel	Добавляет текстовую метку (прямоугольник с текстом, но без рамки и заливки).	
AddLine	Добавляет линию.	
AddMediaObject	Добавляет мультимедийный объект из файла.	OLEFormat, LinkFormat
AddOLEObject	Добавляет объект OLE из файла, имени класса или ProgID.	OLEFormat, LinkFormat
AddPicture	Добавляет графику из файла.	OLEFormat, PictureFormat
AddPlaceholder	Возвращает фигуру-заполнитель, если вы ее удалите.	PlaceholderFormat
AddPolyline	Добавляет серию сегментов линий из серии точек.	ShapeNodes

AddShape	Добавляет одну из 140 predeterminedных автофигур.	
AddTable	Добавляет таблицу (только PowerPoint 2000).	
AddTextbox	Добавляет текстовое поле.	
AddTextEffect	Добавляет объект WordArt.	
AddTitle	Восстанавливает заголовок, если вы удалили заполнитель заголовка.	PlaceholderFormat

Таблица 3. Объекты форматирования, относящиеся к каждому объекту Shape. Столбец «Свойство формы» – это имя свойства объекта Shape, используемого для доступа к объекту. Столбец «Тип объекта» относится к типу объекта, на который ссылается свойство; другими словами, тема для поиска в файле справки.

Свойство формы	Тип объекта	Описание
ActionSettings	ActionSettings	Определяет действие, которое будет происходить при наведении указателя мыши на фигуру во время слайд-шоу.
AnimationSettings	AnimationSettings	Определяет специальные эффекты для фигуры во время слайд-шоу.
Fill	FillFormat	Определяет свойства заливки фигуры.
Line	LineFormat	Определяет свойства границы фигуры (или свойства линии, если это линия).
TextFrame	TextFrame	Содержит свойства и методы текста, если фигура содержит текст.

Эти объекты форматирования имеют свойства и методы, определяющие внешний вид объекта. Эти объекты, а также некоторые другие свойства и методы Slide подробно обсуждаются в следующих разделах.

Добавление линий

Линии, пожалуй, самая простая форма для добавления. Список параметров прост: координаты начала и окончания. После добавления строки свойства объекта LineFormat должны быть настроены на форматирование линии, отличное от значения по умолчанию. Объект LineFormat доступен для всех фигур.

Синтаксис добавления объекта Line:

```
oSlide.Shapes.AddLine(nBeginX, nBeginY, nEndX, nEndY)
```

Параметры представляют начальную и конечную координаты в пунктах. Например, чтобы добавить горизонтальную линию в четырех дюймах от верхней части слайда, начиная с одного дюйма с левой стороны и заканчивая девятью дюймами с левой стороны, используйте следующую команду:

```
oLine = oSlide.Shapes.AddLine(72, 288, 648, 288)
```

Поскольку пункт равен 1/72 дюйма, все измерения умножаются на 72. Не очень интуитивно понятно, когда вы вернетесь к этой строке кода в следующем месяце! Следующий код гораздо проще понять и отладить. Вы можете добавить свою собственную константу, например autoIn2Pts, чтобы сделать код более

интуитивным. (Дополнительную информацию см. в примечании.)

```
#DEFINE autoIn2Pts 72
```

```
oLine = oSlide.Shapes.AddLine(1.00 * autoIn2Pts, ;  
    4.00 * autoIn2Pts, 9.00 * autoIn2Pts, 4.00 * autoIn2Pts)
```



Пункт равен 1/72 дюйма. Большая часть объектов PowerPoint измеряется в пунктах, а не в дюймах или сантиметрах. Чтобы сделать ваш код более читабельным, вам, вероятно, следует определить константу, например autoIn2Pts, равной 72. Если вы работаете в метрической системе, вы можете выбрать константу, например autoCm2Pts, и установить для нее значение 28,35. Тогда вы сможете писать код, используя удобные для вас единицы измерения, а не использовать возмутительно большие числа, которые не имеют отношения ни к чему, что вы понимаете.

Вопросы для размышления

Раз уж мы заговорили о пунктах, вам нужно знать, что PowerPoint хранит все измерения в пунктах. Помните, что такие свойства, как Top, Left, Height и Width, не нужно умножать на константы. Например, следующий код рисует линию шириной заголовка и еще одну на полдюйма выше него. Форма заголовка имеет индекс 1 (к сожалению, для объектов Layout нет предопределенных констант; значения индексов для различных фигур см. в Таблице 1):

```
#DEFINE autoIn2Pts 72
```

```
WITH oSlide.Shapes[1]  
    oLine = oSlide.Shapes.AddLine(.Left, .Top - (.5 * autoIn2Pts), ;  
        .Left + .Width, .Top - (.5 * autoIn2Pts))  
ENDWITH
```

Форматирование линий

Тонкие черные линии могут показаться немного скучными. В PowerPoint так много функций форматирования, что нам нужно их изучить! Объекты Line хранят свойства цвета, стиля, прозрачности, узора, веса и стрелок в объекте LineFormat. Используйте свойство Line объекта Line для доступа к объекту LineFormat. В таблице 4 показаны свойства объекта LineFormat.

Таблица 4. Свойства объекта LineFormat.

Свойство	Тип	Описание
ForeColor	Object	Цвет линии с использованием объекта ColorFormat.
BackColor	Object	Задний цвет узорной линии с использованием объекта ColorFormat. Это вторичный цвет линии с узором, и он игнорируется, если узоры не используются.
DashStyle	Numeric	Стиль штриха линии. Использует одну из следующих констант: msoLineDash 4 msoLineDashDot 5 msoLineDashDotDot 6

Свойство	Тип	Описание
		msoLineLongDash 7 msoLineLongDashDot 8 msoLineRoundDot 3 msoLineSolid 1 msoLineSquareDot 2
Pattern	Numeric	Узор, примененный к линии. В качестве фона рисунка используется цветной фон. Используйте одну из многих шаблонных констант. Некоторые из них перечислены здесь: msoPattern50Percent 7 msoPatternLargeConfetti 33 msoPatternLargeGrid 34 msoPatternLightDownwardDiagonal 21 msoPatternLightHorizontal 19 msoPatternLightUpwardDiagonal 22 msoPatternLightVertical 20 msoPatternPlaid 42 msoPatternSmallGrid 23 msoPatternSolidDiamond 39 msoPatternWideDownwardDiagonal 25 msoPatternWideUpwardDiagonal 26 msoPatternZigZag 38
Style	Numeric	Стиль линии, который может выглядеть как несколько линий. msoLineSingle 1 msoLineThickBetweenThin 5 msoLineThickThin 4 msoLineThinThick 3
Transparency	Numeric	Степень прозрачности линии. Значение варьируется от 0,0 (непрозрачный) до 1,0 (полностью прозрачный).
Weight	Numeric	Толщина линии в пунктах.
Visible	Logical	Логическое значение, определяющее, видна ли линия.
BeginArrowhead Length	Numeric	Длина стрелки в начале линии. Используйте одну из следующих констант: msoArrowheadShort 1 msoArrowheadLengthMedium 2

Свойство	Тип	Описание
		msoArrowheadLong 3
BeginArrowhead Style	Numeric	Форма наконечника стрелки в начале линии. Используйте одну из следующих констант: msoArrowheadDiamond 5 msoArrowheadNone 1 msoArrowheadOpen 3 msoArrowheadOval 6 msoArrowheadStealth 4 msoArrowheadTriangle 2
BeginArrowhead Width	Numeric	Ширина стрелки в начале строки. Используйте одну из следующих констант: msoArrowheadNarrow 1 msoArrowheadWidthMedium 2 msoArrowheadWide 3
EndArrowheadLength	Numeric	Длина стрелки в конце линии. Используйте одну из констант, перечисленных в BeginArrowheadLength.
EndArrowheadStyle	Numeric	Форма стрелки в конце линии. Используйте одну из констант, перечисленных в BeginArrowheadStyle.
EndArrowheadWidth	Numeric	Ширина стрелки в конце линии. Используйте одну из констант, перечисленных в BeginArrowheadWidth.

Цвета линии задаются с помощью свойств ForeColor и BackColor. Эти свойства указывают на объект ColorFormat. Объект ColorFormat имеет только два свойства: одно, необходимое здесь, – это свойство RGB (второе – свойство SchemeColor; оно будет рассмотрено позже).



Чтобы проиллюстрировать использование этих свойств, код в листинге 2 генерирует несколько строк на слайде PowerPoint (код PPTLines.PRG в файлах для загрузки для разработчиков, доступных по адресу www.hentzenwerke.com). На рисунке 4 показаны результаты примера кода.

Листинг 2. Пример кода форматирования строк. Существует множество способов форматирования линии: в этом примере показаны узоры, ширина и стрелки. См. рисунок 4, где показан получившийся слайд.

```

* PPTLines.PRG
* Copyright (c)2000, Tamar Granor and Della Martin
* From _Automating Microsoft Office with Visual FoxPro_,
* Hentzenwerke Press. www.hentzenwerke.com
*
* Эта программа демонстрирует возможности форматирования линий.

#DEFINE autoIn2Pts 72
#DEFINE ppLayoutBlank 12
#DEFINE msoLineThickThin 4
#DEFINE msoPatternSmallGrid 23
#DEFINE msoLineDash 4
#DEFINE msoArrowheadLengthMedium 2

```

```
#DEFINE msoArrowheadOval          6
#DEFINE msoArrowheadWidthMedium  2
#DEFINE msoArrowheadLong         3
#DEFINE msoArrowheadTriangle     2
#DEFINE msoArrowheadWide         3

* Удалите все существующие ссылки на серверы.
* Это предотвращает потерю памяти оставшимися экземплярами.
RELEASE ALL LIKE о*

* В демонстрационных целях сделайте оPowerPoint и оSlide
* доступными после выполнения этой программы.
PUBLIC оPowerPoint, оSlide

* Откройте сервер и добавьте презентацию
оPowerPoint = CreateObject("PowerPoint.Application")
оPowerPoint.Visible = .T.
оPresentation = оPowerPoint.Presentations.Add()

* Получить новый слайд
оSlide = оPresentation.Slides.Add(1, ppLayoutBlank)

* Добавьте линию и отформатируйте ее как синюю,
* толщиной 5 пунктов и стиль «Толстый тонкий».
оLine = оSlide.Shapes.AddLine( ;
    1.00 * autoIn2Pts, 2.00 * autoIn2Pts, ;
    9.00 * autoIn2Pts, 2.00 * autoIn2Pts)
WITH оLine.Line && .Line относится к объекту LineFormat.
    .ForeColor.RGB = RGB(0, 0, 255)
    .Weight = 5
    .Style = msoLineThickThin
ENDWITH

* Добавьте линию и отформатируйте ее так,
* чтобы она была красной на желтом,
* толщиной полдюйма и имела небольшой сетчатый узор.
оLine = оSlide.Shapes.AddLine( ;
    1.00 * autoIn2Pts, 3.00 * autoIn2Pts, ;
    9.00 * autoIn2Pts, 3.00 * autoIn2Pts)
WITH оLine.Line && .Line относится к объекту LineFormat.
    .ForeColor.RGB = RGB(255, 0, 0)
    .BackColor.RGB = RGB(255, 255, 0)
    .Weight = .5 * autoIn2Pts
    .Pattern = msoPatternSmallGrid
ENDWITH

* Добавьте линию и добавьте стрелку в начало и конец.
* Отформатируйте начальную стрелку в виде круга,
* а конечную стрелку — в виде длинного широкого
* треугольника. Сделайте линию пунктирной.
оLine = оSlide.Shapes.AddLine( ;
    1.00 * autoIn2Pts, 4.00 * autoIn2Pts, ;
    9.00 * autoIn2Pts, 4.00 * autoIn2Pts)
```

```

WITH oLine.Line  && .Line относится к объекту LineFormat.
.DashStyle = msoLineDash
.BeginArrowheadLength = msoArrowheadLengthMedium
.BeginArrowheadStyle  = msoArrowheadOval
.BeginArrowheadWidth  = msoArrowheadWidthMedium
.EndArrowheadLength   = msoArrowheadLong
.EndArrowheadStyle    = msoArrowheadTriangle
.EndArrowheadWidth    = msoArrowheadWide
ENDWITH

```

Листинг 2 иллюстрирует несколько интересных нюансов форматирования линий. Свойство BackColor актуально только при использовании шаблона и предоставляет цвет фона для заполнения линии. При использовании стрелок начальный наконечник стрелки идет по первой координате X,Y, переданной методу AddLine; конечная стрелка идет по второй координате.

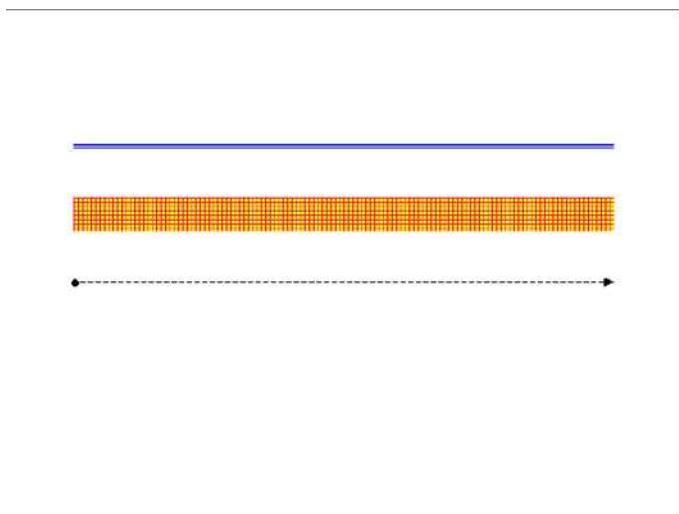


Рисунок 4. Слайд PowerPoint, показывающий форматированные линии, созданные на основе листинга 2.

Если Shape не является линией, объект LineFormat, на который ссылается Line, описывает границы фигуры. В этом случае объект LineFormat имеет все те же свойства, за исключением свойств Arrowhead. Попытка установить для любого свойства стрелки значение, отличное от единицы, приводит к ошибке: «Код исключения OLE IDispatch 0: указанное значение выходит за пределы диапазона».

Добавление автофигур

Office поставляется с более чем 140 предопределенными фигурами для использования в своих приложениях. Доступные фигуры предоставляют гораздо больше, чем просто базовые фигуры рисования, такие как прямоугольники и круги. Существует множество декоративных форм, таких как стрелки, звезды и баннеры. Также доступны символы блок-схемы и многие виды выносок. Любую из этих фигур можно добавить на слайд с помощью метода AddShape коллекции Shapes. Автофигуры используются для создания диаграмм, иллюстраций или просто для украшения слайдов.

Синтаксис добавления автофигуры:

```

oShape = oSlide.Shapes.AddShape(nType, nLeft, nTop,
                                nWidth, nHeight)

```

Первый параметр – это числовая константа, указывающая одну из 140 доступных фигур. В таблице 5 показан образец доступных констант. Обратите внимание, что их

префикс – «mso», что означает, что они доступны для всех приложений Office, а не только для PowerPoint. Следующие два параметра определяют верхний левый угол прямоугольного поля, содержащего фигуру (разумеется, в пунктах). Последние два параметра определяют ширину и высоту объекта в пунктах. Это отличается от линии, для которой требуется абсолютная конечная точка, а не ширина и высота. Эта прямоугольная рамка, содержащая фигуру, называется *ограничивающей рамкой*.

Таблица 5. Выборка констант AutoShape и их значений.

Константа AutoShape	Значение
msoShape5PointStar	92
msoShapeArc	25
msoShapeBalloon	137
msoShapeCube	14
msoShapeDownArrow	36
msoShapeLeftArrow	34
msoShapeLineCallout1	109
msoShapeNoSymbol	19
msoShapeOval	9
msoShapeParallelogram	2
msoShapeRectangle	1
msoShapeRightArrow	33
msoShapeRoundedRectangle	5
msoShapeUpArrow	35

Фигура размещается с текущими настройками AutoShape по умолчанию. Встроенные настройки по умолчанию имеют характерный оттенок зеленого цвета морской пены с тонкой черной рамкой. Мы обсудим изменение этих прекрасных цветов по умолчанию в разделе «Заливка фигуры» далее в этой главе. Итак, выдаем следующее:

```
#DEFINE msoShape5PointStar 92
#DEFINE autoIn2Pts 72
```

```
oShape = oSlide.Shapes.AddShape(msoShape5PointStar, ;
    1 * autoIn2Pts, 1 * autoIn2Pts, ;
    4 * autoIn2Pts, 4 * autoIn2Pts)
```

рисует 4-дюймовую звезду с верхним левым углом ее ограничивающей рамки на расстоянии одного дюйма от верха и сбоку. Результат см. на **рисунке 5** (на пустом слайде).

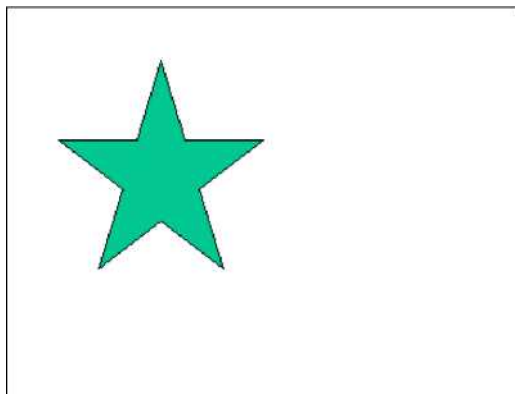


Рисунок 5. Результаты добавления автофигуры звезды. Фигура размещается с текущими настройками автофигуры по умолчанию, включая цвет заливки, цвет границы и стиль границы.

Форматирование границы формы

Поскольку мы знаем, как форматировать линии, мы можем понять, как форматировать границы. Свойство Shape объекта Line (которое ссылается на объект LineFormat) управляет его границей. Итак, если вы очень хотите, вы можете поставить вокруг звезды пунктирную темно-синюю рамку шириной в 5 пунктов:

```
#DEFINE msoLineDash 4

WITH oShape.Line
  .ForeColor.RGB = RGB(0, 0, 128)
  .Weight = 5
  .DashStyle = msoLineDash
ENDWITH
```

Конечно, вы можете заставить границу исчезнуть, установив для ее свойства Visible значение False:

```
oShape.Line.Visible = .F.
```

Заполнение формы

Теперь нужно что-нибудь сделать с этим чудесным цветом морской пены. Для этого требуется свойство Fill, которое ссылается на объект FillFormat. Свойства и методы объекта FillFormat делают его чрезвычайно мощным. Вы можете не только изменить цвет морской пены на любой другой отображаемый сплошной цвет, но и изменить его на градиентную заливку, текстуру, узор, изображение или даже сделать его полупрозрачным. Объект FillFormat подробно описан в разделе «Достижение согласованности с мастер-слайдами» в главе 11. Но нам нужно что-то сделать с этим цветом.

Как и при установке линии, свойство ForeColor управляет цветом. Изменить цвет звезды так же просто, как:

```
oShape.Fill.ForeColor.RGB = RGB(255, 255, 0)  && желтый
```

Теперь он желтый. Если вы хотите создать узор, вам нужно установить свойство BackColor, а затем использовать метод Patterned, чтобы применить узор, используя те же константы шаблона, что и для линий. Этот код устанавливает цвет фона на темно-синий и шаблон msoPatternSmallGrid. Результаты показаны на рисунке 6.

```
#DEFINE msoPatternSmallGrid 23
WITH oShape.Fill
```



```
.BackColor.RGB = RGB(0, 0, 128)  && темно-синий  
.Patterned(msoPatternSmallGrid)  && 23  
ENDWITH
```

Ладно, это не самая привлекательная модель. Но это отличный пример того, почему нам может понадобиться отменить шаблон. Очевидным подходом, конечно, является использование шаблонного метода, но очевидный подход не всегда является правильным. Чтобы удалить узор, используйте метод `Solid`, чтобы вернуть его обратно:

```
oShape.Fill.Solid()
```

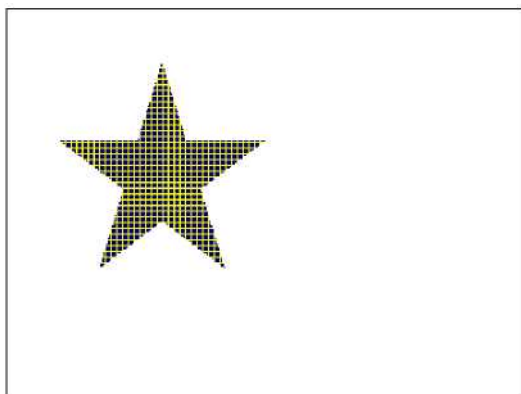


Рисунок 6. Главные результаты нанесения узоров. `ForeColor` используется как цвет самого узора, а `BackColor` заполняет область позади узора.

Берегитесь! Хотя объекты `FillFormat` и `LineFormat` имеют схожие свойства, объект `LineFormat` не имеет метода `Solid`. Чтобы удалить узор из линии, установите для ее свойства `DashStyle` значение `msoLineSolid` (значение 1).

Добавление и форматирование текста

Текст можно добавить к любой фигуре. Есть свойства для установки цвета и шрифта текста. Отдельные символы, слова, предложения и даже абзацы можно форматировать по-разному, например выделяя определенные слова жирным шрифтом или выделяя определенные строки. Вы также можете отформатировать текст в виде маркированного списка, при этом для маркеров доступен широкий выбор форматов.

Добавление текста

Добавление текста в фигуры аналогично добавлению текста в заголовок, тем более что заголовки на самом деле представляют собой просто прямоугольные фигуры. Быстрый обзор процесса напоминает нам, что свойство `TextFrame` объекта `Shape` содержит объект `TextFrame`. Объект `TextFrame` содержит свойства и методы для выравнивания и привязки текстового фрейма, а также для хранения и форматирования текста в текстовом фрейме.

Среди свойств объекта `TextFrame` свойство `TextRange` используется для хранения объекта `TextRange`. Объект `TextRange` содержит свойства, хранящие текст и его форматирование, а также ряд методов для управления текстом. Мы вернемся к объяснению свойств `TextFrame` после того, как рассмотрим объект `TextRange` — гораздо проще увидеть, как свойства `TextFrame` влияют на текст, когда на самом деле есть текст для просмотра!

В объекте `TextRange` сама текстовая строка хранится в свойстве `Text`. Информация о форматировании шрифта хранится в свойстве `Font` `TextRange`, а сохраненный объект `Font` аналогичен объектам `Font`, встречающимся в Word и Excel. В

следующем примере добавляется овал и показано, как добавить текстовую строку и использовать базовое форматирование шрифта для фигуры:

```
#DEFINE msoShapeOval 9
#DEFINE autoIn2Pts 72

oShape = oSlide.Shapes.AddShape(msoShapeOval, ;
    4.0 * autoIn2Pts, 1.5 * autoIn2Pts, ;
    2.0 * autoIn2Pts, 4 * autoIn2Pts)
WITH oShape.TextFrame.TextRange
    .Text = "Это тест." + CHR(13) + "Это только тест."
    .Font.Name = "Arial"
    .Font.Size = 36
    .Font.Bold = .T.
ENDWITH
```

Добавленная форма представляет собой высокий тонкий овал. На рисунке 7 показаны результаты кода. Обратите внимание, что границы фигуры не влияют на размер и ширину текста. Текст намного шире, чем форма. Объект TextFrame имеет свойство AutoSize, которое позволяет изменять размер фигуры в соответствии с текстовой строкой. Установка для свойства AutoSize значения ppAutoSizeShapeToFitText (1) автоматически изменяет размер фигуры в соответствии с текстом. На рисунке 8 показано, как меняется форма примера, когда AutoSize настроен на соответствие тексту. Даже если для AutoSize немедленно установлено значение ppAutoSizeNone (0), автоматически сгенерированный размер останется; он не возвращается к предыдущему размеру.

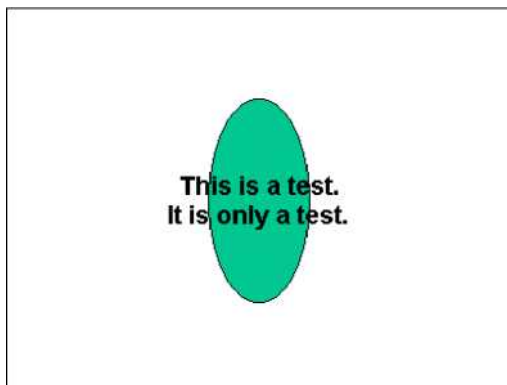


Рисунок 7. Результаты форматированного текста, добавленного в овальную форму.

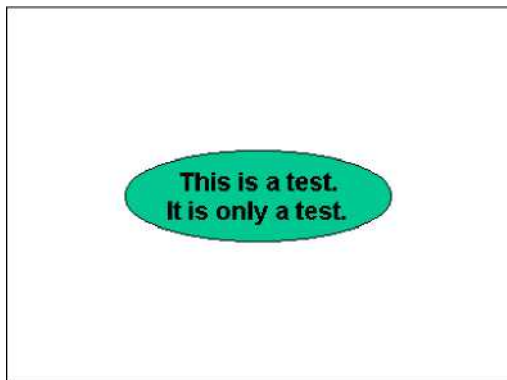


Рисунок 8. Результаты функции AutoSize.

Поиграйтесь с текстовыми сегментами

Объект TextRange в Office предоставляет нам богатый набор методов для работы с

сегментами текста. Эти сегменты обычно известны как символы, слова, строки, предложения, абзацы и фрагменты. Выбрав определенные сегменты, например несколько слов или предложение, выделенный текст можно отформатировать иначе, чем остальной текст. Хотя символы, слова, предложения и абзацы кажутся очевидными, строки и фрагменты нуждаются в дополнительных пояснениях. Строки соответствуют физической строке в текстовом фрейме (здесь вспомните MLINE()). Фрагменты соответствуют тексту с идентичными атрибутами форматирования.

Сегменты текста выбираются с помощью методов TextRange, называемых Characters, Words, Lines, Sentences, Paragraphs и Runs. Каждый метод принимает два необязательных параметра. Первый – это позиция начального фрагмента, а второй – количество сегментов, которые вы хотите вернуть (по умолчанию – 1). Если вы опустите какой-либо параметр, вы получите их все. Все эти методы возвращают другой объект TextRange для манипуляций. Чтобы увидеть текст, вам необходимо сослаться на свойство Text объекта TextRange. Вот несколько примеров, основанных на тексте, показанном на рисунке 8.

```
&& "Это только тест."  
? oShape.TextFrame.TextRange.Lines[2].Text  
&& "тест " (обратите внимание на пробел)  
? oShape.TextFrame.TextRange.Words[2].Text  
&& "." (Точки – это слова)  
? oShape.TextFrame.TextRange.Words[3].Text  
&& "о тест"  
? oShape.TextFrame.TextRange.Characters[3,6].Text
```

Поскольку эти методы возвращают TextRanges, вы можете объединить эти методы, чтобы получить очень конкретный сегмент, например третье слово во втором предложении:

```
? oShape.TextFrame.TextRange.Sentences[2].Words[3].Text && "тест"
```

Это полезно, когда у клиента название компании отображается другим шрифтом или если слова или предложения должны быть выделены. Это также полезно для форматирования абзацев.

Форматирование абзацев

Если вы говорите: «На слайде PowerPoint не должно быть абзацев!» похлопайте себя по спине! Вы понимаете, что такое хороший макет презентации! Для тех из вас, кто ломает голову, задаваясь вопросом, почему, практическое правило – «семь строк по семь слов». Обычно это тоже считается максимумом. Аудитория не должна тратить время на чтение большого количества текста на экране, потому что в этом случае они не слушают ведущего. (В этом отношении докладчику тоже не следует зачитывать слайд аудитории, но мы отвлеклись.)

Так зачем же PowerPoint включает объект ParagraphFormat? Полиморфизм. Имя соответствует объекту, который выполняет аналогичные функции в других продуктах Office, и позволяет легко форматировать маркированные списки. (Думайте о каждом маркере как об абзаце.) Кроме того, он форматирует абзац, если вы (или ваши клиенты) настаиваете на абзацах текста.

Объект ParagraphFormat имеет несколько свойств, содержащих значения для форматирования всего текстового диапазона. Свойство Alignment задает выравнивание всего текстового диапазона. Константы выравнивания: ppAlignLeft (1), ppAlignCenter (2), ppAlignRight (3), ppAlignJustify (4) и ppAlignDistribute (5). Также доступно свойство WordWrap, которое является логическим значением.

Существуют свойства, которые устанавливают вертикальный интервал для каждого абзаца в TextFrame. Используйте числовые свойства SpaceBefore, SpaceAfter и SpaceWithin, чтобы задать интервал между строками текста. Объем пространства задается соответствующим свойством LineRule: LineRuleBefore, LineRuleAfter и LineRuleWithin. Свойства LineRule могут быть установлены с логическим или числовым значением, указывающим, выражаются ли единицы в количестве строк (.T. или 1) или в пунктах (.F. или 0). Для установки значения могут использоваться как логические, так и числовые значения; однако при запросе эти свойства являются числовыми. Задайте свойства LineRule явно перед изменением свойств Space, поскольку изменение свойств LineRule с пунктов на строки правильно преобразует сохраненное значение Space, но изменение со строк на пункты устанавливает соответствующее свойство Space в ноль.

Маркеры

Объект ParagraphFormat использует свойство Bullet для хранения объекта BulletFormat. Объект BulletFormat задает все форматирование маркеров. PowerPoint 2000 предлагает несколько приятных обновлений объекта BulletFormat, наиболее заметным из которых является свойство Type. Объект Type определяет, какие типы маркеров используются. Если маркеры использоваться не должны, задайте для свойства Type значение ppBulletNone (0). Чтобы отобразить символы для маркеров, используйте константу ppBulletUnnumbered (1). Для отображения нумерованных маркеров используйте константу ppBulletNumbered (2). Чтобы отобразить графическое изображение, например BMP, в виде маркера, используйте константу ppBulletPicture (3).



В PowerPoint 97 объект BulletFormat не поддерживает нумерованные маркеры или изображения для маркеров. В результате остаются символические маркеры или нет маркеров. Объект BulletFormat в PowerPoint 97 использует объект Visible для переключения отображения символов маркеров. Свойство Visible по-прежнему доступно в PowerPoint 2000, но оно не указано конкретно в файле справки в разделе «Свойства BulletFormat» или в списке «Применяется к» свойства Visible (хотя оно включено в несколько примеров в файле справки). Обычно это означает, что свойство может быть недоступно в следующей версии PowerPoint. Убедитесь, что ваш код использует свойство Type, а не свойство Visible при работе в PowerPoint 2000.

Управление размером маркеров осуществляется с помощью свойства RelativeSize. Числовое значение должно находиться в диапазоне от 0,25 до 4. Это число указывает размер в процентах от высоты текста. Если вы введете значения больше или меньше ожидаемого диапазона, ошибки не произойдет; вместо этого он отображает символ так, как если бы для него было установлено минимальное или максимальное значение (в зависимости от того, какое значение было превышено). Это свойство не изменилось по сравнению с PowerPoint 97.

Свойство Character устанавливает символ, используемый для маркера. Согласно файлу справки, это свойство хранит значение Юникода для символа. Используйте значение Unicode только для шрифтов Unicode, таких как Lucida Sans Unicode. Если для свойства Font объекта Bullet задан обычный шрифт ASCII, например Wingdings, используйте значение ASCII символа.

Нумерованные маркеры

PowerPoint 2000 предлагает ряд свойств для нумерации маркеров. Свойство Style

объекта BulletFormat выбирает один из примерно 28 предустановленных стилей нумерации (не все доступны на каждом языке). Стили нумерации предусматривают использование маркированных символов в определенном регистре и с указанными круглыми скобками или точками. Конечно, есть ряд констант (несколько примеров показаны в Таблице 6) – полный список есть в файле справки (это одна из немногих страниц справки, на которой также показаны значения констант).

Таблица 6. Константы для стилей нумерации маркеров. Нумерованные маркеры – это новая функция PowerPoint 2000.

Константа	Значение	Пример
ppBulletAlphaLCPeriod	0	a.
ppBulletAlphaUCPeriod	1	A.
ppBulletArabicParenRight	2	1)
ppBulletArabicPeriod	3	1.
ppBulletRomanLCParenBoth	4	(i)
ppBulletAlphaUCParenRight	11	A)
ppBulletArabicPlain	13	1
ppBulletCircleNumWDWhitePlain	19	① (Доступно только для 1-10)

Значением стиля по умолчанию является ppBulletAlphaUCPeriod (1), который представляет собой буквенные символы в верхнем регистре, за которыми следует точка.

Свойство StartValue содержит номер первого маркера на слайде. Это особенно полезно, если маркированные списки разбиваются на слайды, например, когда первые три маркера находятся на слайде 1, а вторые три – на слайде 2. Возможно, вы захотите разбить пронумерованные маркеры на два столбца на одном слайде и иметь две текстовые фигуры, пронумерованные последовательно на одном слайде. StartValue – числовое свойство. Если в стиле маркера отображаются цифры, значение напрямую соответствует номеру, отображаемому в маркере. Сюда входят римские цифры. Если в стиле присутствуют буквенные символы, значение соответствует позиции в алфавите: 1 = A, 2 = B, 26 = Z, 27 = AA, 28 = BB и т. д. Свойство StartValue имеет диапазон от 1 до 32767. Значение по умолчанию – 1.

При установке StartValue используется текущий сохраненный стиль. Поскольку по умолчанию используются буквенные символы в верхнем регистре, не удивляйтесь, если установите для StartValue значение 3 и получите «С». вместо «3». Обязательно задайте явно свойство Style при установке StartValue, чтобы избежать подобных сюрпризов.



Установка значений для Style или StartValue автоматически устанавливает для свойства Type значение ppBulletNumbered (2). Используйте свойство Type, чтобы проверить, есть ли пронумерованные маркеры; не полагайтесь только на свойства Style или StartValue. Эти свойства сохраняют свои настройки, даже если вы выберете другой тип маркера.

Маркер-картинка

PowerPoint 2000 включает новую функцию, позволяющую добавлять графический файл в качестве символа маркера. Это действительно может украсить презентации. Для добавления графического файла в виде растрового изображения используется единственный метод объекта BulletFormat – метод Picture. Он принимает один параметр, который представляет собой полный путь к файлу (или, по крайней мере, путь относительно местоположения PowerPoint; помните, PowerPoint не знает, какие текущие каталоги FoxPro). PowerPoint поддерживает широкий спектр графических форматов; полный список см. в разделе «Метод изображения» в файле справки.

При запуске метода Picture для свойства Type устанавливается значение ppBulletPicture (3). Похоже, что нет открытого свойства для запроса того, какой графический файл используется.

Вы можете установить изображение для каждого маркера индивидуально, используя метод TextRange Paragraphs. Хотя следующий код технически возможен, убедитесь, что он приемлем визуально (этот конкретный пример иллюстрирует технические методы, *но* не рекомендуется в качестве примера хорошей компоновки!).

```
WITH oShape.TextFrame.TextRange
    * Убедитесь, что фигура содержит два абзаца текста.
    .Text = "Test 1" + CHR(13) + "Test 2"

    * Используйте некоторые стандартные растровые
    * изображения, поставляемые с Windows
    .Paragraphs(1).ParagraphFormat.Bullet.Picture( ;
        GETENV("WINDIR") + "\\Triangles.bmp")
    .Paragraphs(2).ParagraphFormat.Bullet.Picture( ;
        GETENV("WINDIR") + "\\Circles.bmp")
ENDWITH
```

Веселье со шрифтами

Если вы занимаетесь разработкой приложений какое-то время, вам, вероятно, это понятно: вы только что продемонстрировали свою действительно потрясающую презентацию, которая извлекает данные из таблиц, манипулирует ими дюжиной способов, превращает их в невероятно хорошо сделанную презентацию, содержащую 20 слайдов, за две секунды – это совершенно потрясающе. Увидев это невероятное технологическое достижение, клиент/начальник задумчиво делает паузу, прежде чем сказать: «Хммм, можем ли мы изменить шрифт на что-нибудь другое?» После того, как вы поднимете с пола свое ушибленное и побитое эго (ведь вы ожидали комментария типа «Ух ты!»), вы можете с уверенностью сказать им, что да, есть способ изменить шрифт. И цвет тоже (попытайтесь не показаться слишком саркастичным, когда говорите это сквозь зубы!).

Оба объекта BulletFormat и TextRange имеют свойство Font, в котором хранится объект Font. Этот объект Font аналогичен объектам Word и Excel. Большинство свойств являются логическими свойствами, которые задают такие свойства, как жирный шрифт, курсив, подчеркивание и т. д. Также имеется свойство Size, которое задает размер текста в пунктах. На рисунке 9 показан слайд с различными атрибутами текста (с использованием шрифта Times New Roman), а в листинге 3 показан код, создавший этот слайд. Он сохраняется как PPTFont.PRG в файлах загрузки для разработчиков, доступных по адресу www.hentzenwerke.com.

Этот код показывает различные свойства объекта Font, а также способы

управления текстовыми сегментами и некоторые функции ParagraphFormat.

- Regular
- **Bold**
- *Italic*
- Shadow
- Underline
- Subscript
- Superscript
- Emboss

Рисунок 9. Примеры объектов Font с использованием кода, показанного в листинге 3. При желании эти восемь эффектов также можно комбинировать.

Листинг 3. Управление объектом Font.

```
* PPTFont.PRG
* Copyright (c)2000, Tamar Granor and Della Martin
* From _Automating Microsoft Office with Visual FoxPro_,
* Hentzenwerke Press. www.hentzenwerke.com
*
#DEFINE ppLayoutBlank      12
#DEFINE msoShapeRectangle  1
#DEFINE ppAlignLeft        1
#DEFINE ppBulletUnnumbered 1

* ПРЕДПОЛОЖЕНИЕ: oPresentation — это переменная,
* указывающая на открытую презентацию PowerPoint.

* Добавьте новый слайд с новой фигурой
oSlide = oPresentation.Slides.Add(1, ppLayoutBlank)
oShape = oSlide.Shapes.AddShape(msoShapeRectangle,100,100,500,350)

* Удалите границу фигуры и залейте ее цветом.
oShape.Line.Visible = .F.
oShape.Fill.ForeColor.RGB = RGB(255,255,255)

WITH oShape.TextFrame.TextRange

* Добавьте пример текста
.Text = "Regular" + CHR(13) + ;
       "Bold" + CHR(13) + ;
       "Italic" + CHR(13) + ;
       "Shadow" + CHR(13) + ;
       "Underline" + CHR(13) + ;
       "Subscript" + CHR(13) + ;
       "Superscript" + CHR(13) + ;
       "Emboss"

* Увеличьте шрифт, выровняйте текст по левому краю и
* установите для формата маркеров символы по умолчанию.
.Font.Size = 32
.ParagraphFormat.Alignment = ppAlignLeft
```



```
.ParagraphFormat.Bullet.Type = ppBulletUnnumbered
* Отформатируйте каждый из маркеров
.Paragraphs[2].Font.Bold = .T.
.Paragraphs[3].Font.Italic = .T.
.Paragraphs[4].Font.Shadow = .T.
.Paragraphs[5].Font.Underline = .T.
.Paragraphs[6].Characters[1,3].Font.Subscript = .T.
.Paragraphs[7].Characters[1,5].Font.Superscript = .T.
.Paragraphs[8].Font.Emboss = .T.
```

ENDWITH

У объекта Font также есть свойство Color, которое ссылается на объект ColorFormat. Свойство RGB ColorFormat задает цвет текста. Цвет фона задается формой. В следующем примере текст заголовка задается синим цветом. Если в заголовке был символ маркера, свойство Font.Color задает цвет текста и маркера.

```
oSlide.Shapes.Title.TextFrame.TextRange.Color = RGB(0,0,128)
```

Иногда вам может потребоваться установить цвет маркера отдельно от цвета текста. Для этого установите свойство TextRange.ParagraphFormat.Bullet.Font.Color.RGB. При этом логическому свойству TextRange.ParagraphFormat.Bullet.UseTextColor присваивается значение .F. Хотя вы не можете явно установить UseTextColor в .F., вы можете установить его в .T. чтобы вернуть цвет маркера к цвету текста.

Свойство Name задает шрифт. Установите для него имя шрифта точно так же, как вы задаете свойства имени шрифта для объектов FoxPro. Шрифт должен существовать на этом компьютере, или он выбирает шрифт (обычно Times New Roman). Как и в случае с цветом, вы можете установить шрифт маркеров отдельно от текста. Это делается аналогичным образом путем установки свойства TextRange.ParagraphFormat.Bullet.Font.Name (в отличие от TextRange.Font.Name), которое затем устанавливает для TextRange.ParagraphFormat.Bullet.UseTextFont значение .F. Вы не можете явно установить для UseTextFont значение .F., но вы можете установить для него значение .T. чтобы вернуть шрифт маркера в текстовый шрифт.

10.6 Делаем это презентабельно

Как только слайды будут созданы, пользователи захотят что-то с ними сделать. Они, вероятно, захотят запустить слайд-шоу или распечатать его. К счастью, PowerPoint предоставляет некоторые методы, позволяющие выполнять некоторые довольно интересные действия.

Запуск режима слайд-шоу

После того как вы создадите презентацию, ваши пользователи могут захотеть просмотреть ее на экране в виде слайд-шоу. Свойство SlideShowSettings объекта Presentation ссылается на объект SlideShowSettings, который позволяет управлять презентацией. Единственный метод – это метод Run, который запускает слайд-шоу:

```
oPresentation.SlideShowSettings.Run()
```

Эта команда сообщает PowerPoint начать слайд-шоу, которое становится самым верхним окном. Ваше приложение скрывается за слайд-шоу. Когда пользователи завершают показ слайдов, они возвращаются в PowerPoint, а не в ваше приложение! Кроме того, ваше приложение продолжает работать во время показа слайдов. Чтобы справиться с этим корректно, убедитесь, что следующая строка вашей

программы находится в состоянии ожидания, чтобы ваше приложение не продолжало работать, пока они смотрят презентацию PowerPoint. Вероятно, вам также понадобится код, который вернет ваше приложение на первое место после завершения показа.

Печать

В Presentation есть свойство PrintOptions, которое указывает на объект PrintOptions. Свойства PrintOptions обобщены в Таблице 7. После установки параметров печати выполните метод PrintOut для печати выбранных элементов.

Таблица 7. Свойства объекта PrintOptions.

Свойство	Тип	Описание
Collate	Logical	Логическое значение, указывающее, следует ли объединять несколько копий.
FitToPage	Logical	Значение True, чтобы слайды заполняли страницу; Значение False, чтобы учитывать значения в диалоговом окне «Параметры страницы».
FrameSlides	Logical	True, чтобы разместить тонкую рамку вокруг слайдов, заметок и раздаточных материалов; False, чтобы пропустить это.
HandoutOrder	Numeric	Устанавливает порядок слайдов в раздаточном материале: ppPrintHandoutHorizontalFirst (1) – Печатает слайды рядами по всей странице. ppPrintHandoutVerticalFirst (2) – Печатает слайды столбцами внизу страницы. По умолчанию – 1; установите необходимое количество копий.
OutputType	Numeric	Числовое значение, соответствующее тому, что печатается (слайды, раздаточные материалы или заметки): ppPrintOutputSlides (1) По умолчанию. ppPrintOutputTwoSlideHandouts (2) – Два слайда на странице. ppPrintOutputThreeSlideHandouts (3) – Три слайда на странице. ppPrintOutputFourSlideHandouts (8) – Четыре слайда на странице. ppPrintOutputSixSlideHandouts (4) – Шесть слайдов на странице. ppPrintOutputNineSlideHandouts (9) – Девять слайдов на странице. ppPrintOutputOutline (6) – Формат структуры. ppPrintOutputNotesPages (5) – Только

		примечания.
PrintColorType	Numeric	Числовое значение, указывающее, как должны печататься цвета: ppPrintColor (1) – По умолчанию. ppPrintBlackAndWhite (2) – Оттенки серого. ppPrintPureBlackAndWhite (3) – Строго черно-белый (без заливки).
PrintFontsAsGraphics	Logical	True для печати шрифтов как графического изображения.
PrintHiddenSlides	Logical	True для печати всех скрытых слайдов.
PrintInBackground	Logical	True для печати в фоновом режиме (по умолчанию).

После установки свойств вызовите метод Printout. Все параметры являются необязательными. Синтаксис метода Printout следующий:

```
oPresentation.PrintOut(nFromSlide, nToSlide, cPrintToFile,
                      nCopies, lCollate)
```

Параметры nFromSlide и nToSlide задают первый и последний слайды для печати. Параметр cPrintToFile – это полный путь и имя файла для приема вывода. Остальные параметры переопределяют параметры объекта PrintOptions: nCopies перезаписывает NumberOfCopies, а lCollate переопределяет Collate.

10.7 Собираем все вместе



Листинг 4 показывает несколько слайдов маркетинговой презентации для тасманских трейдеров. На слайдах показано, что рассматривается в этой главе. Программа создает три слайда, показанных на рисунках 10, 11 и 12.

Листинг 4. Образец презентации для тасманских трейдеров.

```
* PPTSample1.PRG
* Copyright (c)2000, Tamar Granor and Della Martin
* From _Automating Microsoft Office with Visual FoxPro_,
* Hentzenwerke Press. www.hentzenwerke.com

* Это пример «Собираем все вместе» из главы 10.
* Образец презентации для тасманских трейдеров.
CLOSE DATA

#DEFINE ppLayoutTitle          1
#DEFINE ppLayoutText          2
#DEFINE ppBulletArabicPeriod  3
#DEFINE msoLineThickBetweenThin 5
#DEFINE autoIn2Pts            72

* Удалите все существующие ссылки на серверы.
* Это предотвращает потерю памяти оставшимися экземплярами.
RELEASE ALL LIKE o*
```

* В демонстрационных целях сделайте oPowerPoint и oPresentation
* доступными после выполнения этой программы.

```
PUBLIC oPowerPoint, oPresentation
```

```
SET PATH TO _SAMPLES + "\\TasTrade\\Data\\"
```

* Настройте данные

```
OPEN DATABASE "TasTrade"
```

```
LogoFile = _SAMPLES + "\\TasTrade\\Bitmaps\\TTradeSm.bmp"
```

```
USE OrdItems IN 0
```

```
USE Products IN 0
```

* Выберите 5 самых продаваемых товаров всех времен

```
SELECT TOP 5 ;
```

```
    P.English_Name, ;
```

```
    SUM(O.Unit_Price * O.Quantity) AS TotQuan ;
```

```
FROM OrdItems O, Products P ;
```

```
WHERE O.Product_ID = P.Product_ID ;
```

```
GROUP BY 1 ;
```

```
ORDER BY 2 DESC;
```

```
INTO CURSOR TopSellers
```

* Выберите количество продуктов

```
SELECT Count(*) ;
```

```
FROM Products ;
```

```
INTO ARRAY aProducts
```

* Открыть PowerPoint

```
oPowerPoint = CreateObject("PowerPoint.Application")
```

```
oPowerPoint.Visible = .T.
```

* Создать презентацию.

```
oPresentation = oPowerPoint.Presentations.Add()
```

```
SlideNum = 1
```



Tasmanian Traders

Welcomes you...

Рисунок 10. Образец титульного слайда «Тасманийские трейдеры». На этом слайде показано добавление графики и изменение атрибутов текста.

```

* ТИТУЛЬНЫЙ СЛАЙД .....
* Добавить слайд
oSlide = oPresentation.Slides.Add(SlideNum, ppLayoutTitle)

* Установите текст заголовка.
* Измените его шрифт на Arial, синий и жирный.
WITH oSlide.Shapes[1].TextFrame.TextRange
    .Text = "Tasmanian Traders"
    WITH .Font
        .Name = "Arial"
        .Bold = .T.
        .Color = RGB(0, 0, 128)
    ENDWITH
ENDWITH

* Установите текст подзаголовка.
oSlide.Shapes[2].TextFrame.TextRange.Text = "Welcomes you..."

* Добавьте логотип.
oSlide.Shapes.AddPicture(LogoFile, .F., .T., ;
    2.0 * autoIn2Pts, 1.5 * autoIn2Pts)

* Для пользователей PowerPoint '97: последние два
* параметра, высота и ширина, не являются обязательными.
* Вместо этого используйте этот код:
*.Shapes.AddPicture(LogoFile, .F., .T., ;
*     8.5 * autoIn2Pts, 6.0 * autoIn2Pts, ;
*     1.0 * autoIn2Pts, 1.0 * autoIn2Pts)

SlideNum = SlideNum + 1

```

Tasmanian Traders has what you need

With a selection of **77** items, you're
sure to be pleased.

Рисунок 11. Пример Тасманские трейдеры - второй слайд. На этом слайде показано добавление строк и изменение атрибутов сегмента текста.

```

* ВТОРОЙ СЛАЙД .....
* Добавить слайд
oSlide = oPresentation.Slides.Add(SlideNum, ppLayoutTitle)

* Переместите слайд на передний план.
oSlide.Select()

```

```
* Пользователям PowerPoint '97: oSlide.Select() выдаст ошибку.
* Вместо этого используйте эту строку:
* oPresentation.ActiveWindow.View.GoToSlide(2)

* Установите текст заголовка.
WITH oSlide.Shapes[1].TextFrame.TextRange
  .Text = "Tasmanian Traders " + CHR(13) + "has what you need"
  WITH .Font
    .Name = "Arial"
    .Bold = .T.
    .Color = RGB(0,0,128)
  ENDWITH
ENDWITH

* Переместите заголовок примерно на полдюйма вверх.
WITH oSlide.Shapes(1)
  .Top = .Top - (.5 * autoIn2Pts)
ENDWITH

* Добавьте линию на полдюйма ниже заголовка по центру и длиной
* 6 дюймов
LineTop = oSlide.Shapes[1].Top + oSlide.Shapes[1].Height + ;
  (.5 * autoIn2Pts)
LineLeft = 2 * autoIn2Pts
LineEnd = LineLeft + (6.0 * autoIn2Pts)

oLine = oSlide.Shapes.AddLine(LineLeft, LineTop, LineEnd, LineTop)

* Отформатируйте линию как красную и сделайте ее толстой
* с двумя тонкими линиями с каждой стороны.
WITH oLine.Line
  .ForeColor.RGB = RGB(255,0,0)
  .Style = msoLineThickBetweenThin
  .Weight = 8
ENDWITH

* Задайте текст субтитров и измените шрифт 5-го слова на
* жирный и красный.
WITH oSlide.Shapes[2].TextFrame.TextRange
  .Text = "With a selection of " + ALLTRIM(STR(aProducts[1])) + ;
    " items, you're sure to be pleased."
  .Words[5].Font.Bold = .T.
  .Words[5].Font.Color = RGB(255, 0, 0)
ENDWITH

SlideNum = SlideNum + 1
```

Tasmanian Traders Top Sellers

1. Tibetan Barley Beer -- \$1,922,889
2. Côte de Blaye (Red Bordeaux wine) -- \$186,667
3. Thüringer Sausage -- \$114,924
4. Courdavault Raclette Cheese -- \$88,100
5. Pierrot Camembert -- \$59,982

Рисунок 12. Пример Тасманские трейдеры - третий слайд. На этом слайде показано добавление маркированных списков.

```
* СЛАЙД ТОП-5 ПРОДАВЦОВ .....
* Добавить слайд
oSlide = oPresentation.Slides.Add(SlideNum, ppLayoutText)

* Переместите слайд на передний план
oSlide.Select()

* Пользователям PowerPoint '97: oSlide.Select() выдаст ошибку.
* Вместо этого используйте эту строку:
* oPresentation.ActiveWindow.View.GoToSlide(2)

* Вставьте заголовок (обратите внимание на использование
* объекта Title вместо перечисляемого объекта формы).
* Сделайте шрифт Arial синим и жирным.
WITH oSlide.Shapes.Title.TextFrame.TextRange
    .Text = "Tasmanian Traders" + CHR(13) + "Top Sellers"
    WITH .Font
        .Name = "Arial"
        .Bold = .T.
        .Color = RGB(0,0,128)
    ENDWITH
ENDWITH

* Создайте строку, которая будет использоваться для 5 лучших
* продавцов.
* Используйте CR между каждым элементом, чтобы сделать каждый
* маркер отдельным.
BulletString = ""
SELECT TopSellers
SCAN
    BulletString = BulletString + ;
                        TRIM(TopSellers.English_Name) + " -- $" + ;
                        ALLTRIM(TRANSFORM(TopSellers.TotQuan, ;
                        "99,999,999")) + CHR(13)
ENDSCAN

* Добавьте строку маркера в текстовый фрейм и сделайте маркеры
* числовыми.
```



```
WITH oSlide.Shapes[2].TextFrame.TextRange  
  .Text = BulletString  
  && Следующее доступно только в PowerPoint 2000  
  .ParagraphFormat.Bullet.Style = ppBulletArabicPeriod  
ENDWITH
```

* Запустите слайд-шоу.

```
oPresentation.SlideShowSettings.Run()
```

Теперь вы способны создать слайд-шоу, которое обязательно сведет с ума вашего клиента. Но подождите, это еще не все! В дополнение к этим основным функциям PowerPoint БЕСПЛАТНО предлагает некоторые расширенные функции! В следующей главе объясняются некоторые расширенные функции PowerPoint.

Глава 11. Расширенные функции PowerPoint

Как только вы создадите базовую презентацию, ваши клиенты вскоре начнут просить большего. На самом деле, некоторые из этих расширенных функций (например, функция «Мастер-слайд») могут быть настолько полезными, что вы захотите использовать их прямо сейчас. Другие, например анимацию и переходы, интересно смотреть (и программировать). При разумном использовании они поразят ваших клиентов!

Теперь, когда ваши клиенты впечатлены, вы готовы придать своим презентациям еще больше изящества. Мастер-слайды – это функция, которая гарантирует, что все ваши слайды будут иметь единообразный вид. В этой главе также рассматриваются некоторые необычные функции анимации фигур, управления переходами между слайдами, добавления мультимедиа и создания активных точек, запускающих другие программы. В конце приведены способы добавления примечаний для печати вместе с презентацией.

11.1 Достижение согласованности с мастер-слайдами

Объект SlideMaster задает настройки по умолчанию для макета и внешнего вида каждого слайда. SlideMaster сохраняет настройки по умолчанию для цветов и шрифтов, стандартного размещения элементов, а также объектов, которые должны отображаться на любом слайде (например, логотип компании в углу). На объект SlideMaster ссылается свойство SlideMaster Presentation.

Фон

Ваши клиенты будут настаивать на том, чтобы вы изменили простой белый фон на всех слайдах. Вы можете установить фон индивидуально для каждого слайда, но это будет связано с большим количеством избыточного кода, а это сильно скажется на производительности. Одним из первых свойств SlideMaster, которое стоит изучить, является свойство Background. Это позволяет сделать фон всех слайдов одинаковым с использованием минимума кода и максимальной производительности.

Свойство Background хранит объект ShapeRange. Хотя объект ShapeRange имеет ряд свойств, наиболее полезным с точки зрения свойства Background является свойство Fill, которое ссылается на объект FillFormat, тот же объект, который используется для заливки фигур. Вы можете заполнить фон сплошным, мягким желтым цветом, вот так:

```
#DEFINE rgbSoftYellow RGB(255,255,192)
oPresentation.SlideMaster.Background.Fill.ForeColor.RGB = ;
    rgbSoftYellow
```

Чтобы создать узорчатый фон, установите свойства ForeColor и BackColor объекта FillFormat. Затем используйте метод Patterned, который принимает один параметр – числовое значение, соответствующее шаблону (константы msoPattern). Свойству Pattern присвоено то же значение, которое вы передаете методу Patterned.

```
#DEFINE msoPatternDottedGrid 45
#DEFINE rgbDarkGray RGB(128, 128, 128)
#DEFINE rgbMediumGray RGB(192, 192, 192)
```

* Для узорчатого фона:

```
WITH oPresentation.SlideMaster.Background.Fill
    .ForeColor.RGB = rgbDarkGray
```

```
.BackColor.RGB = rgbMediumGray
.Patterned(msoPatternDottedGrid)
```

```
ENDWITH
```

Этот код создает слайд среднего серого цвета с пунктирной сеткой (примерно через каждые восемь пикселей) темно-серого цвета. Мы не уверены, что это наиболее эстетичный фон, поскольку шаблон слишком мал и перегружен, чтобы его можно было использовать для проецируемых слайдов. На вкладке «Узор» диалогового окна «Эффекты заливки» на рисунке 1 показаны примеры узоров. Использование узоров лучше всего оставить для фигур меньшего размера, которые могут обрабатывать более мелкие узоры.

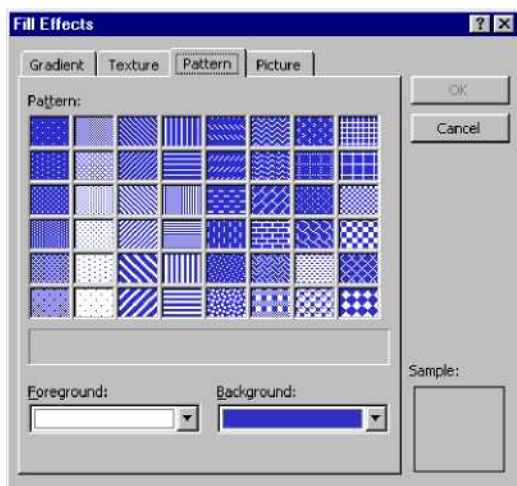


Рисунок 1. Диалоговое окно «Эффекты заливки». В этом диалоговом окне показаны образцы доступных шаблонов.



ForeColor представляет узор, в данном случае пунктирные линии, а BackColor отображается сквозь узор. Некоторые узоры, например пунктирная сетка, отображают больше цвета фона; другие показывают больше цвета переднего плана.

Текстуры

Если вы или ваш клиент очень долго использовали PowerPoint, вы знаете (или вскоре ваш клиент проинформирует вас), что существуют текстурированные фоны. Office предоставляет для использования около 25 предустановленных текстур. Метод `PresetTextured FillFormat` устанавливает фон для указанной текстуры.

```
#DEFINE msoTextureSand 8
oPresentation.SlideMaster.Background.Fill.PresetTextured(msoTextureSand)
```

Эта строка кода устанавливает в качестве фона глубокую текстуру песка. Если вы хотите узнать, какая текстура используется, используйте свойство `PresetTexture`, доступное только для чтения, чтобы вернуть числовое значение.

Что, если вашему клиенту нужна собственная текстура? Не волнуйтесь, PowerPoint предоставляет метод `UserTextured`. Этот метод принимает один параметр, который представляет собой полное имя файла изображения, которое будет располагаться по фону. Это имя файла хранится в свойстве `TextureName` (только для чтения). Свойство `TextureType` указывает, какой тип текстуры используется. Свойство `TextureType` возвращает `msoTexturePreset` (1) или `msoTextureUserDefined` (2).

Заливка изображения

Вы также можете использовать растровое изображение в качестве фона. Метод `UserPicture` принимает параметр, состоящий из полного имени файла растрового изображения. Он заставляет растровое изображение занимать весь фон, а не мозаику (используйте метод `UserTextured` для его мозаики). Он использует некоторые интересные методы сглаживания, когда вы пытаетесь растянуть небольшое растровое изображение на весь слайд. Например, вы можете попробовать:

```
oPresentation.SlideMaster.Background.Fill.UserPicture( ;  
    GETENV("WINDIR") + "\TILES.BMP")
```

Это небольшое растровое изображение, напоминающее кирпич; в увеличенном виде он выглядит как футуристический красно-черный. Это не принесет никаких визуальных наград, но является ярким примером того, как небольшое растровое изображение сглаживается по всему экрану. Не существует документированного свойства, соответствующего методу `UserPicture`, которое бы сообщало вам, какое растровое изображение используется.



Объект `Background` доступен как для `SlideMaster`, так и для фигур. Эти методы и свойства для текстур, заливок изображениями и градиентными заливками доступны и для фигур.

Градиентная заливка

Пожалуй, наиболее востребованным фоном является градиентная заливка — вы знаете такой: приятный средний синий цвет плавно переходит сверху до почти черного цвета внизу. На рисунке 2 показано диалоговое окно PowerPoint, позволяющее выбирать градиентные заливки в интерактивном режиме. Это удобный способ запомнить все различные свойства настройки градиентной заливки.

В PowerPoint доступны три типа цвета градиента. Свойство `GradientColorType` сохраняет текущий выбранный тип градиента. Это свойство доступно только для чтения; для установки градиента используются отдельные методы. Это в ваших интересах, поскольку методы, используемые для установки каждого типа, принимают несколько параметров, гарантируя, что вы установите все необходимые свойства для конкретного градиента. Вот три типа цвета градиента:

- **Предустановленные цвета:** обычно используются три или более цветов. Существует 24 предустановки с такими названиями, как *Daybreak*, *Ocean*, *Fog*, *Moss*, *Wheat*, and *Parchment* (Рассвет, Океан, Туман, Мох, Пшеница и Пергамент) (кстати, это наиболее профессионально выглядящие схемы; остальные могут выглядеть очень ярко в зависимости от используемых цветов текста).
- **Одноцветная заливка:** выбранный цвет переходит в оттенки того же цвета, которые светлее или темнее выбранного цвета (светлые, как белый, или темные, как черный).
- **Двухцветная заливка:** первый цвет переходит во второй цвет.

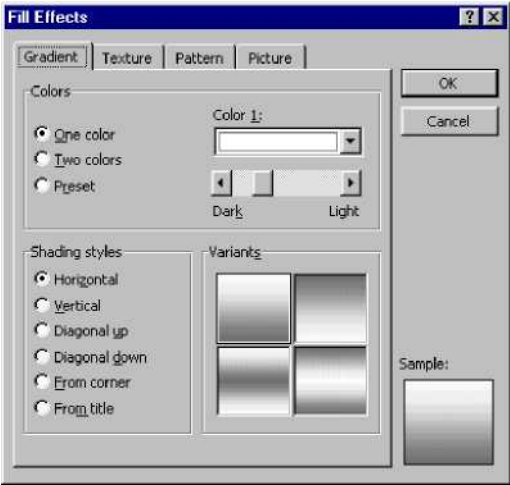


Рисунок 2. Диалоговое окно параметров градиентной заливки PowerPoint. Это удобное визуальное напоминание о свойствах, которые необходимо установить для градиентной заливки.

Предустановленные цвета задаются с помощью метода `PresetGradient`. Этот метод принимает три параметра:

```
oPresentation.SlideMaster.Background.Fill.PresetGradient(nStyle,
nVariant, nType)
```

nStyle	Numeric	Указывает стиль затенения градиента:			
		msoGradientDiagonalDown	4	msoGradientDiagonalUp	3
		msoGradientFromCenter	7	msoGradientFromCorner	5
		msoGradientFromTitle	6	msoGradientHorizontal	1
		msoGradientVertical	2		
nVariant	Numeric	Указывает выбор порядка цветов (без констант):			
		1 = от цвета 1 к цвету 2 (верхнее левое поле в диалоговом окне градиента)			
		2 = от цвета 2 к цвету 1 (верхнее правое поле в диалоговом окне градиента)			
		3 = от цвета 1 к цвету 2 и обратно к цвету 1 (нижнее левое поле в диалоговом окне градиента)			
		4 = от цвета 2 к цвету 1 и обратно к цвету 2 (нижнее правое поле в диалоговом окне градиента)			
nType	Numeric	Указывает предустановленную цветовую схему. Некоторые из 24:			
		msoGradientDaybreak	4	msoGradientOcean	7
		msoGradientFog	10	msoGradientParchment	14
		msoGradientMoss	11	msoGradientWheat	13

Большинство клиентов могут возражать против предустановленных цветов. Как уже говорилось ранее, многие из них яркие, а профессиональные, вероятно, чрезмерно используются конкурентами. Ваш клиент может быть более изысканным, чем предустановленные цвета (кстати, это отличный аргумент, чтобы отговорить

клиента, который выбирает один из ярких пресетов, взгляните на тип Rainbow [16], чтобы понять, что мы имеем в виду).

Следующий тип градиента – одноцветный градиент, который использует метод `OneColorGradient` для установки соответствующих свойств. Как и метод `PresetGradient`, он также принимает три параметра; первые два идентичны методу `PresetGradient`:

```
oPresentation.SlideMaster.Background.Fill.OneColorGradient(nStyle,
    nVariant, nDegree)
```

nStyle	Numeric	Стиль затенения градиента. См. <code>PresetGradient</code> для констант.
nVariant	Numeric	Выбор цветового порядка. Значения см. в <code>PresetGradient</code> .
nDegree	Numeric	Значение от 0 до 1, обозначающее темноту результирующего цвета. 0 – черный, 1 – белый; .25 – темный оттенок цвета, а .75 – бледный оттенок цвета. Обозначает цвет 2.

Обратите внимание, что нет упоминания о том, какой цвет использовать в качестве цвета 1. Свойство `ForeColor` используется для цвета 1. Чтобы установить цвет фона на слайде от среднего королевского синего до бледного оттенка того же синего, используйте следующий код:

```
#DEFINE msoGradientHorizontal 1
#DEFINE rgbMediumBlue          RGB(0, 0, 150)

WITH oPresentation.SlideMaster.Background.Fill
    .ForeColor.RGB = rgbMediumBlue
    .OneColorGradient(msoGradientHorizontal, 1, .75)
ENDWITH
```

Последний тип градиента – двухцветный градиент. Да, как вы уже догадались: используйте метод `TwoColorGradient`. Он принимает только два параметра:

```
oPresentation.SlideMaster.Background.Fill.TwoColorGradient(nStyle,
    nVariant)
```

nStyle	Numeric	Стиль затенения градиента. См. <code>PresetGradient</code> для констант.
nVariant	Numeric	Выбор порядка цветов. Значения см. в <code>PresetGradient</code> .

Эти два цвета задаются цветами переднего плана и фона. В следующем примере цвет меняется от королевского синего к бледно-бирюзовому:

```
#DEFINE msoGradientHorizontal 1
#DEFINE rgbMediumBlue          RGB( 0, 0, 150)
#DEFINE rgbPaleTeal            RGB(192, 255, 255)

WITH oPresentation.SlideMaster.Background.Fill
    .ForeColor.RGB = rgbMediumBlue
    .BackColor.RGB = rgbPaleTeal
    .TwoColorGradient(msoGradientHorizontal, 1)
ENDWITH
```

Поскольку вы устанавливаете ряд свойств, вызывая методы, какие именно свойства вы устанавливаете? В таблице 1 показаны свойства FillFormat, которые задаются с помощью каждого метода. Все эти свойства доступны только для чтения, и их можно запросить, чтобы определить текущие настройки. Обязательно сначала проверьте свойство GradientColorType, а затем запросите только те свойства, которые применимы к типу градиента. Неиспользуемые свойства (например, GradientDegree, если это заданный или двухцветный тип градиента) не сбрасываются до значений по умолчанию при изменении типа градиента. Например, определение того, что GradientDegree равно 0,75, не гарантирует, что используется одноцветный градиент, для уверенности необходимо запросить GradientColorType.

Таблица 1. Свойства градиента объекта FillFormat. Каждый тип градиента использует большинство свойств градиента, но не все. Проверьте GradientColorType, чтобы убедиться, что вы установили соответствующие свойства для градиентной заливки.

Свойство	PresetGradient	OneColorGradient	TwoColorGradient
GradientColorType	msoGradientPresetColors (3)	msoGradientOneColor (1)	msoGradientTwoColors (2)
GradientStyle	параметр nStyle	параметр nStyle	параметр nStyle
GradientVariant	параметр nVariant	параметр nVariant	параметр nVariant
GradientDegree	NA	0 (черный) - 1 (белый)	NA
PresetGradientType	параметр nType	NA	NA

Все типы фона (сплошной, узорчатый, графический, текстурированный и градиентный) задают свойства с помощью методов FillFormat. В таблице 2 показаны все свойства, которые устанавливаются или используются для каждого метода. Помните, что установка другого фона не приводит к сбросу неиспользуемых свойств к значениям по умолчанию. Запросите свойство Type FillFormat, чтобы убедиться, какой формат фона используется.

Таблица 2. Свойства FillFormat, заданные различными методами FillFormat. Помните, что свойство Type является единственным индикатором используемого формата фона. Не полагайтесь на значения других свойств исключительно для определения типа формата фона.

Property	Solid	Textured Patterned	Preset Texture	User Defined Texture	Gradient UserPicture	Preset Gradient	OneColor Gradient	TwoColor Gradient
Read Only								
Type	msoFillSolid (1)	msoFillPatterned (2)	msoFillTextured (4)	msoFillTextured (4)	msoFillPicture (6)	msoFillGradient (3)	msoFillGradient (3)	msoFillGradient (3)
Pattern	-	msoPattern constants	-	-	-	-	-	-
TextureType	-	-	msoTexturePreset (1)	msoTextureUserDefined (3)	-	-	-	-
PresetTexture	-	-	msoTexture constants	-	-	-	-	-
TextureName	-	-	-	BMP filename	-	-	-	-
GradientColorType	-	-	-	-	-	msoGradientPresetColors (3)	msoGradientOneColor (1)	msoGradientTwoColors (2)

Property	Solid	Textured Patterned	Preset Texture	User Defined Texture	Gradient UserPictu re	Preset Gradient	OneColor Gradient	TwoColor Gradient
GradientStyle	-	-	-	-	-	nStyle	nStyle	nStyle
GradientVariant	-	-	-	-	-	nVariant	nVariant	nVariant
PresetGradientType	-	-	-	-	-	nType	-	-
GradientDegree	-	-	-	-	-	-	0.0-1.0	-
Read/Write								
ForeColor	✓	✓	-	-	-	-	✓	✓
BackColor	-	✓	-	-	-	-	-	✓

Стандартизация внешнего вида текста

Текст управляется коллекцией `TextStyles` объекта `SlideMaster`. В коллекции три объекта. Первый – это стиль по умолчанию, используемый при добавлении фигуры с текстом. Второй – это стиль заголовка, используемый для всех объектов-заполнителей заголовка. Последний объект – это стиль тела, используемый для всех остальных заполнителей в `AutoLayout`.

Внутри каждого объекта `TextStyle` находится коллекция объектов `Level` (в справке они называются объектами `TextStyleLevel`, хотя доступ к ним осуществляется через свойство `Level`). Текстовые уровни легче всего объяснить с помощью маркеров: обычно каждый уровень имеет отступ от предыдущего, и каждый уровень имеет разные символы маркеров и характеристики шрифта, хотя каждый уровень не обязательно должен иметь отступ или маркировку. На рисунке 3 показан мастер-слайд в PowerPoint, где показаны характеристики по умолчанию для каждого уровня. PowerPoint поддерживает пять уровней.

Каждый объект `Level` содержит объект `Font` и объект `ParagraphFormat`. Объект `ParagraphFormat` содержит объект `Bullet`. Эти объекты описаны в главе 10 «Основы PowerPoint» и имеют те же свойства и методы при использовании для уровней, что и для `TextRanges`.

Вы можете использовать коллекции `TextStyles` и `Levels` для изменения цвета и шрифта объектов `Title`. Обычно фон слайд-шоу имеет более темный цвет, например, от среднего до темно-синего, чтобы обеспечить читаемость. Когда фон темный, текст должен быть светлым, возможно, с оттенком желтого. Кроме того, для заголовка обычно используется привлекательный шрифт без засечек. Следующий код демонстрирует это. На рисунке 4 показаны результаты.

```
#DEFINE pptTitleStyle 2
#DEFINE rgbYellow      RGB(255,255,0)

WITH oPresentation.SlideMaster.TextStyles[pptTitleStyle].Levels[1].Font
    .Name = "Arial"
    .Color.RGB = rgbYellow
ENDWITH
```



Рисунок 3. SlideMaster по умолчанию, как показано в PowerPoint. Обратите внимание на уровни, показанные в области объекта.



Рисунок 4. SlideMaster в PowerPoint. Здесь показаны результаты примера кода, который устанавливает текст по умолчанию для заголовков и заполнителей.

Как показано в примере, Title использует коллекцию Levels. Заголовок может иметь несколько уровней, например, заголовок и подзаголовок находятся в одном прямоугольнике. Хотя уровни полезны для изменения шрифтов в заголовках, они действительно превосходны в заполнителях, особенно когда используются маркеры.

В следующем примере показано, как отформатировать первые три уровня в заполнителях. Константа ppBodyStyle используется для ссылки на третий объект TextStyle. Первый уровень имеет темно-синий цвет. Символ маркера – это точка, но мы ее отключили. Символ настроен так, что его можно включить при необходимости. Второй уровень выставляется аналогично, но маркер остается включенным. Третий уровень немного меньше, средне-голубой, и в нем используется тире. Результат показан на рисунке 4.

```
#DEFINE ppBodyStyle 3
#DEFINE rgbDarkBlue RGB(0, 0, 128)
#DEFINE rgbMediumCyan RGB(0, 192, 192)
```

* Установить первый уровень.

```
WITH oPresentation.SlideMaster.TextStyles[ppBodyStyle].Levels[1]
    WITH .Font
```

```

        .Name      = "Arial"
        .Color.RGB = rgbDarkBlue
        .Size      = 20
    ENDWITH
    WITH .ParagraphFormat.Bullet
        * Установите в качестве символа маркера точку
        * (используйте 149, если используете PowerPoint 97).
        .Character = 8226
        .Visible   = .F.    && Не показывать маркер!
    ENDWITH
ENDWITH

* Установите второй уровень.
WITH oPresentation.SlideMaster.TextStyles[ppBodyStyle].Levels[2]
    WITH .Font
        .Name      = "Arial"
        .Color.RGB = rgbDarkBlue
        .Size      = 20
    ENDWITH
    * Установите в качестве символа маркера точку
    * (используйте 149, если используете PowerPoint 97).
    .ParagraphFormat.Bullet.Character = 8226

ENDWITH

* Установите третий уровень.
WITH oPresentation.SlideMaster.TextStyles[ppBodyStyle].Levels[3]
    WITH .Font
        .Name      = "Arial"
        .Color.RGB = rgbMediumCyan
        .Size      = 18
    ENDWITH
    * Установите в качестве маркера тире
    * (используйте 150, если используете PowerPoint 97).
    .ParagraphFormat.Bullet.Character = 8211
ENDWITH

```

Аналогичный код можно использовать для установки текста по умолчанию для добавленных фигур, используя константу ppDefaultStyle для доступа к первому объекту TextStyle.

Стандартизация цветов с помощью ColorSchemes

Альтернативой настройке цветов в SlideMaster является использование объекта ColorScheme, в котором хранятся цвета восьми стандартных элементов слайда. На рисунке 5 показано диалоговое окно PowerPoint, позволяющее настраивать цвета. Чтобы отобразить его в PowerPoint, выберите в меню «Формат | Цветовая схема слайда», затем перейдите на вкладку «Пользовательский». Может существовать до 16 объектов ColorScheme; они сохраняются в коллекции ColorSchemes.

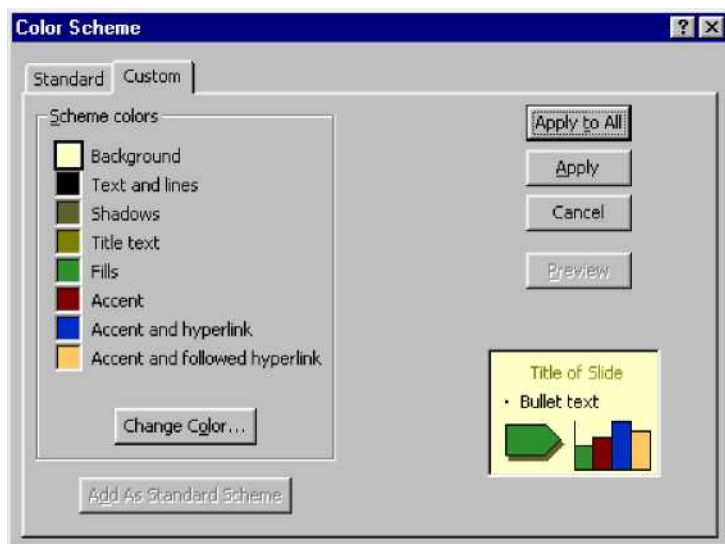


Рисунок 5. Диалоговое окно «Цветовая схема PowerPoint». Цветовые схемы устанавливают восемь стандартных цветов для объектов.

Хотя коллекция ColorSchemes принадлежит объекту Presentation, объект SlideMaster или любой слайд может ссылаться на любой объект ColorScheme. В файле справки довольно подробно описано, как использовать несколько цветовых схем в презентации (например, иметь одну цветовую схему для титульных слайдов, а другую – для стандартных слайдов).

Самое простое и практичное использование одной цветовой схемы – это задание цветов восьми элементов в SlideMaster. ColorScheme содержит коллекцию из восьми цветов, на каждый из которых ссылается константа. Константы для восьми цветов: ppBackground (1), ppForeground (2), ppShadow (3), ppTitle (4), ppFill (5), ppAccent1 (6), ppAccent2 (7) и ppAccent3 (8). Внимание: установка цветов фона и заливки в ColorScheme переопределяет любые установленные свойства объектов Background и Fill – если вы настраиваете фон с градиентной заливкой, не задавайте цвет фона цветовой схемы!

Следующий код меняет цвет заголовка на красный, а текста (ppForeground) на темно-синий. Обязательно используйте этот код вместо настройки цветов в SlideMaster. Цвета, заданные явно через TextStyles SlideMaster, будут иметь приоритет над ColorSchemes (хотя, если вы посмотрите на диалоговое окно, цвет ColorSchemes изменится, но не изменит цвета на слайдах).

```
#DEFINE ppTitle      4
#DEFINE ppForeground 2
#DEFINE rgbRed       RGB(255, 0, 0)
#DEFINE rgbDarkBlue  RGB( 0, 0, 128)

WITH oPresentation.SlideMaster.ColorScheme
    .Colors[ppTitle].RGB = rgbRed
    .Colors[ppForeground].RGB = rgbDarkBlue
ENDWITH
```

Подробнее о формах SlideMaster

Пять форм в SlideMaster именно такие: Shape объекты. Вы можете установить любые свойства формы, а не только шрифты. Изменение размера или границ форм SlideMaster влияет на заполнители всех форм. Чтобы получить доступ к свойствам форм, используйте коллекцию Shapes. Индексы имеют значения от 1 до 5 в следующем порядке: область заголовка, область объекта, область даты, область

нижнего колонтитула и область номеров. Учитывая множество констант, доступных в VBA, удивительно, что для этих объектов нет констант. (Если вы планируете часто с ними работать, вы, безусловно, можете определить свои собственные.)

Все формы получают цвет из объекта ColorScheme в SlideMaster. В примере из предыдущего раздела весь текст темно-синего цвета. Возможно, вы захотите, чтобы область нижнего колонтитула была темно-зеленой. Переопределите настройку ColorScheme, обратившись напрямую к форме. Следующие строки кода устанавливают текст нижнего колонтитула на «Tasmanian Traders» и устанавливают темно-зеленый цвет.

```
#DEFINE rgbDarkGreen RGB(0, 128, 0)

WITH oPresentation.SlideMaster.Shapes[4].TextFrame.TextRange
    .Font.Color.RGB = rgbDarkGreen
    .Text = "Tasmanian Traders"
ENDWITH
```

Первые две формы, область заголовка и область объекта, получают свойства шрифта из объектов TextStyles. На остальные три (область даты, область нижнего колонтитула и область чисел) TextStyles не влияют. Установите свойства шрифта этих трех объектов отдельно.

11.2 Необычные функции

Предупреждение: следующие функции интересно программировать и интересно наблюдать. Однако принцип «лучше меньше, да лучше» здесь очень уместен. В простоте есть элегантность. При умеренном использовании эти функции могут выделить вашу презентацию среди других. Злоупотребляйте ими, и аудитория устает от всех уловок, что приводит к тому, что они игнорируют слайды и, вполне возможно, докладчика, который их представляет. Помните, аудитория видит только презентацию, а не крутые трюки в вашем коде.

Анимации

Анимация — это специальные визуальные и звуковые эффекты, которые выделяют важные элементы на слайде. Некоторые анимационные эффекты, поддерживаемые PowerPoint, — это объекты, летящие от края (выберите край), текст, отображающийся по буквам, и воспроизведение звуковых файлов. Эти анимации могут быть синхронизированы, чтобы они происходили последовательно или все сразу. Они могут происходить автоматически или в ответ на щелчки мыши. Существует так много комбинаций и вариантов анимации; мы не будем пытаться охватить каждый из них. Диалоговое окно «Анимация» предоставляет вам отличный интерфейс для их изучения. Мы дадим вам представление о том, с чего можно начать.

В PowerPoint 2000 поддерживается 19 эффектов анимации, многие из которых имеют несколько вариантов, в результате чего общее количество эффектов достигает примерно 80. На рисунке 6 показано диалоговое окно «Пользовательская анимация» PowerPoint с открытым раскрывающимся списком «Эффекты». Это диалоговое окно также дает некоторое представление о доступных коллекциях, объектах и свойствах. В левом верхнем углу находится список всех объектов на слайде; каждый может быть анимирован отдельно (или не анимирован вообще). Откроется вкладка «Эффекты», на которой отображено множество свойств, участвующих в настройке эффекта. На других вкладках указано, что имеются свойства для установки порядка и времени эффектов, анимации диаграмм (если таковые имеются) и добавления мультимедийных эффектов.

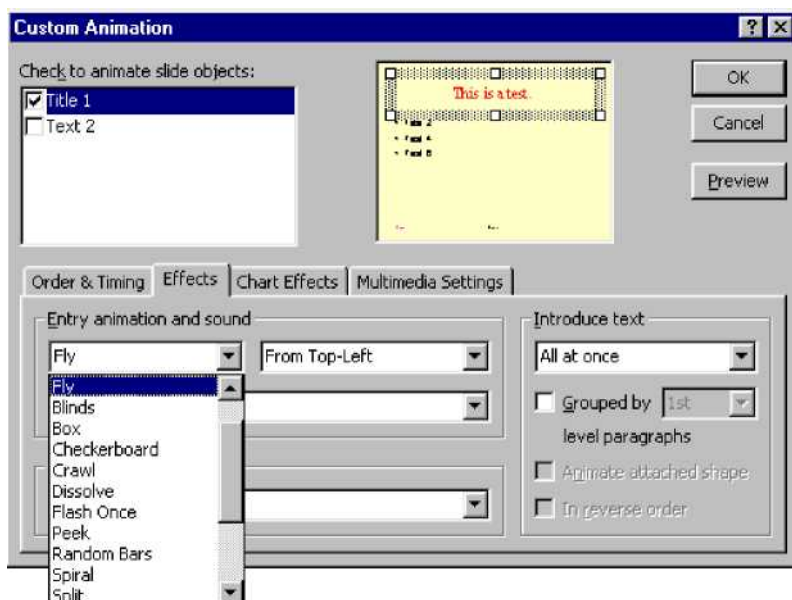


Рисунок 6. Диалоговое окно «Настраиваемая анимация» PowerPoint.

Раскрывающийся список «Эффекты» открыт, в нем показаны многие доступные эффекты анимации.

Объект AnimationSettings хранит все объекты, коллекции и свойства, относящиеся к анимации. Объект AnimationSettings хранится в свойстве AnimationSettings каждого объекта Shape. Формы в SlideMaster также можно анимировать.

Настройка эффекта

Свойство EntryEffect задает эффект. Этому свойству присвоено числовое значение, для которого существуют константы VBA. Константы созданы для объединения двух полей в диалоговом окне, описывающих то, что должно произойти (показано на рисунке 6 как «Вылет») и любых доступных опций для этого эффекта, например, откуда он может переместиться (показано на рисунке 6 как «Сверху-слева»). Вот некоторые константы эффектов: ppEffectAppear (3844), ppEffectFlashOnceFast (3841), ppEffectFlashOnceMedium (3842), ppEffectFlyFromLeft (3329), ppEffectFlyFromTopLeft (3333), ppEffectRandom (513), ppEffectZoomIn (3345) и ppEffectZoomInSlightly (3346). Используйте Обзорщик объектов, чтобы найти остальные 72 константы. Подробную информацию об обзорщике объектов см. в разделе «Просмотр» в главе 2.

Следующий код устанавливает первую фигуру на слайде, которая будет вылетать слева:

```
#DEFINE ppEffectFlyFromLeft 3329
oSlide.Shapes(1).AnimationSettings.EntryEffect = ;
    ppEffectFlyFromLeft
```

Изменение эффектов

Очень популярный эффект – слайд, содержащий маркированный текст, и каждый из маркеров вылетает слева, когда пользователь щелкает мышью. Модификация эффекта «вылета слева» заключается в том, чтобы указать, на каких уровнях маркеры вылетают вместе. TextLevelEffect сообщает, какие уровни анимируются. Для TextLevelEffect задано числовое значение. Некоторые константы – это ppAnimateByFirstLevel (1), ppAnimateBySecondLevel (2), ppAnimateByAllLevels (16) и ppAnimateLevelNone (0). Установка эффекта для анимации первого уровня отправляет каждый маркер первого уровня отдельно со всеми его подчиненными.

Установка эффекта для анимации второго уровня отправляет каждый элемент первого и второго уровня отдельно, все подчиненные элементы второго уровня передаются вместе со своим родителем.

Следующий код создает пример маркетингового слайда Tasmanian Traders, показывающий маркированный список с двумя уровнями. Код устанавливает, что заголовок и маркеры будут появляться слева, по одному при каждом щелчке мыши. Код настроен на появление каждого маркера по отдельности (на уровне 2). Измените код для второго запуска, изменив TextLevelEffect на ppAnimateByFirstLevel, чтобы увидеть, что первый маркер появится вместе с двумя подчиненными маркерами. На рисунке 7 показан получившийся слайд.

```
#DEFINE ppEffectFlyFromLeft      3329
#DEFINE ppLayoutText              2
#DEFINE ppAnimateByFirstLevel     1
#DEFINE ppAnimateBySecondLevel    2
```

* Добавьте заголовок слайда и текстовые объекты в макет.

```
oSlide = oPresentation.Slides.Add(1, ppLayoutText)
```

* Вставьте текст заголовка и установите эффект.

```
WITH oSlide.Shapes[1]
    .TextFrame.TextRange.Text = "Тасманские трейдеры"
    .AnimationSettings.EntryEffect = ppEffectFlyFromLeft
ENDWITH
```

* Вставьте несколько маркеров (на 2 уровнях) и установите эффект.

```
m.BulletString = "Дистрибьютор изысканных продуктов питания" + ;
    CHR(13) + ;
    "Напитки" + CHR(13) + ;
    "Десерты" + CHR(13) + ;
    "Международная доставка" + CHR(13) + ;
    "Исполнение обязательств гарантировано"
WITH oSlide.Shapes[2]
    .TextFrame.TextRange.Text = m.BulletString
```

* Отступы для "Напитки" и "Десерты"

```
.TextFrame.TextRange.Sentences[2, 2].IndentLevel = 2
```

```
.AnimationSettings.EntryEffect      = ppEffectFlyFromLeft
.AnimationSettings.TextLevelEffect = ppAnimateBySecondLevel
```

* Запустите это еще раз, но измените константу TextLevelEffect
 * в строке выше на ppAnimateByFirstLevel, чтобы увидеть, как
 * установка ее на первый уровень приведет к появлению двух
 * маркеров «уровня 2» с их родительским элементом «уровня 1».

```
ENDWITH
```

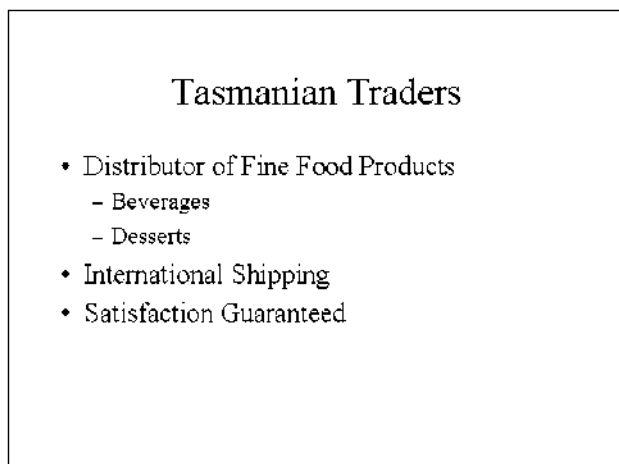



Рисунок 7. Пример маркированного списка. Уровни используются для форматирования маркеров и текста. Уровень 1 имеет круглый маркер. На уровне 2 имеется тире вместо маркера и текст немного меньшего размера.

Автоматическое определение времени

Чтобы исключить необходимость ручного щелчка мышью и позволить показу запускаться автоматически, используйте свойство `AdvanceMode`. Есть две константы: `ppAdvanceOnClick` (1) и `ppAdvanceOnTime` (2). Если для свойства `AdvanceMode` установлено значение `ppAdvanceOnTime`, оно проверяет количество секунд, сохраненное в свойстве `AdvanceTime`. Добавьте следующие строки в конец предыдущего примера, и он автоматически анимирует элементы с интервалом в одну секунду.

```
#DEFINE ppAdvanceOnTime 2
```

```
* Измените настройки формы заголовка
```

```
WITH oSlide.Shapes[1].AnimationSettings  
    .AdvanceMode = ppAdvanceOnTime  
    .AdvanceTime = 1  
ENDWITH
```

```
* Измените настройки формы маркированного текста
```

```
WITH oSlide.Shapes[2].AnimationSettings  
    .AdvanceMode = ppAdvanceOnTime  
    .AdvanceTime = 1  
ENDWITH
```

Если форм несколько, порядок анимации форм задается с помощью свойства `AnimationOrder`. По умолчанию `AnimationOrder` устанавливает порядок создания формы (конечно, при условии, что форма вообще должна быть анимирована). Установив свойство `AnimationOrder`, вы можете изменить порядок анимации объектов.

Это лишь некоторые из множества свойств и методов, доступных для анимации форм. Полный список см. в файле справки в разделе «Объект `AnimationSettings`».

Переходы

Тускнеет, растворяется, вытирается; все это различные способы перехода между слайдами. Переходы можно применять к `SlideMaster`, который влияет на все слайды, или только к одному слайду. Переходы хранятся в объекте `SlideShowTransition`. Объект `SlideShowTransition` аналогичен объекту `AnimationSettings`. Переход на

другой слайд аналогичен переходу одного объекта на экран.

Свойство `EntryEffect` используется для хранения эффекта перехода. Оно использует тот же набор констант, что и свойство `AnimationSettings EntryEffect`. Однако не все константы для `EntryEffects` доступны для переходов (например, `ppFlyInFromLeft` предназначен только для анимации).

Как и `AnimationSettings`, объект `SlideShowTransition` имеет свойства для автоматического перемещения. Однако синтаксис отличается от синтаксиса `AnimationSettings`. Объект `SlideShowTransition` имеет логическое свойство `AdvanceOnTime`. Свойство `AdvanceTime` хранит количество секунд (так же, как и в объекте `AnimationSettings`).

Следующий код добавляет переход к первому слайду, который «закрывается» и автоматически перемещается через две секунды. Если бы этот код был применен к `SlideMaster`, он повлиял бы на каждый слайд, но, как написано здесь, он влияет только на первый слайд.

```
#DEFINE ppEffectCoverDown 1284

WITH oPresentation.Slides[1].SlideShowTransition
    .EntryEffect = ppEffectCoverDown
    .AdvanceOnTime = .T.
    .AdvanceTime = 2
ENDWITH
```

Выполнение действий

PowerPoint позволяет выполнять определенные действия при наведении указателя мыши на фигуру или при щелчке по ней. Действия включают в себя переход к определенному слайду в слайд-шоу, запуск другого слайд-шоу, запуск отдельной программы, воспроизведение звука, а также некоторые другие. На рисунке 8 показано интерактивное диалоговое окно «Параметры действий» PowerPoint, в котором перечислены возможные действия.

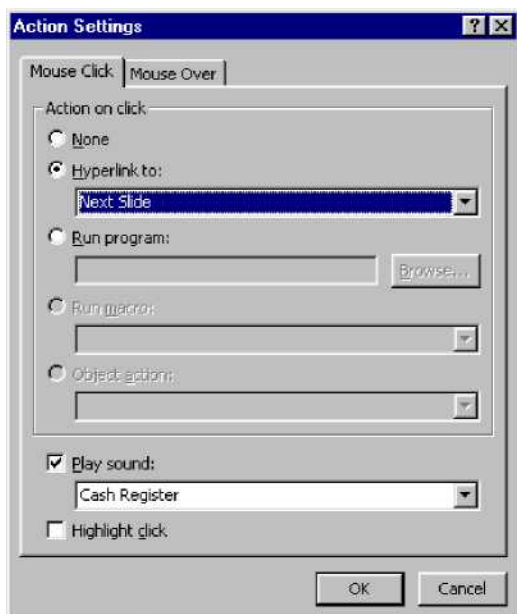


Рисунок 8. Диалоговое окно настроек действий PowerPoint. Настройки действий определяют действие, выполняемое при наведении курсора мыши и щелчках мыши.

Коллекция `ActionSettings` хранит эти действия. Коллекция содержит два объекта `ActionSettings`: один для щелчка мыши, другой для наведения указателя мыши.

Индексами коллекций являются ppMouseClicked (1) и ppMouseOver (2).

Свойство Action – это свойство, управляющее действием. Задайте для свойства Action одну из констант, перечисленных в таблице 3.

Таблица 3. Где происходит действие. Задайте свойство Action, чтобы определить действие, выполняемое, когда пользователь щелкает форму.

Константа	Значение	Константа	Значение
ppActionNone	0	ppActionHyperlink	7
ppActionNextSlide	1	ppActionRunMacro	8
ppActionPreviousSlide	2	ppActionRunProgram	9
ppActionFirstSlide	3	ppActionNamedSlideShow	10
ppActionLastSlide	4	ppActionOLEVerb	11
ppActionLastSlideViewed	5	ppActionPlay	12
ppActionEndShow	6		

Следующий код добавляет фигуру стрелки в правом нижнем углу слайда и устанавливает свойство Action для перехода к предыдущему слайду.

```
#DEFINE msoShapeRightArrow 33
#DEFINE ppMouseClicked 1
#DEFINE ppActionPreviousSlide 2
```

```
oShape = oSlide.Shapes.AddShape(msoShapeRightArrow, 600, 450, 50, 50)
oShape.ActionSettings[ppMouseClicked].Action = ppActionPreviousSlide
```

В PowerPoint есть функция под названием «Кнопки действий», которая представляет собой серию predetermined кнопок. Эти кнопки имеют единообразный вид и помогают дизайнеру презентаций поддерживать единообразный внешний вид. При интерактивном вводе в PowerPoint в диалоговом окне отображаются значения действий по умолчанию для выбранного типа кнопки. Например, размещение кнопки с изображением «Конец» устанавливает действие по умолчанию для перехода к концу слайд-шоу. Когда эти кнопки добавляются в код (макросы Автоматизации или VBA), действие по умолчанию не указывается, его необходимо указать явно, используя код, подобный показанному ранее.

Коллекция ActionSettings имеет дополнительные свойства для поддержки других типов действий. Чтобы ввести гиперссылку, сначала установите для свойства Action значение ppActionHyperlink, а затем задайте свойства объекта Hyperlink. Чтобы создать горячую ссылку, установите свойство Address объекта Hyperlink, в котором хранится URL-адрес. Вот пример кода, который добавляет фигуру и прикрепляет гиперссылку к URL-адресу.

```
#DEFINE msoShapeRectangle 1
#DEFINE ppMouseClicked 1
#DEFINE ppActionHyperlink 7
```

```
oShape = oSlide.Shapes.AddShape(msoShapeRectangle, 300, 200, 150, 100)
WITH oShape.ActionSettings[ppMouseClicked]
    .Action = ppActionHyperlink
    .Hyperlink.Address = "http://www.hentzenwerke.com"
```

ENDWITH

Вы даже можете запустить любую программу, включая скомпилированные программы FoxPro, одним щелчком мыши или наведением курсора мыши. Это достигается с помощью метода Run. Передайте ему полное имя программы.

```
#DEFINE msoShapeRectangle 1  
#DEFINE ppMouseClicked 1
```

```
oShape = oSlide.Shapes.AddShape(msoShapeRectangle, 300, 200, 150, 100)  
oShape.ActionSettings[ppMouseClicked].Run = ;  
    "C:\Program Files (x86)\Microsoft Office\Office11\WINWORD.EXE"
```

11.3 Мультимедиа.

В этом мире высокотехнологичных фильмов и видеоигр мультимедиа в бизнес-презентациях почти ожидаема. Мультимедиа способна улучшить презентацию, делая ее более интересной для просмотра и облегчая зрителю запоминание представленного. Однако существует тонкая грань между «очень интересным» и «очень отвлекающим» – опять же, мы прислушиваемся к поговорке «лучше меньше, да лучше».

Разрабатывая автоматизированные презентации, убедитесь, что ваша программа может масштабироваться для широкого спектра оборудования, доступного для презентаций. Даже сегодня не все компьютеры имеют хорошие звуковые системы (или динамики, которые достаточно приличны, чтобы проецировать звук на всю комнату, полную посетителей), мощные видеокарты, а также большой объем оперативной памяти и процессора, позволяющие провести презентацию успешно. Убедитесь, что у ваших пользователей есть альтернатива потрясающим мультимедийным дисплеям. Если презентационный компьютер не может обрабатывать мультимедиа, «невероятно потрясающая» презентация быстро воспринимается как «невероятно ужасная».

Отключение звука

Управление звуками осуществляется через объект SoundEffect. Каждый из объектов ActionSettings, AnimationSettings и SlideShowTransition (обсуждавшихся ранее в этой главе) имеет свойство SoundEffect для доступа к объекту SoundEffect.

В PowerPoint доступен ряд звуковых эффектов без использования отдельных звуковых файлов. Доступ к этим встроенным звукам осуществляется по именам, указанным в любом раскрывающемся списке звуковых эффектов PowerPoint. На рисунке 9 показан раскрывающийся список «Воспроизвести звук» в диалоговом окне «Настройки действий».

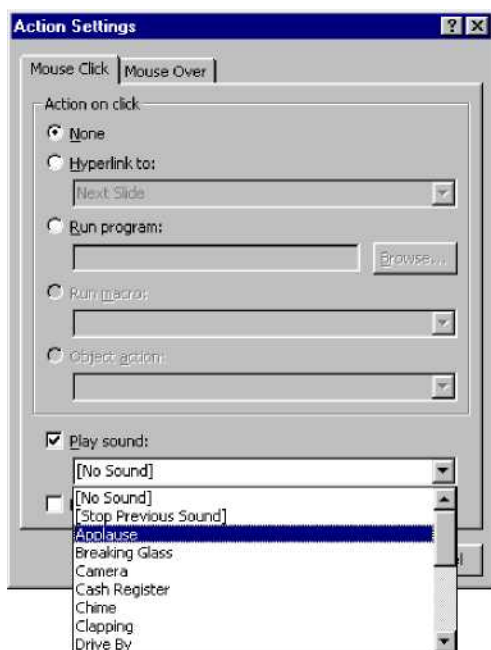


Рисунок 9. Диалоговое окно «Настройки действий» PowerPoint, показывающее встроенные звуковые эффекты.

Задайте для свойства Name текстовую строку, соответствующую звуку, который вы хотите использовать. Это исключение из длинного списка числовых констант, обычно используемых в VBA! Список опций: Аплодисменты, Барабан, Бластер, Ветер, Взрыв, Граната, Задвигание, Засасывание, Камера, Касса, Колокольчики, Молоток, Монета, Напряжение, Пишущая машинка, Стрела, Хлыст, Шум, Щелчок (Applause, Breaking Glass, Camera, Cash Register, Chime, Clapping, Drive By, Drum Roll, Explosion, Gun Shot, Laser, Ricochet, Screeching Brakes, Slide Projector, Typewriter, Whoosh)

```
#DEFINE ppMouseClicked 1
oShape.ActionSettings[ppMouseClicked].SoundEffect.Name = "Касса"
```

Установка свойства Name автоматически устанавливает для свойства Type значение ppSoundFile (2). Чтобы отключить звук, установите для свойства Type значение ppSoundNone (0). Константа ppStopPrevious (1) останавливает любой воспроизводимый в данный момент звук.

Чтобы указать файл WAV для воспроизведения, используйте метод ImportFromFile, чтобы связать его с файлом WAV и настроить объект для его воспроизведения. Передайте полное имя файла WAV. После того как вы импортировали файл WAV, он добавляется в список доступных звуков, поэтому на него могут ссылаться другие объекты (по имени файла без пути), как и на встроенные звуки. Следующие строки кода импортируют WAV-файл, задают форму для его воспроизведения при щелчке, а затем используют импортированный файл по его имени, чтобы установить тот же звук для воспроизведения при переходе слайд-шоу.

```
#DEFINE ppMouseClicked 1

SoundFile = GETENV("WINDIR") + "\Media\Chord.WAV"
oShape.ActionSettings[ppMouseClicked].SoundEffect.ImportFromFile(SoundFile)
oSlide.SlideShowTransition.SoundEffect.Name = "CHORD.WAV"
```

Объект SoundEffect имеет метод Play, который воспроизводит звук по требованию. Используйте метод Play только в том случае, если вы хотите, чтобы звук воспроизводился во время работы вашего кода; использование метода Play не

влияет на воспроизведение звуков в слайд-шоу. Он воспроизводит набор звуков для указанного объекта:

```
oSlide.SlideShowTransition.SoundEffect.Play()
```

Эта строка воспроизводит любой звук, установленный в объекте SlideShowTransition слайда.

Движение

Чтобы добавить видеоклипы, вы используете метод AddMediaObject коллекции Shapes. Как указано в главе 10 «Основы PowerPoint», хотя автомакеты существуют с заполнителями для объектов OLE, их нельзя использовать с Автоматизацией или макросами.

Метод AddMediaObject принимает до пяти параметров. Первое – это имя файла, оно не является обязательным. Далее идут Left и Top. Это необязательно; значение по умолчанию равно нулю для обоих. Последние два, Width и Height, также являются необязательными и по умолчанию соответствуют ширине и высоте объекта. Следующая строка кода показывает, как добавить один из примеров файлов AVI FoxPro (это вращающийся глобус, а не лиса, как следует из названия).

```
oShape = oSlide.Shapes.AddMediaObject( ;  
    _SAMPLES + "Solution\Forms\Fox.avi", 240, 156)
```

Это добавит файл AVI примерно по центру слайда.



В поиске параметров Left и Top не было никакого волшебства. Это было сделано в интерактивном режиме. Используя командное окно FoxPro, мы открыли экземпляр PowerPoint и добавили новую презентацию с пустым слайдом. Затем мы выполнили предыдущую команду, оставив параметры Left и Top, которые поместили изображение в верхний левый угол. Активировав PowerPoint, мы переместили изображение с помощью мыши в нужное место. Вернувшись в командное окно FoxPro, мы запросили у PowerPoint свойства Left и Top и ввели результаты в наш код:

```
? oShape.Left  
? oShape.Top
```

Важно помнить, что инструменты должны делать всю работу за вас. Ссылки на объекты FoxPro сохраняются до тех пор, пока переменная не будет освобождена или объект в PowerPoint не станет недоступен (независимо от того, закрыта ли презентация или удален сам объект). Установка ссылки на объект в FoxPro, переключение в PowerPoint для интерактивного управления объектами PowerPoint, а затем запрос свойств из FoxPro – это очень мощный способ быстрого определения желаемых свойств объектов PowerPoint.

Добавление медиаклипа с помощью AddMediaObject позволяет настроить его воспроизведение, когда пользователь нажимает на него во время презентации. Объекты AnimationSettings и ActionSettings изменяют поведение, настраивая его на воспроизведение при выборе слайда, на непрерывное воспроизведение во время просмотра слайда или на воспроизведение при наведении на него указателя мыши. Объект ActionSettings, который относится к щелчкам мыши и перемещению, не имеет дополнительных свойств для мультимедиа (см. раздел «Выполнение действий» ранее в этой главе). Однако объект AnimationSettings имеет ряд свойств, относящихся к мультимедиа.

Рисунок 10 отображает диалоговое окно «Настраиваемая анимация» с выбранной вкладкой «Настройки мультимедиа». На этой вкладке задаются свойства объекта AnimationSettings PlaySettings. Объект PlaySettings содержит ряд свойств, которые определяют, как воспроизводится медиаклип во время слайд-шоу. Флажок с надписью «Воспроизведение с использованием порядка анимации» соответствует логическому свойству PlayOnEntry. Если PlayOnEntry имеет значение True, он воспроизводит медиаклип при отображении слайда (на основе свойства AnimationOrder объекта AnimationSettings). PlayOnEntry также учитывает другие свойства AnimationSettings, такие как AdvanceMode и AdvanceTime, если для анимации других объектов, стоящих выше по порядку, требуются щелчки мыши, то объекты, расположенные ниже по порядку, не анимируются до тех пор, пока щелчки мыши не заставят произойти предыдущие анимации. Установите для PlayOnEntry значение False, чтобы пользователь мог анимировать его вручную.

Два переключателя, задающие действие «Во время игры», соответствуют свойству PauseAnimation. Если установлено значение True (когда переключатель имеет надпись «Пауза слайд-шоу»), другие автоматические функции слайд-шоу ждут, пока воспроизведение видеоклипа не завершится. Если установлено значение False, другие автоматические функции воспроизводятся одновременно. Это может быть полезно, если вы хотите, чтобы несколько анимированных GIF-файлов воспроизводились одновременно. Установите для PauseAnimation значение True, если у вас есть только один объект для воспроизведения и вы не хотите, чтобы слайд-шоу продолжалось до завершения воспроизведения объекта.

В диалоговом окне «Настройка анимации» на вкладке «Настройки мультимедиа» есть кнопка «Дополнительные параметры», которая соответствует дополнительным свойствам анимации. Флажки соответствуют свойствам LoopUntilStopped и RewindMovie. Чтобы зациклить анимацию до тех пор, пока пользователь не выберет другое действие (или автоматическое выполнение другого действия не будет запланировано), установите для свойства LoopUntilStopped значение True. Установка значения False запускает его один раз. Свойство RewindMovie определяет, возвращает ли фильм представление первого кадра после завершения (True) или остается ли фильм на последнем кадре (False).

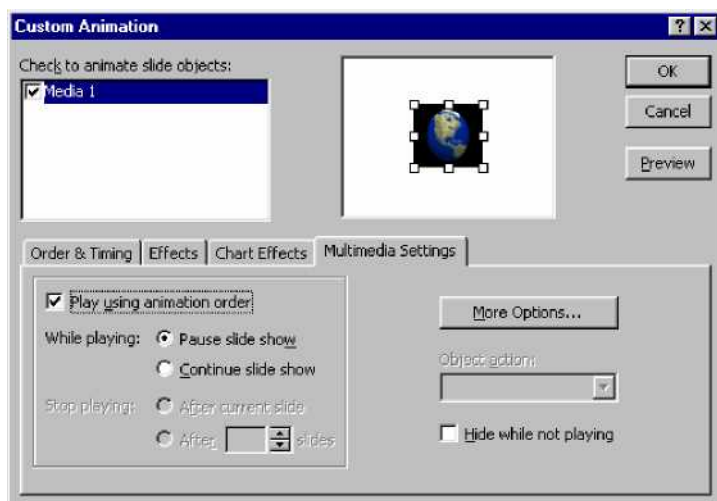


Рисунок 10. Настройки пользовательской анимации мультимедиа. На вкладке «Настройки мультимедиа» показаны свойства, доступные для медиаклипов.

Распространенным сценарием является непрерывное воспроизведение видео при выборе слайда. Следующий код добавляет объект видеоклипа и устанавливает соответствующие свойства:


```
oShape = oSlide.Shapes.AddMediaObject( ;  
    _SAMPLES + "\Solution\Forms\Fox.avi", 240, 156)  
  
WITH oShape.AnimationSettings.PlaySettings  
    .PlayOnEntry = .T.  
    .LoopUntilStopped = .T.  
ENDWITH
```



Многим свойствам, таким как PlayOnEntry, можно задать логическое значение (.T. или .F.) или числовое значение (1 или -1). Однако когда вы их читаете, они всегда будут числовыми. Это «особенность» того, как логические значения FoxPro преобразуются в числовое свойство; всегда будет возвращаться собственное числовое значение.

11.4 Добавление примечаний

Примечания можно вводить для каждого слайда. Примечания отображаются на нижней панели (в PowerPoint 2000) или распечатываются с изображением слайдов вверх и примечаний вниз, по одному слайду на странице. Примечания можно использовать в качестве заметок докладчика или распечатать и раздать в качестве раздаточного материала, при этом примечания будут сопровождаться комментариями к изображениям слайдов.

Объяснить, как размещать заметки на каждом слайде, на самом деле будет проще, если мы начнем с объяснения мастера заметок. На рисунке 11 показано основное представление заметок в PowerPoint. Как и коллекция Placeholders для слайдов (обсуждаемая в главе 10 «Основы PowerPoint»), индексы коллекции Placeholders имеют смысл, когда мы видим их в главном представлении заметок: 1 – область заголовка, 2 – область даты, 3 – объект SlideMaster, 4 – область тела заметок, 5 – область нижнего колонтитула и 6 – область чисел (Microsoft не предоставила для них констант, но вы, безусловно, можете определить свои собственные). Как и в случае с мастер-слайдами (см. «Достижение согласованности с мастер-слайдами» ранее в этой главе), эти заполнители можно форматировать с использованием цвета, шрифтов и стандартизированного текста. Объект NotesMaster содержит свойства, к которым осуществляется доступ при печати страниц заметок.

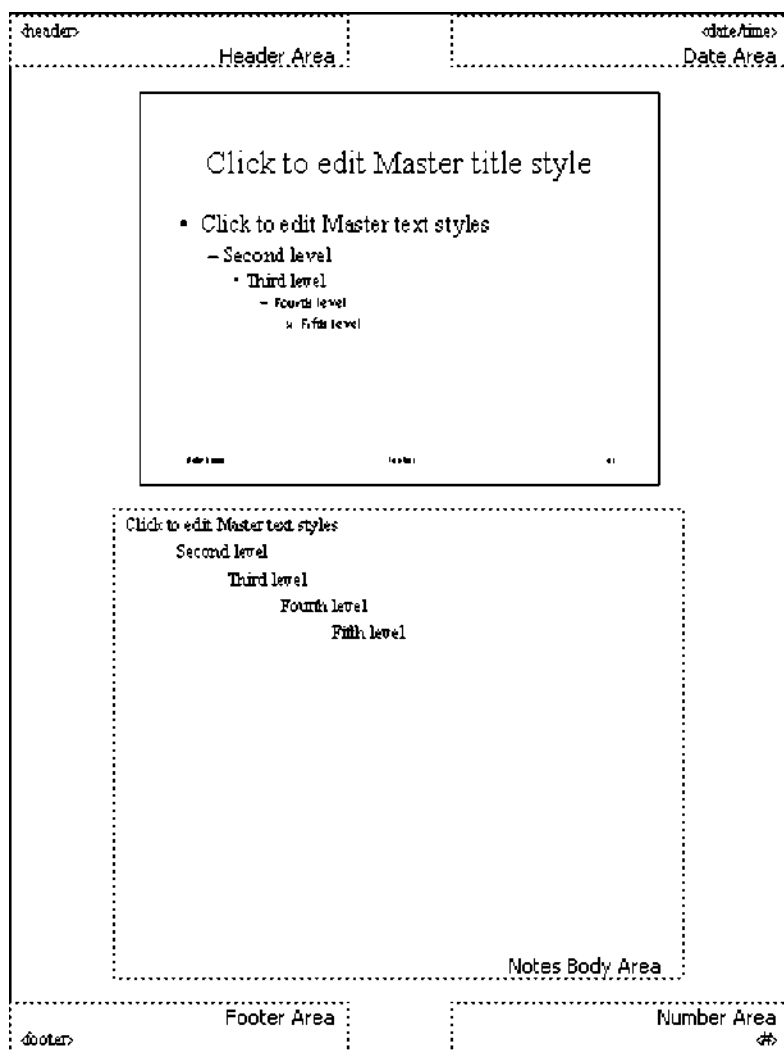


Рисунок 11. Главная страница заметок, как показано в PowerPoint. Заполнители заметок располагаются слева направо и сверху вниз.

Примечания можно добавлять к каждому слайду с помощью объекта NotesPage. Объект NotesPage содержит коллекцию из двух объектов Shape: один для изображения слайда, а второй для текста. При просмотре мастера заметок эти две фигуры меняются на каждой странице, а остальные остаются постоянными (номер слайда можно задать в поле, которое рассчитывает номер слайда). Изображение слайда имеет индекс 1, а текст – индекс 2 (опять же, здесь нет констант, выпускаемых Microsoft). Следующие строки кода добавляют пример текста заметки докладчика для слайдов 1 и 2:

```
WITH oPresentation
    .Slides[1].NotesPage.Shapes.Placeholders[2].TextFrame.TextRange.Text = ;
    "Не забудьте поприветствовать публику." + CHR(13) + ;
    "Представьте компанию."
    .Slides[2].NotesPage.Shapes.Placeholders[2].TextFrame.TextRange.Text = ;
    "Объясните маркетинговый слоган."
ENDWITH
```

Вы можете форматировать заметки так же, как и любой другой текст. Дополнительную информацию о форматировании текста см. в разделе «Добавление и форматирование текста» в главе 10, а также в разделе «Стандартизация внешнего вида текста» ранее в этой главе.

11.5 Собираем все вместе



В главе 10 «Основы PowerPoint» мы подготовили небольшое слайд-шоу, демонстрирующее рассмотренные нами темы. В этой главе мы используем те же основные слайды и добавляем множество дополнительных функций, которые мы рассмотрели. В листинге 1 (PPTSample2.PRG в файлах для загрузки для разработчиков, доступных на сайте www.hentzenwerke.com) показан код, а на рисунке 12, рисунке 13 и рисунке 14 показаны готовые слайды. Обратите внимание, что после добавления цвета, звука и анимации этот код необходимо запустить, чтобы увидеть презентацию. Черно-белые снимки экрана не могут показать все функции.

Листинг 1. Пример презентации для тасманских трейдеров.

```
* PPTSample2.PRG
* Copyright (c)2000, Tamar Granor and Della Martin
* From _Automating Microsoft Office with Visual FoxPro_,
* Hentzenwerke Publishing. www.hentzenwerke.com

* Это пример «Собираем все вместе» из главы 11.
* Образец презентации для тасманских трейдеров.
CLOSE DATA

#DEFINE ppTitleStyle          2
#DEFINE ppBodyStyle           3
#DEFINE ppLayoutTitle         1
#DEFINE ppLayoutText          2
#DEFINE ppEffectCoverDown     1284
#DEFINE ppEffectDissolve      1537
#DEFINE ppEffectAppear        3844
#DEFINE ppAnimateByFirstLevel 1
#DEFINE ppAdvanceOnTime       2
#DEFINE msoGradientHorizontal 1
#DEFINE ppBulletArabicPeriod  3
#DEFINE msoLineThickBetweenThin 5
#DEFINE autoIn2Pts            72

#DEFINE rgbDarkBlue    RGB( 0, 0, 138)
#DEFINE rgbMediumBlue  RGB( 96, 96, 204)
#DEFINE rgbPaleGray    RGB(192, 192, 192)
#DEFINE rgbYellow      RGB(255, 255, 0)
#DEFINE rgbBurgundy    RGB(128, 0, 0)
#DEFINE rgbLineColor   RGB(255, 255, 0)

SET PATH TO (_SAMPLES + "\TasTrade\Data") ADDITIVE

*****
* Настройка данных
*****
OPEN DATABASE TasTrade
LogoFile = (_SAMPLES + "\TasTrade\Bitmaps\TTradeSm.bmp")

USE OrdItems IN 0
```

```
USE Products IN 0
```

```
* Выберите 5 самых продаваемых товаров всех времен.
```

```
SELECT TOP 5 ;  
    P.English_Name, ;  
    SUM(O.Unit_Price * O.Quantity) AS TotQuan ;  
FROM OrdItems O, Products P ;  
WHERE O.Product_ID = P.Product_ID ;  
GROUP BY 1 ;  
ORDER BY 2 DESC ;  
INTO CURSOR TopSellers
```

```
* Выберите количество продуктов.
```

```
SELECT Count(*) ;  
FROM Products ;  
INTO ARRAY aProducts
```

```
* Удалите все существующие ссылки на серверы.
```

```
* Это предотвращает потерю памяти оставшимися экземплярами.
```

```
RELEASE ALL LIKE o*
```

```
* В демонстрационных целях сделайте oPowerPoint и oPresentation
```

```
* доступными после выполнения этой программы.
```

```
PUBLIC oPowerPoint, oPresentation
```

```
* Открыть PowerPoint
```

```
oPowerPoint = CreateObject("PowerPoint.Application")  
oPowerPoint.Visible = .T.
```

```
* Создать презентацию
```

```
oPresentation = oPowerPoint.Presentations.Add()
```

```
SlideNum = 1
```

```
*****
```

```
* МАСТЕР СЛАЙД
```

```
*****
```

```
WITH oPresentation.SlideMaster
```

```
    * Установите для фона градиентную заливку
```

```
WITH .Background.Fill  
    .ForeColor.RGB = rgbMediumBlue  
    .BackColor.RGB = rgbPaleGray  
    .TwoColorGradient(msoGradientHorizontal, 1)  
ENDWITH
```

```
* Установить заголовки
```

```
WITH .TextStyles[ppTitleStyle].Levels[1].Font  
    .Name = "Arial"  
    .Shadow = .T.  
    .Color.RGB = rgbDarkBlue  
ENDWITH
```

```

* Установить уровни стиля основной части
WITH .TextStyles[ppBodyStyle]
  WITH .Levels[1]
    WITH .Font
      .Name      = "Arial"
      .Bold      = .T.
      .Color.RGB = rgbBurgundy
      .Size      = 20
    ENDWITH
  WITH .ParagraphFormat.Bullet
    * Установите для маркера точку (используйте 149, если
    * используете PowerPoint 97).
    .Character = 8226
    .Visible   = .F.    && Не показывать маркер!
  ENDWITH
ENDWITH
WITH .Levels[2]
  WITH .Font
    .Name      = "Arial"
    .Color.RGB = rgbDarkBlue
    .Size      = 20
  ENDWITH
  * Установите для маркера точку (используйте 149, если
  * используете PowerPoint 97).
  .ParagraphFormat.Bullet.Character = 8226

ENDWITH

ENDWITH

* Добавьте логотип.
.Shapes.AddPicture(LogoFile, .F., .T., ;
  8.5 * autoIn2Pts, 6.0 * autoIn2Pts)

* Пользователи PowerPoint 97: последние два параметра,
* высота и ширина, не являются обязательными.
* Вместо этого используйте этот код:
*.Shapes.AddPicture(LogoFile, .F., .T., ;
*   8.5 * autoIn2Pts, 6.0 * autoIn2Pts, ;
*   1.0 * autoIn2Pts, 1.0 * autoIn2Pts)

ENDWITH

```

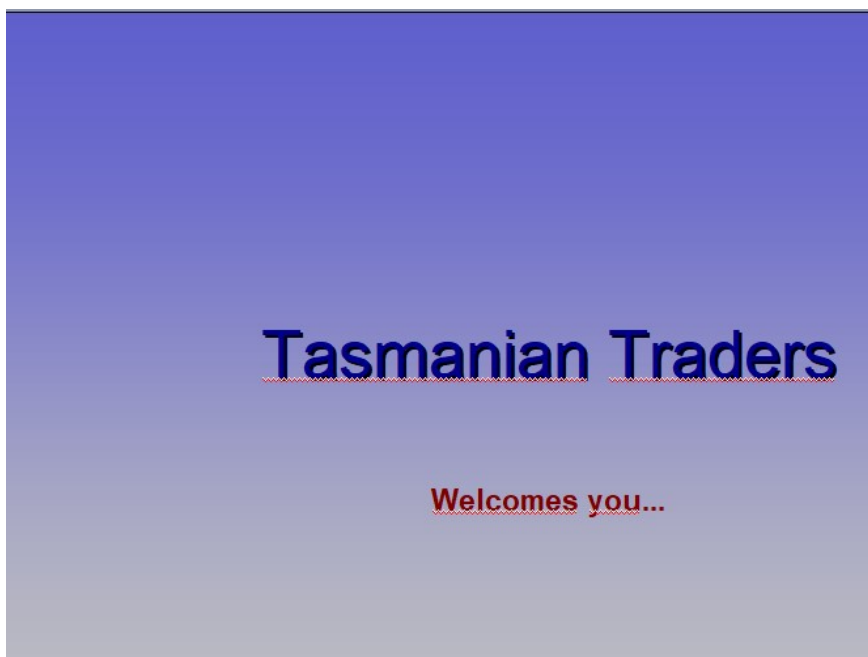


Рисунок 12. Образец титульного слайда «Тасманийские торговцы». Фон градиентной заливки, шрифты и цвета заголовков, а также логотип на каждом слайде создаются с помощью мастер-слайда.

```
*****
* ТИТУЛЬНЫЙ СЛАЙД
*****
* Добавить слайд.
oSlide = oPresentation.Slides.Add(SlideNum, ppLayoutTitle)

* Установите текст заголовка.
oSlide.Shapes[1].TextFrame.TextRange.Text = "Tasmanian Traders"

* Установите текст подзаголовка
oSlide.Shapes[2].TextFrame.TextRange.Text = "Welcomes you..."

WITH oSlide.SlideShowTransition
    .EntryEffect = ppEffectCoverDown
    .AdvanceOnTime = .T.
    .AdvanceTime   = 2
ENDWITH

SlideNum = SlideNum + 1
```



Рисунок 13. Пример второго слайда «Тасманские трейдеры». Как и в случае с первым слайдом, в коде минимальное форматирование, поскольку большую часть форматирования берет на себя SlideMaster.

```
*****
```

```
* SECOND SLIDE
```

```
*****
```

```
* Добавить слайд.
```

```
oSlide = oPresentation.Slides.Add(SlideNum, ppLayoutTitle)
```

```
* Переместите слайд на передний план.
```

```
oSlide.Select()
```

```
* Пользователи PowerPoint 97: oSlide.Select()
```

```
* выдаст ошибку. Вместо этого используйте эту строку:
```

```
* oPresentation.ActiveWindow.View.GoToSlide(2)
```

```
* Установите текст заголовка.
```

```
oSlide.Shapes[1].TextFrame.TextRange.Text = "Tasmanian Traders "+  
CHR(13) + "has what you need"
```

```
* Переместите заголовок примерно на полдюйма вверх
```

```
WITH oSlide.Shapes[1]
```

```
.Top = .Top - (.5 * autoIn2Pts)
```

```
ENDWITH
```

```
* Добавьте линию на полдюйма ниже заголовка по центру и длиной 6
```

```
* дюймов
```

```
LineTop = oSlide.Shapes[1].Top + oSlide.Shapes[1].Height + ;  
(.5 * autoIn2Pts)
```

```
LineLeft = 2 * autoIn2Pts
```

```
LineEnd = LineLeft + (6.0 * autoIn2Pts)
```

```
oLine = oSlide.Shapes.AddLine(LineLeft, LineTop, LineEnd, LineTop)
```

```
* Отформатируйте линию как толстую линию
```



```

* с двумя тонкими линиями с каждой стороны.
WITH oLine.Line
  .ForeColor.RGB = rgbLineColor
  .Style = msoLineThickBetweenThin
  .Weight = 8
ENDWITH

* Установите текст подзаголовка и измените число на полужирный и синий
WITH oSlide.Shapes[2].TextFrame.TextRange
  .Text = "With a selection of " + ALLTRIM(STR(aProducts[1])) + ;
        " items, you're sure to be pleased."
  .Words[5].Font.Color = rgbDarkBlue
ENDWITH

WITH oSlide.SlideShowTransition
  .EntryEffect = ppEffectDissolve
  .AdvanceOnTime = .T.
  .AdvanceTime = 5
ENDWITH

SlideNum = SlideNum + 1

```



Рисунок 14. Пример третьего слайда «Тасманские трейдеры». Опять же, основная часть форматирования выполняется в SlideMaster. Шрифты, цвета и фоны чрезвычайно единообразны, а ваш код легко поддерживать.

```

*****
* СЛАЙД ТОП ПРОДАВЦОВ
*****
* Добавить слайд
oSlide = oPresentation.Slides.Add(SlideNum, ppLayoutText)

* Переместите слайд на передний план
oSlide.Select()

* Пользователи PowerPoint 97: oSlide.Select()
* выдаст ошибку. Вместо этого используйте эту строку::

```

```

* oPresentation.ActiveWindow.View.GoToSlide(2)

* Вставьте заголовок (обратите внимание на использование объекта
* Title вместо перечисляемого объекта формы).
oSlide.Shapes.Title.TextFrame.TextRange.Text = "Tasmanian Traders" + ;
    CHR(13) + "Top Sellers"

* Создайте строку, которая будет использоваться для 5 лучших продавцов.
* Используйте CR между каждым элементом, чтобы сделать каждый маркер
* отдельным.
BulletString = ""
SELECT TopSellers
SCAN
    BulletString = BulletString + ;
        TRIM(TopSellers.English_Name) + CHR(13) + "$" + ;
        ALLTRIM(TRANSFORM(TopSellers.TotQuan, "99,999,999")) +;
        CHR(13)
ENDSCAN

* Добавьте строку маркера в текстовый фрейм.
WITH oSlide.Shapes[2]
    WITH .TextFrame.TextRange
        .Text = BulletString

        * Отступ для всех квот продаж.
        FOR I = 1 TO 5
            .Lines[I*2, 1].IndentLevel = 2
        ENDFOR
    ENDWITH

* Переместите строку вправо примерно на 1,5 дюйма.
.Left = .Left + (1.5 * autoIn2Pts)

* Каждый маркер (с подчиненными) появляется с интервалом в 1 секунду
WITH .AnimationSettings
    .EntryEffect = ppEffectAppear
    .TextLevelEffect = ppAnimateByFirstLevel
    .AdvanceMode = ppAdvanceOnTime
    .AdvanceTime = 0.5
    .SoundEffect.Name = "Хлыст"    && "Whoosh"
ENDWITH
ENDWITH

oSlide.SlideShowTransition.EntryEffect = ppEffectDissolve

* Запустите слайд-шоу.
oPresentation.SlideShowSettings.Run()

```

В этой главе был рассмотрен визуальный мир PowerPoint. Мы рассмотрели многие часто используемые функции PowerPoint. Однако в PowerPoint есть гораздо больше, поэтому не стесняйтесь использовать средство записи макросов, обозреватель объектов и файлы справки, чтобы найти те функции, которые нужны вашему приложению.

Глава 12. Автоматизация Outlook

Представьте себе, что вы пишете приложения, которые отправляют вам электронное письмо в случае возникновения ошибки или помещают важные даты (например, дату, когда нужно оплатить чек) прямо в календарь пользователя.

Outlook часто напоминает нам о тех рекламных роликах, которые показывают по вечернему телевидению: он нарезает ломтиками, кубиками, соломкой. Outlook делает так много разных вещей, что их сложно классифицировать. На самом деле это гораздо больше, чем просто клиент электронной почты, никто из нас не использует его в таком режиме. Но это не просто календарь/планировщик или адресная книга. Во многих отношениях Outlook является заменой всей коллекции предметов, которые большинство из нас до сих пор хранят на своих столах: Rolodex® (вращающийся каталог с проиндексированными карточками), ежедневника, блокнота, телефона и коллекции стикеров Post-It™ (самоклеющаяся бумага для заметок). Хотя мы пока не готовы отказаться от любого из них, мы все больше и больше полагаемся на Outlook (по крайней мере, пока компьютер включен).

12.1 Объектная модель Outlook

Объектная модель Outlook сильно отличается от объектной модели других приложений Office. Это потому, что он вырос в другой семье и был усыновлен Office. Только в Office 2000 вы можете использовать VBA для написания макросов для Outlook, и до сих пор нет возможности записать макрос, как в других продуктах Office. (Конечно, это значительно усложняет понимание синтаксиса для Автоматизации Outlook. Вы не можете просто записать макрос и преобразовать код.)

Из-за разнообразной истории Outlook примеры в этой главе были протестированы только с Outlook 2000. Они вполне могут работать с более ранними версиями, но мы не даем никаких обещаний в этом отношении.

Как и в других приложениях Office, в Справке есть живая диаграмма объектной модели. На рисунке 1 показана главная страница диаграммы.

Как и в остальной части Office (и на многих других серверах), вверху находится объект Application, который представляет сам сервер. Это объект, который вы создаете с помощью CreateObject(). Мы рассмотрим его в следующем разделе «Знакомство с Outlook». Как и следовало ожидать, у него есть метод Quit для закрытия приложения. Однако нет ни свойства Visible, ни свойств Top и Left, указывающих, где расположено приложение, ни свойства StartupPath, указывающего, где следует разместить данные.

Однако еще более поразительно то, что, как только у вас есть объект Application, ни один объект не бросается в глаза так же явно, как ключ к объектной модели Outlook, как это делает Document в Word или Presentation в PowerPoint. Это связано с тем, что не существует какого-либо доминирующего объекта, и наиболее близким, вероятно, является MAPIFolder. Каждый MAPIFolder представляет одну из папок Outlook, поэтому существует объект MAPIFolder для календаря, другой для контактов, третий для папки «Входящие» и так далее. Объекты MAPIFolder собираются в коллекцию Folders. Чтобы усложнить (или запутать?), MAPIFolders могут быть вложенными. Фактически коллекция Folders, содержащая ключевые объекты Outlook, содержится в другом объекте MAPIFolder, который называется «Личные папки». Этот MAPIFolder содержится в еще одной коллекции Folders, которая содержится в объекте NameSpace. (Не волнуйтесь, если это покажется вам сложным. Позже мы углубимся в это на примерах.)

Microsoft Outlook Objects

See Also

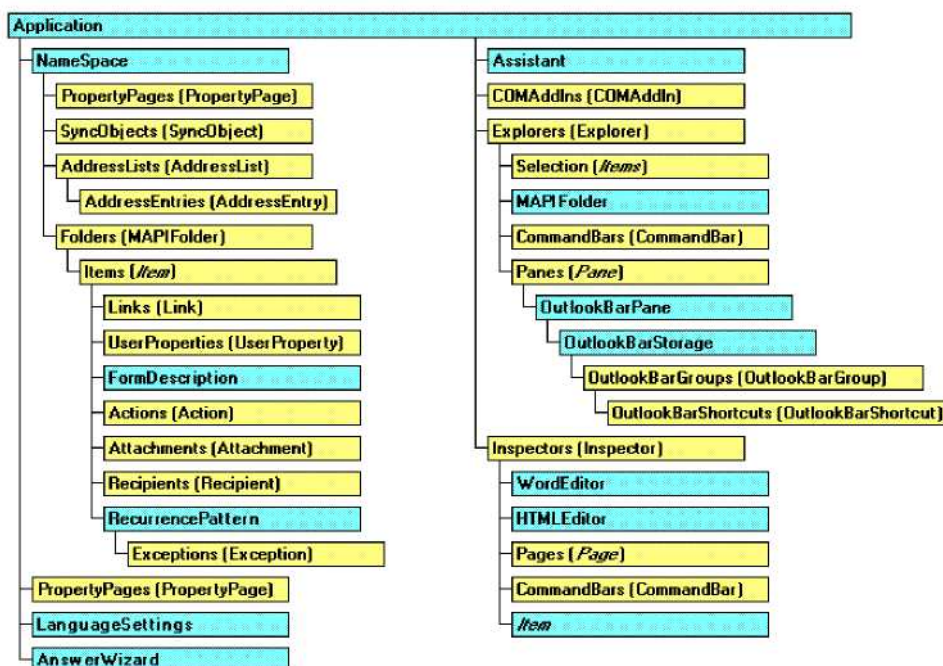


Рисунок 1. Объектная модель Outlook. В отличие от других объектов Office, нет очевидного кандидата на роль ключевого объекта под Application.

12.2 Знакомство с Outlook

Хотя мы используем те же команды для запуска Outlook, что и другие приложения Office, это еще одна область, где все одинаково, но и отличается. Опять же, функция CreateObject() VFP является ключом к запуску сервера Автоматизации Outlook. Однако, в отличие от других, если Outlook уже открыт, эта команда:

```
oOutlook = CreateObject("Outlook.Application")
```

присоединяется к работающему экземпляру Outlook, а не запускает новый. Это как если бы вы использовали GetObject(), а не CreateObject(). Фактически, вы получите тот же результат, выполнив:

```
oOutlook = GetObject(, "Outlook.Application")
```

Если Outlook еще не запущен, вызов CreateObject() создает экземпляр, как и следовало ожидать. Когда вы открываете Outlook таким образом, вы сталкиваетесь с еще одним отличием. Как отмечалось ранее, объект приложения Outlook не имеет свойства Visible. Чтобы сделать Outlook видимым и видеть, что вы делаете, вам необходимо создать и отобразить объект Explorer, лицо Outlook миру. Вот код:

```
#DEFINE olFolderDisplayNormal 0
oNameSpace = oOutlook.GetNameSpace("MAPI")
oExplorer = oOutlook.Explorers.Add(oNameSpace.Folders[1], ;
    olFolderDisplayNormal)
oExplorer.Activate()
```

Первая строка (после #DEFINE), скорее всего, будет первой строкой, которую вы вводите каждый раз при запуске нового экземпляра Outlook. Она создает объект NameSpace в Outlook. NameSpace находится на вершине иерархии объектов и необходим для спуска к другим объектам. (На самом деле, вы можете спуститься вниз с помощью Explorer, но для Автоматизации работы вы обычно будете работать за кулисами и Explorer вам не понадобится.)

Следующая строка создает проводник, сообщая ему отобразить первую папку в пространстве имен, созданную предыдущей строкой. Затем последняя строка сообщает Outlook, что нужно показать вновь созданный объект Explorer. Эта строка действительно эквивалентна строк `Visible = .T.` . Однако соответствующего метода деактивации не существует. После того, как вы активируете Explorer, единственный способ заставить его исчезнуть – это вызвать его метод `Close`, который не только скрывает его, но и уничтожает.

Что вы сможете с ним делать, если Outlook запущен? Это то, что нам предстоит выяснить.

12.3 Доступ к содержимому Outlook

Посмотрите еще раз на пример из предыдущего раздела. Первая настоящая строка кода довольно загадочна. Он вызывает метод объекта `Application` с именем `GetNameSpace`, передавая ему параметр `MAPI`. Что такое `NameSpace` и почему ему нужен «`MAPI`»?

Это один из тех вопросов, на который правильный ответ – «потому что». Если вы ищете объект `NameSpace` в файле справки Outlook VBA, первое предложение гласит: «Представляет абстрактный корневой объект для любого источника данных». Не очень полезно, не так ли? Читая дальше, вы обнаружите, что единственным источником данных, который поддерживает Outlook, является «`MAPI`», который представляет собой интерфейс программирования почтовых приложений и включает в себя все данные, хранящиеся в Outlook. Итог `NameSpace` – это просто объект, который обеспечивает шлюз для остальных данных Outlook. Возможно, в какой-то момент разработчики Outlook решили, что они будут поддерживать различные хранилища данных.

Мы не уверены, почему вам нужно активно вызывать `GetNameSpace` для загрузки данных вместо того, чтобы загружать их при запуске Outlook. Но вы это делаете, так что просто привыкните делать это автоматически.

Как только вы преодолеете этот момент, все станет немного интереснее. Объект `NameSpace` обеспечивает доступ к объектам `Folder`, которые являются сердцем Outlook. Есть несколько способов войти в эти папки. Вы можете просто спуститься вниз по иерархии. `NameSpace` имеет коллекцию `Folders`, содержащую один или несколько элементов, в зависимости от вашей конфигурации. Для автономного пользователя единственным элементом коллекции являются "Personal Folders" («Личные папки»). При наличии Exchange Server коллекция может также содержать элементы "Public Folders" и "Mailbox" («Общие папки» и «Почтовый ящик»). Каждая из этих папок имеет свои собственные коллекции папок, содержащие папку для каждого отдельного апплета в Outlook. Есть папки `Calendar`, `Tasks`, `Inbox`, `Outbox`, `Contacts` («Календарь», «Задачи», «Входящие», «Исходящие», «Контакты») и т. д. Вы можете получить к ним доступ, например, если вы сохранили ссылку на объект `NameSpace` в `oNameSpace`, как показано ранее:

```
oCalendar = oNameSpace.Folders[1].Folders["Calendar"]
oInbox = oNameSpace.Folders[1].Folders["Inbox"]
```

и т. д.

Однако есть более простой способ добраться до отдельных папок. `NameSpace` имеет метод `GetDefaultFolder`, которому можно передать соответствующую константу и получить взамен ссылку на указанную папку. В таблице 1 показаны константы для 10 типов папок, поддерживаемых Outlook.

Таблица 1. Папки Outlook. Эти константы представляют 10 основных типов папок в Outlook. Передайте их в GetDefaultFolder для доступа к данным Outlook.

Константа	Значение	Константа	Значение
olFolderDeletedItems	3	olFolderContacts	10
olFolderOutbox	4	olFolderJournal	11
olFolderSentMail	5	olFolderNotes	12
olFolderInbox	6	olFolderTasks	13
olFolderCalendar	9	olFolderDrafts	16

Чтобы использовать этот подход, вызовите метод следующим образом:

```
#DEFINE olFolderTasks 13
#DEFINE olFolderContacts 10
oTasks = oNameSpace.GetDefaultFolder( olFolderTasks )
oContacts = oNameSpace.GetDefaultFolder( olFolderContacts )
```

Получив папку для одного из апплетов Outlook, вы можете изучить ее содержимое. Каждая папка содержит набор предметов, но детали существенно различаются. Здесь мы рассмотрим те из них, которые, по нашему мнению, вы, скорее всего, захотите автоматизировать, но не будем подробно рассматривать все 10. Если вам нужно работать с другими, файл справки Outlook VBA расскажет вам все, что вам нужно знать. (На самом деле, как только вы освоитесь, поиск конкретных свойств не составит большого труда.)

Достигнув папки определенного апплета Outlook, вам все равно придется спускаться по коллекции Items, чтобы добраться до фактического содержимого. Например, чтобы узнать, сколько задач находится в папке Tasks, вы должны проверить:

```
? oTasks.Items.Count
```

Новые элементы каждого типа создаются с помощью метода CreateItem Application. Вы передаете ему константу (или фактическое значение) для типа элемента, который хотите создать, и он добавляет участника в соответствующую папку. В таблице 2 показаны константы, которые нужно передать в CreateItem.

Таблица 2. Создание новых предметов. Метод CreateItem добавляет данные. Передайте одну из этих констант, чтобы создать элемент указанного типа.

Константа	Значение	Константа	Значение
olMailItem	0	olJournalItem	4
olAppointmentItem	1	olNoteItem	5
olContactItem	2	olPostItem	6
olTaskItem	3	olDistributionListItem	7

Например, чтобы создать новую встречу, выполните следующий вызов:

```
oAppt = oOutlook.CreateItem( olAppointmentItem )
```

Затем вы можете изменить свойства новой встречи, такие как Start, End, AllDayEvent и т. д., чтобы правильно ввести ее в календарь. Остальные типы предметов

обрабатываются аналогичным образом.

Заполнив свойства элемента, вызовите его метод Save, чтобы сохранить его в соответствующей коллекции. Каждый раз, когда вы меняете свойство, вам нужно снова вызвать Save, чтобы сохранить изменения.

Вы можете удалить элемент, вызвав его метод Delete. Этот метод перемещает элемент в папку DeletedItems, из которой его можно при необходимости восстановить, но со временем он будет очищен. (Папку DeletedItems можно настроить на автоматическую очистку при выходе или, как и в случае с корзиной, ее можно очистить вручную.)

12.4 Отправка электронной почты.

Первое, что каждый хочет автоматизировать с помощью Outlook, – это отправка сообщений по электронной почте, будь то отправка отчетов об ошибках разработчикам, насыщение потенциальных клиентов предложениями или просто общение между различными сайтами, на которых работает приложение.

Ключевым объектом для отправки и получения сообщений является MailItem, который представляет собой одно почтовое сообщение. Чтобы создать новый объект MailItem, вызовите CreateItem, передав olMailItem (0) в качестве параметра:

```
#DEFINE olMailItem 0
oMailItem = oOutlook.CreateItem( olMailItem )
```

Таблица 3 перечисляет ключевые свойства MailItem.

Этот код создает простое сообщение одному человеку:

```
#DEFINE CR CHR(13)
WITH oMailItem
    .Subject = "Встреча на следующей неделе"
    .Body = "Просто подтверждаю нашу встречу во вторник в 3." + ;
        CR + "Пожалуйста, принесите идеи, а также свои заметки " + ;
        "с прошлогоднего мероприятия."
    .Recipients.Add("Билл Гейтс")
ENDWITH
```

Таблица 3. Что в сообщении? Вот свойства MailItem, с которыми вы, скорее всего, будете работать.

Свойство	Тип	Описание
Subject	Character	Описание сообщения. Его можно использовать в качестве индекса коллекции Items.
Body	Character	Содержание сообщения.
Recipients	Object	Ссылка на коллекцию объектов Recipient с одной записью для каждого человека, который получит это сообщение. (См. следующий раздел «Получатели и контакты».)
To	Character	Список получателей, разделенный точкой с запятой. Хотя вы можете указать список получателей, заполнив это свойство, это не рекомендуется. Вместо этого используйте .Recipients.Add(). (См. следующий раздел «Получатели и контакты».)

Свойство	Тип	Описание
CC	Character	Разделенный точкой с запятой список людей, получивших копии сообщения (список cc:). Предостережение в свойстве To применимо и к CC.
BCC	Character	Список людей, получающих скрытые копии сообщения, разделенный точкой с запятой (список скрытых копий). Предостережение в свойстве To применимо и к BCC.
Importance	Numeric	Приоритет сообщения (соответствует полю «Важность» в диалоговом окне «Параметры»). Использует эти константы: olImportanceLow 0 olImportanceNormal 1 olImportanceHigh 2
Sensitivity	Numeric	Уровень конфиденциальности сообщения (соответствует полю «Чувствительность» в диалоговом окне «Параметры»). Использует эти константы: olNormal 0 olPersonal 1 olPrivate 2 olConfidential 3
Attachments	Object	Ссылка на коллекцию объектов Attachment, по одному на каждый прикрепленный файл. (См. «Прикрепление файлов» далее в этой главе.)

Как и в случае с другими элементами, после заполнения свойств вы можете вызвать метод `Save` для сохранения нового сообщения. Однако новое сообщение не сохраняется автоматически в папке «Исходящие». Папка, в которой хранятся неотправленные сообщения, определяется настройкой в диалоговом окне параметров Outlook. На рисунке 2 показано диалоговое окно «Дополнительные параметры электронной почты», в котором пользователи делают этот выбор. Доступ к этому диалоговому окну осуществляется путем выбора в меню «Инструменты|Параметры», затем выбора «Параметры электронной почты» на вкладке «Предпочтения», а затем выбора «Дополнительные параметры электронной почты» в появившемся диалоговом окне.

Однако сохранения сообщения недостаточно для его отправки получателю. Вам также необходимо вызвать метод `Send`. (На самом деле вы можете просто вызвать `Send` и вообще пропустить `Save`.) Этот метод, как следует из названия, немедленно отправляет сообщение, направляя его в папку `Sent Messages` («Отправленные сообщения»).

```
oMailItem.Save()
oMailItem.Send()
```

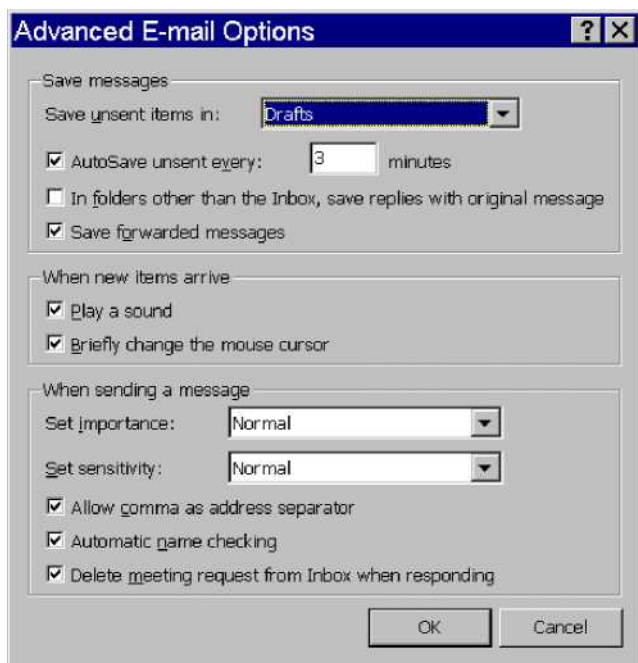


Рисунок 2. Указание параметров электронной почты. В этом диалоговом окне можно указать, где метод Save будет хранить новые почтовые сообщения. Их можно поместить в любую из нескольких папок, включая Drafts и Outbox («Черновики» и «Исходящие»).

Если Outlook не подключен к почтовому серверу, возникает ошибка отправки (хотя вы не получаете никаких указаний на ошибку в VFP), и сообщение попадает в папку Outbox («Исходящие»), готовое к отправке при следующем подключении. (На самом деле оно может оказаться в папке Drafts («Черновики»), причину почему смотрите в следующем абзаце.) Однако в этот момент, в зависимости от конфигурации Outlook, сообщение не обязательно отправляется автоматически пользователю, возможно, придется предпринять некоторые действия, чтобы запустить процесс отправки и получения сообщений. (Ничто из этого на самом деле не имеет большого отношения к самому Outlook, а лишь к тому, как Outlook настроен для отправки и получения почты, и ничто из этого не влияет на код, который вы пишете в VFP для отправки почты.)

Если вы следовали примерам и по какой-то причине в вашей адресной книге Outlook нет Билла Гейтса, эта команда «Отправить» привела к другой ошибке, которая отображалась в VFP: «Код исключения OLE IDispatch 4096 из Microsoft Outlook». : Outlook не распознает одно или несколько имен». (Конечно, если в вашей адресной книге Outlook есть Билл Гейтс, вам, вероятно, не следовало отправлять это сообщение, если только у вас не назначена встреча с ним на вторник в 3.) Вторая часть сообщения на самом деле довольно ясна. Это означает, что Outlook попросили отправить кому-то сообщение, но этого человека нет в папке Contacts («Контакты»). К счастью, есть способ предотвратить эту ошибку до того, как она произойдет. В следующем разделе показано, как это сделать.

Получатели и контакты

Как показано в Таблице 3, сообщение может быть отправлено более чем одному человеку, и любое лицо может быть включено в любой из трех списков: основной список, список CC («копия») или список BCC («скрытая копия»). Однако все они управляются через одну коллекцию под названием Recipients. Свойство Type объекта Recipient указывает для каждого получателя, к какому списку он принадлежит. Возможные варианты: olTo (1), olCC (2) и olBCC (3).

Чтобы добавить человека в список получателей сообщения, используйте метод Add коллекции Recipients. Он ожидает добавления одного параметра: имени или адреса электронной почты человека. Outlook имеет множество встроенных функций для сопоставления этой информации с папкой Contacts. Если будет хотя бы полшанса, он выполнит свою работу правильно. Для этого процесса используются методы Resolve и ResolveAll. Resolve относится к объекту Recipient, а ResolveAll – к методу коллекции. Оба пытаются сопоставить текущую информацию имени/адреса с папкой Contacts и заполнить свойства Name и Address, а также создать объект AddressEntry для получателя, когда совпадение найдено. Resolve может работать даже с частичным именем или фамилией, если они уникальны.

```
#DEFINE olMailItem 0
#DEFINE CR CHR(13)
#DEFINE olCC 2

oMailItem = oOutlook.CreateItem( olMailItem )

WITH oMailItem
    .Subject = "Присылайте идеи прямо сейчас!"
    .Body = "Если у вас есть какие-либо идеи о том, " + CR + ;
           "как продолжить этот проект, пришлите их " + ;
           "всей группе прямо сейчас."

    * Добавьте получателей
    WITH .Recipients
        .Add("Фред Смит")
        .Add("Мэри Луиза МакГилликадди")
        .Add("Дарт") && наверняка единственный Дарт, поэтому
                     && фамилия не нужна
        .Add("Бэзил Рэтбоун")

    * Скопируйте босса
    oRecip = .Add("Ruth Less")
    oRecip.Type = olCC

    * Заставьте Outlook добавить все адреса электронной почты
    lResolved = .ResolveAll()
ENDWITH

* Сохраните и отправьте
.Save()
.Send()
ENDWITH
```

Конечно, вы не можете запустить этот пример как есть. Вам нужно будет заменить людей, которые действительно есть в вашей адресной книге Outlook (чтобы избежать сообщения об ошибке, описанного ранее). Даже в этом случае вы, возможно, не захотите запускать весь пример, поскольку они будут задаваться вопросом, почему вы отправляете им такое странное сообщение. К счастью, вы можете прекратить отправку сообщения (то есть не выполняйте метод «Отправить») и просто проверьте его в папке Drafts или Outbox (или в любой другой папке, в которой у вас есть Outlook, в которой сохраняются неотправленные сообщения), а затем удалите его перед отправкой.



Мы столкнулись с проблемой метода `ResolveAll` в ситуациях, когда не каждый получатель был найден. Должно произойти следующее: адрес электронной почты заполняется для тех получателей, у которых есть соответствующие записи в контактах, а их свойству `Resolved` присвоено значение `.T.` . Мы обнаружили, что в некоторых случаях для параметра `Resolved` не установлено значение `.T.`, однако вызов метода `Resolve` для отдельной записи получателя возвращает `.T.` . Поэтому мы рекомендуем использовать `Resolve`, а не `ResolveAll`, если есть вероятность, что для некоторых получателей проблема не сможет быть решена. См. пример в разделе «Собираем все вместе» в конце этой главы.

Прикрепление файлов

Помимо текста, вы можете отправить файлы вместе с сообщением. Коллекция `Attachments` позволяет вам это сделать. Используйте метод `Add` для прикрепления каждого файла. При этом в коллекции создается новый объект `Attachment`. Передайте имя файла, включая путь, по которому его можно найти, в метод `Add`.

```
oMailItem.Attachments.Add( "C:\My Documents\QuarterlyReport.DOC" )
```

Если вы добавляете вложения к уже сохраненному (но не отправленному) сообщению, не забудьте сохранить сообщение еще раз. Если вы попытаетесь добавить вложение к существующему сообщению, а файл не существует, элемент все равно будет добавлен в коллекцию `Attachments` («Вложения»). (Если вы проверите `.Attachments.Count`, вы увидите, что оно увеличилось.) Прежде чем продолжить, обязательно используйте метод `Delete` или `Remove`, чтобы избавиться от ложного вложения.

12.5 Управление папкой *Contacts*.

В Outlook есть встроенная адресная книга — папка `Contacts` («Контакты»). Это похоже на супер-Ролодекс, способный отслеживать несколько адресов, номеров телефонов и адресов электронной почты каждого человека, представленного объектом `ContactItem`.

Удивительно, но Outlook не использует коллекции для управления различными адресами, номерами телефонов и адресами электронной почты. Для каждого используются отдельные свойства, а количество каждого типа ограничено (три адреса, 19 номеров телефонов, три адреса электронной почты). В таблице 4 перечислены некоторые свойства `ContactItem`. Дополнительную информацию см. в справке, но шаблон должен быть очевиден из таблицы.

Таблица 4. Я в книге. Объект `ContactItem` представляет человека в адресной книге Outlook.

Свойство	Тип	Описание
<code>FullName</code>	<code>Character</code>	Полное имя человека, неанализированное. Имя также доступно в различных конфигурациях, таких как <code>LastNameAndFirstName</code> , <code>LastFirstAndSuffix</code> и т. д. Полный список смотрите в справке.
<code>FirstName</code>	<code>Character</code>	Имя человека.
<code>LastName</code>	<code>Character</code>	Фамилия человека.

Свойство	Тип	Описание
HomeAddress	Character	Домашний адрес человека, неразобранный. Как и в названии, компоненты доступны индивидуально. Outlook также способен выполнять сложный анализ предоставленного ему адреса, чтобы разбить его на компоненты.
HomeAddressStreet	Character	Часть улицы (первая строка) домашнего адреса.
HomeAddressCity	Character	Часть домашнего адреса, где указан город.
HomeAddressState	Character	Часть домашнего адреса, содержащая штат.
HomeAddressCountry	Character	Страна проживания.
HomeAddressPostalCode	Character	Почтовый индекс домашнего адреса.
BusinessAddress	Character	Рабочий адрес человека, неанализированный. Доступны те же отдельные компоненты, что и для домашнего адреса.
SelectedMailingAddress	Numeric	Указывает, какой адрес является основным адресом для этого человека. Использует эти константы: oINone 0 oIBusiness 2 oIHome 1 oIOther 3
HomeTelephoneNumber	Character	Номер домашнего телефона человека.
BusinessTelephoneNumber	Character	Рабочий номер телефона человека.
HomeFaxNumber	Character	Номер домашнего факса человека.
BusinessFaxNumber	Character	Номер делового факса человека.
PagerNumber	Character	Номер пейджера человека.
CarTelephoneNumber	Character	Номер телефона автомобиля человека.
PrimaryTelephoneNumber	Character	Основной номер телефона, который будет использоваться этим человеком.
Email1Address	Character	Первый адрес электронной почты этого человека.
Birthday	Datetime	День рождения человека.



Программа в листинге 1 считывает папку Contacts и создает курсор с записью для каждого человека, содержащей имя, выбранный почтовый адрес и основной номер телефона. Вы найдете его как MakeContactList.PRG в файлах загрузки для разработчиков, доступных по адресу www.hentzenwerke.com.

Листинг 1. Сбор контактной информации. Эта программа считывает контакты из Outlook и помещает их в курсор VFP.

```
* MakeContactList.PRG
* © 2000, Tamar E. Granor and Della Martin
* From: Microsoft Office Automation with Visual FoxPro
* Hentzenwerke Publishing. www.hentzenwerke.com

* Чтение контактной информации Outlook в курсор.
#DEFINE olContacts 10
#DEFINE olNone      0
#DEFINE olHome      1
#DEFINE olBusiness  2
#DEFINE olOther      3

LOCAL oNameSpace, oContacts, oContact
LOCAL cFirst, cLast, cAddr, cPhone

IF VarType(oOutlook) <> "O"
    * Запустите или подключитесь к Outlook.
    * Сделайте его общедоступным для демонстрационных целей.
    RELEASE oOutlook
    PUBLIC oOutlook
    oOutlook = CreateObject("Outlook.Application")
ENDIF
oNameSpace = oOutlook.GetNameSpace("MAPI")

* Получить папку "Контакты".
oContacts= oNameSpace.GetDefaultFolder( olContacts )

* Создайте курсор для хранения контактной информации.
CREATE CURSOR ContactInfo ;
    (cFirstName C(15), cLastName C(20), mAddress M, ;
     cPhoneNum C(30))

* Пройтись по контактам
FOR EACH oContact IN oContacts.Items
    WITH oContact
        cFirst = .FirstName
        cLast = .LastName
        cPhone = .PrimaryTelephoneNumber

        * Выберите правильный адрес. Если основной номер телефона
        * пуст, выберите номер телефона, связанный с этим адресом.
        DO CASE
        CASE .SelectedMailingAddress = olHome
            cAddr = .HomeAddress
            IF EMPTY(cPhone)
                cPhone = .HomeTelephoneNumber
            ENDIF
        CASE .SelectedMailingAddress = olBusiness
            cAddr = .BusinessAddress
            IF EMPTY(cPhone)
                cPhone = .BusinessTelephoneNumber
            ENDIF
        CASE .SelectedMailingAddress = olOther
```

```

        cAddr = .OtherAddress
        IF EMPTY(cPhone)
            cPhone = .OtherTelephoneNumber
        ENDIF
        CASE .SelectedMailingAddress = olNone
            cAddr = ""
        ENDCASE

        INSERT INTO ContactInfo VALUES ;
            (cFirst, cLast, cAddr, cPhone)
    ENDWITH
ENDFOR

* Показать результаты
BROWSE

RETURN

```



Скорее всего, вам придется пойти и в другом направлении и создать контакт Outlook на основе данных FoxPro. Хотя эта книга посвящена решениям FoxPro, имейте в виду, что Outlook поддерживает прямой импорт файлов во многих форматах и имеет довольно быстрый и гибкий интерфейс мастера импорта. Рассмотрите возможность экспорта контактной информации из VFP в один из распространенных форматов (например, CSV), а затем импортируйте ее оттуда в Outlook. Контакты создаются, как и другие элементы Outlook, с помощью метода CreateItem. Пример, показанный в листинге 2 (MakeContact.PRG в файлах для загрузки для разработчиков, доступных по адресу www.hentzenwerke.com) добавляет запись из таблицы поставщиков TasTrade в контакты Outlook.

Листинг 2. Создайте контакт Outlook. Эта программа отправляет данные из VFP в Outlook для создания нового контакта.

```

* MakeContact.PRG
* © 2000, Tamar E. Granor and Della Martin
* From: Microsoft Office Automation with Visual FoxPro
* Hentzenwerke Publishing. www.hentzenwerke.com

* Добавить информацию о поставщике.

#DEFINE olContactItem 2
#DEFINE olBusiness 2

LOCAL oNameSpace, oContact

IF VarType(oOutlook) <> "O"
    * Запустите или подключитесь к Outlook.
    * Сделайте его общедоступным для демонстрационных целей.
    RELEASE oOutlook
    PUBLIC oOutlook
    oOutlook = CreateObject("Outlook.Application")
ENDIF
oNameSpace = oOutlook.GetNameSpace("MAPI")

```



```

* Откройте таблицу "Поставщики"
OPEN DATA _SAMPLES + "TasTrade\Data\TasTrade"
USE Supplier

* Выберите случайную запись для демонстрационных целей
GO RAND() * RECCOUNT()

* Создать новую запись контакта
oContact = oOutlook.CreateItem( olContactItem )

WITH oContact
    .FullName           = Contact_Name
    .CompanyName        = Company_Name
    .BusinessAddressStreet = Address
    .BusinessAddressCity  = City
    .BusinessAddressState = Region
    .BusinessAddressPostalCode = Postal_Code
    .BusinessAddressCountry = Country
    .SelectedMailingAddress = olBusiness
    .BusinessTelephoneNumber = Phone
    .PrimaryTelephoneNumber = Phone
    .BusinessFaxNumber      = Fax
    .Save()
ENDWITH

USE

RETURN

```

12.6 Добавление встреч и задач

Помимо общения с внешним миром, вы можете использовать Outlook, чтобы держать пользователей в курсе событий. Хотя вы можете встроить календарь в свое приложение, зачем дублировать усилия? Почему бы просто не использовать тот, который уже есть на компьютере пользователя? В Outlook легко создавать новые встречи и читать уже существующие. Точно так же не встраивайте в свое приложение список дел. Позвольте вашим приложениям взаимодействовать со списком задач Outlook.

Как и в случае с другими элементами Outlook, добавлять встречи и задачи так же просто, как вызвать `CreateItem` и заполнить поля, а затем вызвать метод `Save` для нового объекта. В таблице 5 показаны ключевые свойства метода `AppointmentItem`. В таблице 6 рассматриваются свойства `TaskItem`.

Метод `MarkComplete TaskItem` указывает, что задача завершена на текущую дату. Он устанавливает также `PercentComplete` на 100 и `Complete` на `.T.` Мы не знаем, почему вы не можете передать ему параметр, указывающий, когда задача была выполнена.

Таблица 5. Назначение встреч. Это свойства, которые вы, скорее всего, зададите для объекта `AppointmentItem`.

Свойство	Тип	Описание
Start, End	DateTime	Время начала и окончания встречи соответственно.

Свойство	Тип	Описание
Duration	Numeric	Продолжительность встречи, в минутах. Разумеется, взаимодействует с Start и End. Вы не можете установить все три.
AllDayEvent	Logical	Указывает, является ли элемент «событием на весь день», то есть чем-то, что длится весь день, а не происходит в определенное время.
Subject	Character	Описание предмета. Тему можно использовать в качестве индекса коллекции Items.
Location	Character	Место назначения. В календаре местоположение отображается рядом с темой в скобках.
Body	Character	Примечания к назначению. Они вообще не отображаются в календаре, а только в самом диалоговом окне «Встреча».
ReminderSet	Logical	Указывает, должен ли Outlook отображать напоминание об этой встрече.
ReminderMinutesBeforeStart	Numeric	Количество минут до времени начала, когда Outlook должен предоставить напоминание о встрече.

Таблица 6. Описание задач. Эти свойства являются ключом к помещению TaskItem в список задач.

Свойство	Тип	Описание
Subject	Character	Описание задачи отображается в столбце «Тема» списка задач. Это свойство можно использовать в качестве индекса коллекции Items.
DueDate	DateTime	Дата, к которой задача должна быть выполнена.
Complete	Logical	Указывает, выполнена ли задача. Хотя это свойство не ReadOnly, для установки этого свойства лучше использовать метод MarkComplete.
DateCompleted	DateTime	Дата выполнения задачи. Обратите внимание, что это элемент даты и времени, но важна только часть даты. Как и в случае со свойством Complete, для установки этого свойства лучше использовать метод MarkComplete.
StartDate	DateTime	Дата запуска задачи.

Свойство	Тип	Описание
Status	Numeric	Текущий статус задачи. Используйте одну из этих констант: olTaskNotStarted 0 olTaskWaiting 3 olTaskInProgress 1 olTaskDeferred 4 olTaskComplete 2
PercentComplete	Numeric	Указывает, насколько далеко продвинулась задача.
Importance	Numeric	Приоритет задачи (соответствует полю Приоритет в диалоговом окне Задача). Использует эти константы: olImportanceLow 0 olImportanceNormal 1 olImportanceHigh 2
Categories	Character	Список категорий, разделенных запятыми или точками с запятой, в которые следует поместить задачу.
Body	Character	Примечания к заданию. Отображается только в диалоговом окне «Задача» и при выводе в виде заметки.
ReminderSet	Logical	Указывает, должен ли Outlook предоставлять напоминание об этой задаче.
ReminderTime	DateTime	Дата и время, когда Outlook должен предоставить напоминание об этой задаче.



Код в листинге 3 добавляет в Календарь дни рождения всех сотрудников TasTrade как события, рассчитанные на целый день, на следующие два года и создает задачу по отправке каждому из них поздравительной открытки. Срок выполнения задачи варьируется в зависимости от страны, чтобы обеспечить достаточно времени для отправки по почте. Вы найдете его как MakeBdays.PRG в файлах загрузки для разработчиков, доступных по адресу www.hentzenwerke.com.

Листинг 3. Добавление встреч и задач. Эта процедура добавляет дни рождения всех сотрудников TasTrade в календарь Outlook на следующие два года, а также создает задачу каждый раз отправлять каждому поздравительную открытку.

```
* MakeBDays.PRG
* © 2000, Tamar E. Granor and Della Martin
* From: Microsoft Office Automation with Visual FoxPro
* Hentzenwerke Publishing. www.hentzenwerke.com

* Добавьте в Календарь дни рождения сотрудников ТасТрейд
* и создайте задачу для каждого отправить поздравительную
* открытку.
#DEFINE olAppointmentItem 1
#DEFINE olTaskItem 3
```

```

#DEFINE CR CHR(13)

LOCAL oNameSpace, oAppt1, oAppt2, oTask1, oTask2
LOCAL dNextBirthday

IF VarType(oOutlook) <> "O"
    * Запустите или подключитесь к Outlook.
    * Сделайте его общедоступным для демонстрационных целей.
    RELEASE oOutlook
    PUBLIC oOutlook
    oOutlook = CreateObject("Outlook.Application")
ENDIF
oNameSpace = oOutlook.GetNameSpace("MAPI")

* Откройте таблицу "Сотрудники".
OPEN DATABASE _SAMPLES+"\\TasTrade\\Data\\TasTrade"
USE Employee

SCAN
    * Создайте одну встречу и одну задачу для каждого
    * сотрудника на каждый из двух лет.
    oAppt1 = oOutlook.CreateItem( olAppointmentItem )
    WITH oAppt1
        * Выясните, когда наступит следующий день рождения
        * сотрудника.
        dNextBirthday = GOMONTH( Birth_Date, ;
            (YEAR(DATE()) - YEAR(Birth_Date)) * 12)
        IF dNextBirthday <= DATE()
            dNextBirthday = GOMONTH( dNextBirthday, 12)
        ENDIF

        * Установите дату и сделайте ее событием на весь день.
        .Start = DTOT( dNextBirthday )
        .AllDayEvent = .T.

        * Используйте имя сотрудника в теме.
        .Subject = First_Name - ( " " + Last_Name ) - ;
            " День рождения"

        * Отключите напоминания для этого.
        .ReminderSet = .F.

        .Save()
    ENDWITH

    oTask1 = oOutlook.CreateItem( olTaskItem )
    WITH oTask1
        * Задайте имя задачи, используя имя сотрудника.
        .Subject = "Отправить поздравительную открытку " + ;
            First_Name - ( " " + Last_Name)

        * Определите срок сдачи в зависимости от страны,
        * в которую открытка направляется.

```

```

DO CASE
CASE INLIST(Country , "USA", "Canada")
    .DueDate = dNextBirthday - 3
CASE INLIST(Country, "UK", "France")
    .DueDate = dNextBirthday - 7
OTHERWISE
    * Подумайте, это займет много времени.
    .DueDate = dNextBirthday - 10
ENDCASE

* Для удобства укажите почтовый адрес в теле сообщения.
cAddress = Address + CR + ;
    City + CR + ;
    Region + CR + ;
    Postal_Code + CR + ;
    Country
.Body = "Почтовый адрес (неформатированный): " + CR + ;
    cAddress

* Установите напоминание на день раньше.
.ReminderTime = .DueDate - 1440*60
.ReminderSet = .T.

* Установите категорию.
.Categories = "Gifts"    && "Подарки"
.Save()
ENDWITH

* Теперь выполните второй год.
oAppt2 = oAppt1.Copy()
WITH oAppt2
    * Переместить дату на один год вперед.
    .Start = GOMONTH( .Start, 12 )
    .Save()
ENDWITH

oTask2 = oTask1.Copy()
WITH oTask2
    * Переместить дату на один год вперед.
    .DueDate = GOMONTH( .DueDate, 12 )
    .Save()
ENDWITH

ENDSCAN

USE IN Employee
CLOSE DATA

RETURN

```



На случай, если вы, как и мы, на самом деле не хотите оставлять эти элементы разбросанными по всей вашей копии Outlook, листинг 4 – это программа-

противоядие. Она включена как CleanBdays.PRG в файлы загрузки для разработчиков, доступные по адресу www.hentzenwerke.com. Она использует метод Find коллекции Items. Функция Find принимает единственный параметр – строку фильтра и возвращает первый элемент, соответствующий этому фильтру. Требуемый формат для фильтра нас очень радует: в FoxPro есть три набора символов-разделителей: свойства элемента должны быть заключены в квадратные скобки. Если соответствующий элемент не найден, Find возвращает .null.

Стоит отметить, что существует также метод FindNext. В этой программе это не требуется, потому что каждый раз, когда мы находим соответствующий элемент, мы немедленно удаляем его. Итак, следующий поиск снова будет искать первый элемент, соответствующий строке фильтра. (Другой подход к этой проблеме можно найти в методе Restrict в файле справки. Это похоже на установку фильтра в списке элементов.)

Листинг 4. Поиск и удаление событий. Эта программа выполняет очистку после программы в листинге 3.

```
* CleanBdays.PRG
* © 2000, Tamar E. Granor and Della Martin
* From: Microsoft Office Automation with Visual FoxPro
* Hentzenwerke Publishing. www.hentzenwerke.com

* Очистка записей о днях рождения, добавленных в Outlook.
#DEFINE olFolderCalendar 9
#DEFINE olFolderTasks 13

LOCAL oNameSpace, oAppt, oTask
LOCAL oAppts, oTasks, cFilter

IF VarType(oOutlook) <> "O"
    * Запустите или подключитесь к Outlook.
    * Сделайте его общедоступным для демонстрационных целей.
    RELEASE oOutlook
    PUBLIC oOutlook
    oOutlook = CreateObject("Outlook.Application")
ENDIF
oNameSpace = oOutlook.GetNameSpace("MAPI")

oAppts = oNameSpace.GetDefaultFolder( olFolderCalendar )
oTasks = oNameSpace.GetDefaultFolder( olFolderTasks )

* Откройте таблицу «Сотрудники».
OPEN DATABASE _SAMPLES+"\\TasTrade\\Data\\TasTrade"
USE Employee

SCAN
    * Создать строку фильтра.
    cFilter = '[Subject] = "' + First_Name - ;
              (" " + Last_Name ) - " День рождения" + "'

    * Цикл, пока не останется совпадений.
    oAppt = oAppts.Items.Find( cFilter )
    DO WHILE NOT IsNull( oAppt )
        oAppt.Delete()
```

```

oAppt = oAppts.Items.Find( cFilter )
ENDDO

* Теперь сделайте то же самое для задач.
cFilter = '[Subject] = "Отправить поздравительную открытку ' ;
        + First_Name - ( " " + Last_Name) + "'"
* Цикл, пока совпадений больше не останется.
oTask = oTasks.Items.Find( cFilter )
DO WHILE NOT IsNull( oTask )
    oTask.Delete()
    oTask = oTasks.Items.Find( cFilter )
ENDDO

```

ENDSCAN

RETURN

12.7 Собираем все вместе



Листинг 5 показывает программу, которая добавляет встречу в календарь, добавляет несколько связанных задач, а затем отправляет электронное письмо другим участникам встречи, в данном случае всем сотрудникам ТасТрейд. Программа ожидает, что они будут в адресной книге Outlook, однако создает окно сообщения со списком всех, кому не удалось отправить сообщение. (Как и в другом примере электронной почты в этой главе, вам необходимо заменить людей, которые на самом деле находятся в вашей адресной книге Outlook, чтобы этот пример работал правильно.) На рисунке 3 показана встреча, которую устанавливает программа, а на рисунке 4 показана одна из результирующих задач. Листинг 5 вы найдете как OutlookSample.PRG в файлах для загрузки для разработчиков, доступных на сайте www.hentzenwerke.com.

Листинг 5. Запланируйте встречу. Эта программа настраивает встречу, добавляя ее в календарь, добавляя несколько задач и отправляя электронное письмо всем сотрудникам.

```

* OutlookSample.PRG
* © 2000, Tamar E. Granor and Della Martin
* From: Microsoft Office Automation with Visual FoxPro
* Hentzenwerke Publishing. www.hentzenwerke.com

* Это пример «Собираем все вместе» из главы 12.
* Добавьте встречу, несколько задач и отправьте соответствующее
* электронное письмо всем сотрудникам ТасТрейд.

#DEFINE olMailItem          0
#DEFINE olAppointmentItem  1
#DEFINE olTaskItem          3
#DEFINE CR                  CHR(13)
#DEFINE olImportanceHigh    2
#DEFINE MB_ICONINFORMATION  64      && Информационное сообщение
#DEFINE MB_OK                0      && Только кнопка ОК
#DEFINE OneDayInSeconds     86400

```



```

LOCAL oNameSpace, oMessage, oAppt, oTask

IF VarType(oOutlook) <> "O"
    * Запустите или подключитесь к Outlook.
    * Сделайте его общедоступным для демонстрационных целей.
    RELEASE oOutlook
    PUBLIC oOutlook
    oOutlook = CreateObject("Outlook.Application")
ENDIF

oNameSpace = oOutlook.GetNameSpace("MAPI")

* Сначала назначьте встречу.
oAppt = oOutlook.CreateItem( olAppointmentItem )
WITH oAppt
    .Subject      = "Ежемесячное собрание сотрудников"
    .Location     = "Конференц-зал А"
    .Start        = {^ 2000/02/01 9:00}
    .Duration     = 90
    .ReminderSet  = .T.
    .ReminderMinutesBeforeStart = 15
    .Save()
ENDWITH

* Теперь настройте некоторые связанные задачи.
* Во-первых, повестка дня.
oTask = oOutlook.CreateItem( olTaskItem )
WITH oTask
    .Subject      = "Повестка дня собрания сотрудников"
    .DueDate      = {^ 2000/01/31 12:00}
    .Categories   = "Совещание персонала"
    * Получить напоминание за день до наступления срока.
    .ReminderSet  = .T.
    .ReminderTime = .DueDate - OneDayInSeconds
    .Save()
ENDWITH

* Далее закуски
oTask = oOutlook.CreateItem( olTaskItem )
WITH oTask
    .Subject      = "Заказать прохладительные напитки"
    .DueDate      = {^ 2000/01/30 17:00}
    .Categories   = "Совещание персонала"
    * Получить напоминание за полчаса
    .ReminderSet  = .T.
    .ReminderTime = .DueDate - 30*60
    * Разместите в задаче информацию о том, что и где заказать.
    .Body = "Приобретите бублики и датскую еду в кошерной пекарне."
    .Save()
ENDWITH

* Наконец, разошлите уведомление всем сотрудникам.
OPEN DATA _SAMPLES + "TasTrade\Data\TasTrade"

```

```
USE Employee
```

```
* Создайте курсор, чтобы отслеживать сотрудников,  
* которым мы не смогли отправить письмо.
```

```
CREATE CURSOR EmailProbs (cFullName C(40))
```

```
oMessage = oOutlook.CreateItem( olMailItem )
```

```
WITH oMessage
```

```
    .Subject = "Ежемесячное собрание сотрудников"
```

```
    .Body = "Ежемесячное собрание сотрудников будет проводиться "+;
```

```
        "во вторник, 1 февраля, 9:00 " + ;
```

```
        "в конференц-зале А. Ожидается, что все будут " + ;
```

```
        "присутствовать." + ;
```

```
    + CR + CR + ;
```

```
    "Принесите, пожалуйста, отчеты о продажах за декабрь и " + ;
```

```
    "предварительные данные за январь." + ;
```

```
    + CR + CR + ;
```

```
    "Закуски будут поданы как обычно."
```

```
    .Importance = olImportanceHigh
```

```
* Теперь пройдите по сотрудникам, добавив получателей.
```

```
SELECT Employee
```

```
SCAN
```

```
    oRecipient = .Recipients.Add(First_Name - (" " + Last_Name))
```

```
    IF NOT oRecipient.Resolve()
```

```
        * Либо это имя неоднозначно, либо в файле
```

```
        * нет адреса электронной почты этого человека.
```

```
        * Зарегистрируйте упущение
```

```
        INSERT INTO EmailProbs VALUES (oRecipient.Name)
```

```
        * Удалить этого человека.
```

```
        oRecipient.Delete()
```

```
    ENDIF
```

```
ENDSCAN
```

```
IF .Recipients.Count > 0
```

```
    * Отправьте это.
```

```
    .Send()
```

```
ELSE
```

```
    * Получателей нет, поэтому избавьтесь от сообщения.
```

```
    .Delete()
```

```
ENDIF
```

```
ENDWITH
```

```
* Сообщить о неудавшихся электронных письмах.
```

```
SELECT EmailProbs
```

```
IF RECCOUNT() = 0
```

```
    MESSAGEBOX( "Все электронные письма отправлены", ;
```

```
        MB_ICONINFORMATION+MB_OK, ;
```

```
        "Пример Автоматизации Outlook")
```

```
ELSE
```

```

* Создайте строку, содержащую список ошибок.
cFailString = ""
SCAN
    cFailString = cFailString + cFullName + CR
ENDSCAN
MESSAGEBOX("Электронное письмо не было отправлено " + ;
    "следующим сотрудникам: " + ;
    CR + cFailString, ;
    MB_ICONINFORMATION+MB_OK, ;
    "Пример Автоматизации Outlook")
ENDIF

USE IN EmailProbs
USE IN Employee

RETURN

```

Outlook – это многофункциональная программа для управления личными данными, такими как расписания, списки дел, адресные книги и электронная почта. Хотя его объектная модель не полиморфна остальной части Office, она предоставляет множество возможностей для Автоматизации.

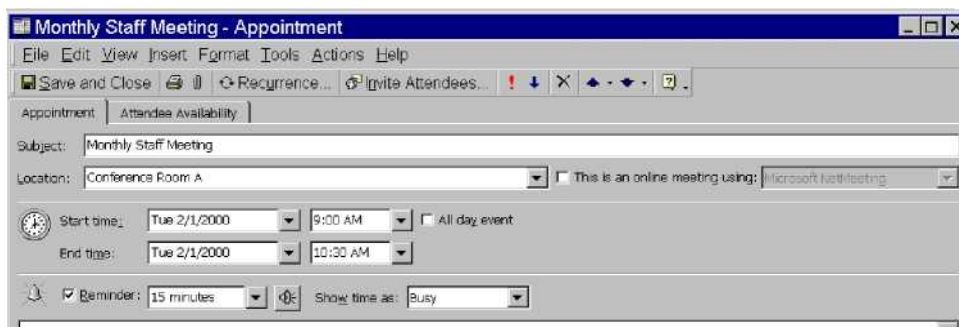


Рисунок 3. Автоматизированное планирование. Эта встреча была добавлена в календарь программой из листинга 5.

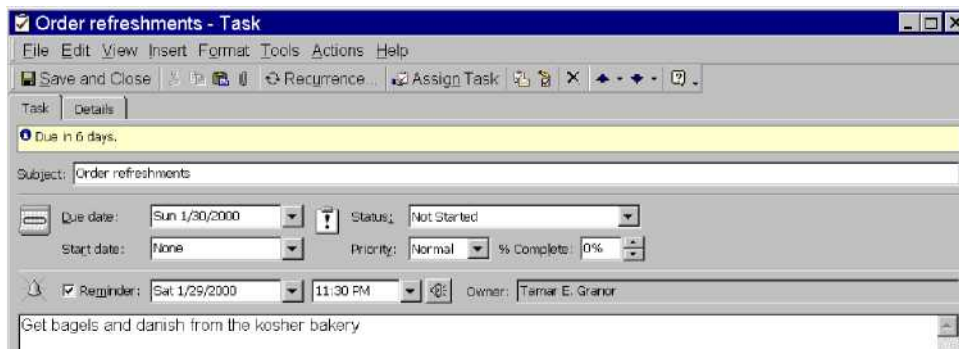


Рисунок 4. Скажи мне что делать. Это одна из двух задач, созданных в листинге 5.

Глава 13. Межофисная связь

После того как вы разобрались с серверами как с отдельными приложениями, следующим естественным шагом будет заставить их взаимодействовать друг с другом или с вашим приложением FoxPro.

Пройдет немного времени, прежде чем вы, ваши клиенты или ваш начальник придет к мысли, что вы должны иметь возможность вырезать и вставлять между классными приложениями, которые вы разработали. В конце концов, интерактивный пользователь может нажимать Ctrl-C и Ctrl-V для копирования и вставки между различными приложениями Office. Насколько сложно это автоматизировать? К счастью, это совсем не сложно. Мы рассмотрим несколько способов заставить данные с одного сервера появиться на другом сервере.

Следующая идея, которая кажется очевидной разработчику, заключается в том, что серверы Office прекрасно объектно-ориентированы и обладают полиморфными свойствами, методами и событиями. События VFP позволяет запускать код при определенных событиях; возможно, было бы целесообразно сделать то же самое на серверах Office. Разве не было бы неплохо, если бы сервер Office сообщал вашему приложению VFP, когда пользователь пытается сохранить, распечатать или закрыть документ, который ваше приложение так тщательно открыло? Это вполне возможно с помощью утилиты VFPCOM.DLL, которая связывает события сервера с процедурами VFP.

13.1 Взаимодействие между приложениями Office

Существует два основных метода Автоматизации обмена данными между двумя приложениями Office: один – вырезать выделенный фрагмент в буфер обмена и вставить его в другое приложение. Это хороший и простой метод. Второй метод – добавить его как объект OLE, что затем дает пользователю широкие возможности контроля над конечным продуктом, поскольку функции редактирования на месте OLE затем можно использовать для настройки внешнего вида, если это необходимо. Для добавления объекта OLE доступно несколько методов.

Копирование и вставка между приложениями

Интерактивный метод копирования между приложениями выглядит примерно так: откройте приложение с исходными данными, выберите исходные данные и с помощью «Вырезать» или «Копировать» вставьте их в буфер обмена. Затем откройте целевое приложение, найдите место для вставки данных и вставьте содержимое буфера обмена. Тот же процесс применим и к автоматическому копированию и вставке.

Чтобы проиллюстрировать Автоматизацию процедуры копирования и вставки, мы создадим небольшую электронную таблицу и поместим ее в документ Word.

Настройка исходных данных



В этом примере источником является небольшая электронная таблица. Главы 7, 8 и 9 («Основы Excel», «Расширенные функции Excel» и «Графический механизм Excel» соответственно) объясняют детали настройки электронной таблицы, поэтому мы не будем вдаваться в подробности здесь. В листинге 1 показан код для создания образца электронной таблицы, показанной на рисунке 1. Листинг 1 доступен как SourceSetup.PRG в файлах для загрузки для разработчиков, доступных по адресу www.hentzenwerke.com.

Листинг 1. Образец электронной таблицы Excel, которую мы будем использовать в наших примерах копирования. Результаты см. на рисунке 1.

```
* SourceSetup.prg
* © 2000, Tamar E. Granor and Della Martin
* From: Microsoft Office Automation with Visual FoxPro
* Hentzenwerke Publishing. www.hentzenwerke.com

* Удалите все существующие ссылки на серверы.
* Это предотвращает потерю памяти оставшимися экземплярами.
RELEASE ALL LIKE o*

* Сделайте следующие переменные общедоступными в демонстрационных
* целях, чтобы вы могли проверить объекты после завершения
* программы. oSourceRange должен быть общедоступным,
* если вы следуете примеру из книги.
PUBLIC oExcel, oSourceRange

* Создать экземпляр Excel.
oExcel = CreateObject("Excel.Application")

* Сделайте Excel видимым.
oExcel.Visible = .T.

oBook = oExcel.Workbooks.Add()
WITH oBook.Sheets[1]
    .Range("A1").Value = "Квартал"
    .Range("A2").Value = "Первый"
    .Range("A3").Value = "Второй"
    .Range("A4").Value = "Третий"
    .Range("A5").Value = "Четвертый"

    .Range("B1").Value = "Объем продаж"
    .Range("B2").Value = "100000"
    .Range("B3").Value = "125000"
    .Range("B4").Value = "150000"
    .Range("B5").Value = "175000"

    oSourceRange = .Range("A1:B5")
ENDWITH
```

	A	B	C
1	Quarter	Sales Volume	
2	First	100000	
3	Second	125000	
4	Third	150000	
5	Fourth	175000	

Рисунок 1. Пример электронной таблицы для копирования в документ Word. Хотя это простая электронная таблица, можно использовать любые функциональные возможности Excel. В главах 7, 8 и 9 подробно рассматривается Excel (см. «Основы Excel», «Расширенные функции Excel» и «Графический механизм Excel» соответственно).

Обратите внимание, что предпоследняя строка кода устанавливает переменную `oSourceRange`. Чтобы немного облегчить жизнь, стоит настроить переменную, содержащую данные, которые вы хотите скопировать, пока у вас еще открыты конструкции `WITH ... ENDWITH`. Мы сделали её `PUBLIC`, чтобы её можно было использовать вместе с остальным примером. Обычно `oSourceRange` имеет более подходящую область действия.

Для Excel в этом примере показано, что объект исходных данных представляет собой диапазон. Это также может быть целый рабочий лист или объект диаграммы. Для Word исходными данными, скорее всего, будет диапазон текста. Для PowerPoint исходными данными, скорее всего, будет один слайд. Имейте в виду, что вы можете использовать всю многостраничную книгу, презентацию из нескольких слайдов или многостраничный документ в качестве источника, но при копировании в целевое приложение он может не отображаться или распечатываться не так, как вы ожидаете, поскольку целевые приложения обычно показывают только один лист, слайд или страницу.

Скопируйте данные в буфер обмена

Получив ссылку на объект, используйте метод `Copy`, чтобы поместить ее в буфер обмена. В Excel объекты `Range`, `Chart` и `Worksheet` имеют метод копирования. Использование этого метода без параметров копирует объект в буфер обмена (доступны параметры, позволяющие копировать объект в другие места в Excel). Введите следующую команду, чтобы поместить исходные данные в буфер обмена:

```
oSourceRange.Copy()
```

Некоторые объекты Word, включая объект `Range`, имеют метод `Copy`, который используется без параметров для копирования в буфер обмена. Аналогично, метод `Copy` доступен для ряда объектов PowerPoint, включая объекты `Slide`, `Shape` и `TextRange`.

Подготовка целевого приложения



Подготовка целевого приложения состоит из открытия приложения и размещения точки вставки в нужном месте. Это описано в главах «Основы» для каждого из серверов Office. Чтобы проиллюстрировать эту концепцию, в примере открывается пустой документ и добавляется строка текста. Курсор устанавливается в точку ввода таблицы. На рисунке 2 показаны результаты листинга 2, который хранится как `TargetSetup.PRG` в файлах загрузки для разработчиков, доступных по адресу www.hentzenwerke.com.



Если вы работаете с нами в интерактивном режиме, обязательно используйте `TargetSetup.PRG` или введите следующие строки в командном окне. Не копируйте и не вставляйте их из электронной версии книги, так как вы перезапишете таблицу Excel в буфере обмена. Затем Word аккуратно помещает этот код в документ, а не в электронную таблицу.

К счастью, это происходит только при интерактивной работе; как правило, никто не может изменить буфер обмена во время выполнения кода Автоматизации. Просто будьте осторожны, работая в командном окне и разрабатывая код, использующий буфер обмена, чтобы не создавать проблем, не связанных с вашим кодом.

Листинг 2. Пример документа Word, в который можно поместить электронную таблицу из листинга 1. Результаты см. на рисунке 2.

```
* TargetSetup.prg
* © 2000, Tamar E. Granor and Della Martin
* From: Microsoft Office Automation with Visual FoxPro
* Hentzenwerke Publishing. www.hentzenwerke.com

#DEFINE wdCollapseEnd 0

* Сделайте oRange и oWord общедоступными, чтобы их можно
* было использовать во время упражнения, следуя по тексту.
RELEASE oWord, oRange
PUBLIC oWord, oRange

* Создайте экземпляр Word.
oWord = CreateObject("Word.Application")

* Сделайте Word видимым.
oWord.Visible = .T.
oDocument = oWord.Documents.Add()

oRange = oDocument.Range()
oRange.InsertAfter("Это пример копирования части " + ;
                  "из электронной таблицы в Word." + CHR(13))
oRange.Collapse(wdCollapseEnd)
```

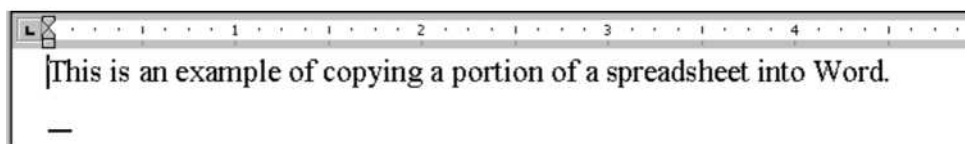


Рисунок 2. Образец документа Word, в который можно получить электронную таблицу. Это простой документ Word, но он может быть настолько сложным, насколько вы захотите. В главах 4, 5 и 6 («Основы Word», «Средний уровень Word» и «Продвинутый уровень Word» соответственно) сервер Word подробно описан.

Поскольку после вставки текста диапазон сворачивается до конца, oRange (объявленный в этом интерактивном примере PUBLIC) позиционируется правильно. Не обращайте внимания на фактическое положение курсора на рисунке 2. Диапазоны не зависят от положения курсора. Будьте уверены, что oRange расположен правильно для вставки таблицы.

Вставка данных из буфера обмена

В Word объект Range описывает позицию в документе, поэтому имеет смысл иметь метод Paste, который вставляет содержимое буфера обмена в Range (поэтому обязательно сверните свой диапазон, если вы не хотите, чтобы содержимое Range заменены). Синтаксис простой:

```
oRange.Paste()
```


Рисунок 3 показывает результирующую таблицу в документе Word. Электронная таблица Excel преобразуется в объект «Таблица Word».

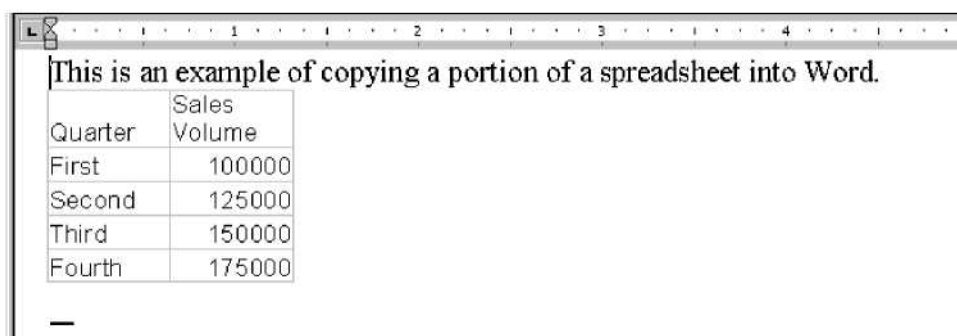


Рисунок 3. Таблица вставлена в документ. Обратите внимание, что электронная таблица преобразуется в таблицу.

Word удобен и прост: объект Range имеет команду Paste, которая не принимает никаких параметров. Excel немного другой. Команда Paste Worksheet принимает один из двух параметров: первый – это объект Range, в котором исходные данные помещаются на лист, а второй используется, если первый не указан, и является логическим параметром, указывающим, следует ли ссылаться на исходные данные (по умолчанию установлено значение False).

PowerPoint немного отличается. Во-первых, вы должны убедиться, что находитесь в обычном режиме, а не в режимах Slide Sorter, Notes или Slideshow views. Затем вы можете использовать метод Paste объекта коллекции Shapes или объекта TextRange (только для текстовых строк). Коллекция Shapes подходит для вставки объектов из других приложений, например:

```
oNewObj = oSlide.Shapes.Paste()
```

При этом электронная таблица (или другой объект) вставляется в середину указанного слайда. Управляйте свойствами Top и Left вновь размещенного объекта oNewObj, чтобы поместить его туда, где он должен быть. Интересно отметить, что диапазон слов, вставленный в коллекцию Shapes, становится одним текстовым полем. Однако когда диапазон из Excel вставляется в коллекцию Shapes, он помещается как внедренный объект Excel. Обычно для вставки встроенного или связанного объекта требуются другие команды, такие как PasteSpecial или AddOLEObject.

PasteSpecial

Метод Paste обычно помещает преобразованную копию исходных данных. Исходные данные преобразуются в наиболее похожий тип объекта в целевом приложении. После преобразования данные теряют возможности редактирования исходного приложения. Хотя в некоторых случаях преобразование таблицы Excel в таблицу Word является преимуществом, в других случаях могут потребоваться возможности редактирования исходного приложения.

Здесь на помощь приходит метод PasteSpecial. Возможно, вы использовали его в интерактивном режиме, щелкая Правка|Специальная вставка в любом меню приложений Office. При этом открывается диалоговое окно, показанное на рисунке 4, в котором отображается источник данных, определяющий не только имя приложения, но и диапазон (в обозначениях приложения). Кнопки выбора слева указывают, как вставить ссылку, а в поле списка указаны форматы, в которые можно преобразовать исходные данные.

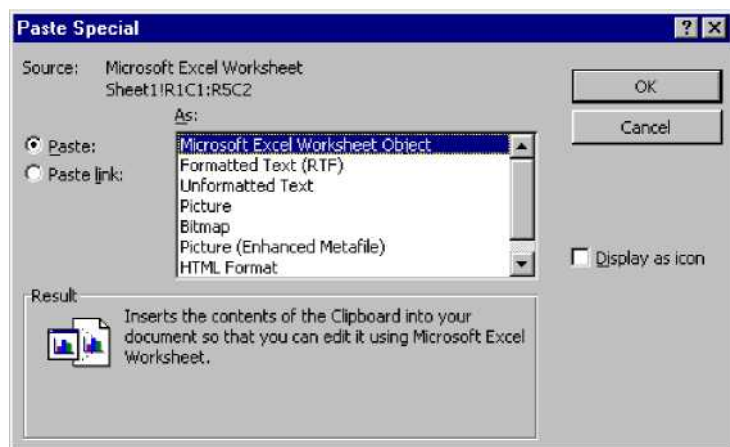


Рисунок 4. Диалоговое окно «Специальная вставка», полученное в интерактивном режиме из Word. Здесь показано множество опций, доступных при использовании метода PasteSpecial.

Метод PasteSpecial Word доступен из объекта Range. Он принимает ряд параметров, аналогичных параметрам, показанным на рисунке 4:

```
oRange.PasteSpecial( [nIconIndex], [lLink], [nPlacement],  
                    [lDisplayAsIcon], [nDataType],  
                    [cIconFileName], [cIconLabel] )
```

nIconIndex	Numeric	Если lDisplayAsIcon имеет значение .T., то указывает значок, отображаемый для этого объекта, вместо отображения самого объекта (если lDisplayAsIcon имеет значение .F., значение по умолчанию). Число соответствует положению значка в имени файла значка, указанном cIconFileName. Если этот параметр опущен, по умолчанию используется первый значок в указанном файле.			
lLink	Logical	Указывает, следует ли создать ссылку на исходный файл (.T.) или встроить его (.F.). По умолчанию — .F. Дополнительную информацию о связывании и встраивании см. в тексте. Этот параметр соответствует кнопкам выбора «Вставить» и «Вставить ссылку» в диалоговом окне, показанном на рисунке 4.			
nPlacement	Numeric	Определяет, как разместить объект относительно существующего текста. Используйте одну из следующих констант: <table> <tr> <td>wdFloatOverText</td> <td>1</td> <td>Объект плавает над текстом. Объект можно отформатировать так, чтобы обтекание текста происходило справа, слева, с обеих сторон объекта или останавливалось вверху и возобновлялось внизу. Объект также можно разместить в любом месте страницы.</td> </tr> </table>	wdFloatOverText	1	Объект плавает над текстом. Объект можно отформатировать так, чтобы обтекание текста происходило справа, слева, с обеих сторон объекта или останавливалось вверху и возобновлялось внизу. Объект также можно разместить в любом месте страницы.
wdFloatOverText	1	Объект плавает над текстом. Объект можно отформатировать так, чтобы обтекание текста происходило справа, слева, с обеих сторон объекта или останавливалось вверху и возобновлялось внизу. Объект также можно разместить в любом месте страницы.			

		wdInLine	0	Объект позиционируется так, как если бы он был символом в тексте. (По умолчанию)
IDisplayAsIcon	Logical	Указывает, следует ли отображать это в виде значка, требующего от пользователя щелчка по значку, чтобы открыть объект. Если установлено значение True, объект отображает значок, выбранный параметром nIconIndex в clconFileName, и отображает текстовую метку (переданную в clconLabel) под значком. Если установлено значение False, объект отображается в своем собственном формате (например, он выглядит как электронная таблица или слайд). Это соответствует флажку «Отображать как значок» в диалоговом окне, показанном на рисунке 4. Значение по умолчанию – False.		
nDataType	Numeric	Формат вставки объекта. Это соответствует полю со списком «Как»: в диалоговом окне, показанном на рисунке 4. Некоторые из допустимых значений:		
		wdPasteOLEObject	0	Сохраняет ссылку на исходное приложение (и, возможно, на исходный файл, если lLink имеет значение .T.).
		wdPasteHTML	10	Преобразуется в HTML.
		wdPasteBitmap	4	Преобразует исходные данные в простое растровое изображение.
		wdPasteText	2	Преобразует исходные данные в текст (аналогично методу Paste).
		Значение по умолчанию варьируется в зависимости от того, что содержится в буфере обмена..		
clconFileName	Character	Полное имя файла, содержащее отображаемый значок, если IDisplayAsIcon имеет значение .T.		
clconLabel	Character	Текст, отображаемый под значком, если IDisplayAsIcon имеет значение .T.		

Наиболее важным параметром является параметр lLink, который определяет, должен ли быть связан объект исходных данных или нет. Связанный объект по сути является указателем на исходный файл. Любые изменения, внесенные в исходный файл, отображаются в документе (при обновлении объекта), а любые изменения, внесенные в объект в документе, могут быть сохранены непосредственно в исходном файле (при условии, что пользователь сохраняет изменения). Если объект не связан, он считается «встроенным», что означает, что объект редактируется на месте в исходном виде, как если бы он был связан, но при этом нет связи с исходным исходным файлом. Объект полностью инкапсулирован в документ Word.

Чтобы проиллюстрировать метод PasteSpecial для внедрения объекта электронной таблицы Excel, мы будем использовать те же файлы примеров, что и ранее. Единственное изменение – переход от вызова метода Paste к вызову метода PasteSpecial. На рисунке 5 показаны результаты.

```
DO SourceSetup.PRG
oSourceRange.Copy()
DO TargetSetup.PRG
```

```
#DEFINE wdInLine          0
#DEFINE wdPasteOLEObject 0
```

```
oRange.PasteSpecial( ,.F., wdInLine, , wdPasteOLEObject)
```

Объект Excel добавляется как объект Shape в коллекцию Shapes. Обратите внимание, что на рисунке 5 метка «Объем продаж» обрезана. Если ширина столбца B была отрегулирована до обрезки или если диапазон включал пустой столбец C, это будет отображаться правильно. По умолчанию отображается именно указанный диапазон, не больше и не меньше.

Пользователь может редактировать любую часть электронной таблицы, дважды щелкнув только что добавленный объект Shape. Дисплей изменится на показанный на рисунке 6. Пользователь может внести изменения, щелкнув таблицу в любом месте документа.

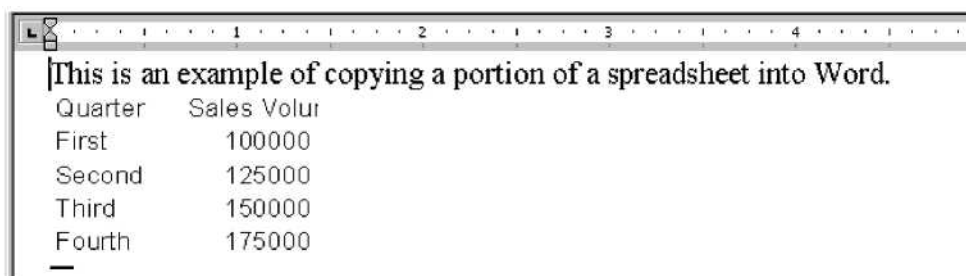


Рисунок 5. Пример использования PasteSpecial для связи диапазона Excel с Word. Обратите внимание, что метка «Объем продаж» обрезана. Небольшая работа над форматированием ширины столбца перед вырезанием в буфер обмена могла бы решить проблему.

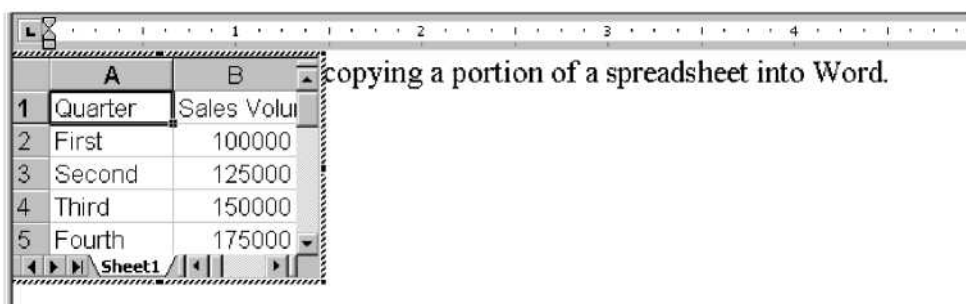


Рисунок 6. Дважды щелкните объект, добавленный с помощью PasteSpecial. Меню изменится и включит в себя меню Excel.

Когда пользователь завершает редактирование (о чем свидетельствует щелчок по электронной таблице), отображение возвращается к рисунку 5, а также отражаются все изменения.

Если вы связываете объект, исходный объект необходимо сохранить в виде файла. При явном сохранении исходного документа перед копированием его в буфер обмена объект в буфере обмена сообщит об этом приложению, выполняющему

вставку, а объект буфера обмена может предоставить исходное имя файла. Если объект не сохранен, ошибки не возникнет, если параметр `ILink` метода `PasteSpecial` равен `.T.`, но при этом он не будет связан.

А как насчет команды `PasteSpecial` в Excel? Ну, это похоже, но другое. Он использует большинство тех же параметров, но они организованы более последовательно, поскольку все параметры значков сгруппированы вместе:

```
oExcel.ActiveSheet.PasteSpecial( [cFormat], [lLink],  
                                  [lDisplayAsIcon], [cIconFileName],  
                                  [nIconIndex], [cIconLabel] )
```

Параметр `cFormat` (который соответствует параметру `nDataType` для метода Word) представляет собой существенное изменение синтаксиса Word. Он ожидает строку, идентифицирующую формат данных в буфере обмена, например «Объект документа Microsoft Word 8.0». К счастью, это необходимо только в том случае, если вы хотите изменить формат объекта на какой-то другой, отличный от того, какой он уже есть, например, изменить его на растровый или любой другой поддерживаемый формат. По умолчанию `ILink` имеет значение `False`, что означает встраивание объекта. Остальные параметры относятся к значку и работают так же, как в Word.

А как насчет команды `PasteSpecial` в PowerPoint? Удивительно, но его нет. Поскольку метод `Paste` добавляет текст Word как объект `Shape` с текстом и в любом случае добавляет электронную таблицу Excel как встроенный объект, мы предполагаем, что Microsoft посчитала, что в этом нет необходимости. Но как добавить связанный объект в PowerPoint? Существует альтернативный метод `PasteSpecial`, который не требует использования буфера обмена. Это метод `AddOLEObject`, доступный на всех серверах Office.

Использование `AddOLEObject`

Зачем вам использовать `AddOLEObject`, если доступен `PasteSpecial`? Есть несколько причин. Во-первых, если вы используете PowerPoint, это единственный способ связать объект (это довольно веская причина). Во-вторых, `AddOLEObject` обращается к файлу, который не обязательно открывать в коде (это может быть даже файл, созданный пользователем). Вам не нужно выбирать объект и использовать буфер обмена для его передачи.

Здесь мы собираемся использовать другой пример. Поскольку в PowerPoint нет метода `PasteSpecial` для связи с файлом, мы будем использовать PowerPoint в качестве целевого приложения. Мы будем использовать диаграмму Excel в качестве исходных данных. Диаграммы представляют собой инструменты визуального анализа и очень хорошо подходят для презентаций. Понятно, что кто-то захочет придумать автоматизированный метод создания слайдов PowerPoint с диаграммами, отражающими самые актуальные данные.

Метод PowerPoint `AddOLEObject` доступен только для объекта коллекции `Shapes`. Его синтаксис:

```
oSlide.Shapes.AddOLEObject([nLeft], [nTop], [nWidth], [nHeight],  
                           [cClassName], [cFileName],  
                           [lDisplayAsIcon], [cIconFileName],  
                           [nIconIndex], [cIconLabel], [lLink] )
```

Первые два параметра – это положение левого и верхнего краев объекта. Они указаны в пунктах, значение по умолчанию – 0. Следующие два параметра определяют ширину и высоту объекта в пунктах. К счастью, в PowerPoint 2000 они не являются обязательными (в PowerPoint 97 они обязательны).

Вы передаете либо cClassName, либо cFileName, оставляя другое поле пустым. Если вы не создаете пустые объекты, вам не нужно будет передавать имя класса или ProgID в качестве cClassName. Скорее всего, вы передадите имя файла, содержащего добавляемый объект.

Следующие четыре параметра (IDisplayAsIcon, clconFileName, nlconIndex и clconLabel) определяют, отображается ли объект в виде значка, и аналогичны параметрам PasteSpecial.

Последний параметр, lLink, определяет, является ли объект связанным или внедренным.



Листинг 3, хранящийся как AddOLEObject.PRG в файлах для загрузки для разработчиков, доступных на сайте www.hentzenwerke.com, демонстрирует создание и сохранение простой диаграммы в Excel, а затем открытие PowerPoint и связывание диаграммы.

Листинг 3. Использование AddOLEObject для связи диаграммы Excel с презентацией PowerPoint.

```
* AddOleObject.prg
* © 2000, Tamar E. Granor and Della Martin
* From: Microsoft Office Automation with Visual FoxPro
* Hentzenwerke Publishing. www.hentzenwerke.com

* Удалите все существующие ссылки на серверы.
* Это предотвращает потерю памяти оставшимися экземплярами.
RELEASE ALL LIKE o*

#DEFINE xlColumn          3
#DEFINE xlColumns         2
#DEFINE autoColumnFormat  4
#DEFINE autoOneSeriesLabel 1
#DEFINE autoOneCategoryLabel 1
#DEFINE autoHasLegend      .T.
#DEFINE autoIn2Pt          72
#DEFINE ppLayoutBlank      12

* Добавьте книгу, используя настройки по умолчанию.
oExcel = CreateObject("Excel.Application")
oExcel.Visible = .T.

oBook  = oExcel.Workbooks.Add()

* Добавьте простые данные в график
WITH oBook.Sheets[1]
    .Range("A2").Value = "Первый"
    .Range("A3").Value = "Второй"
    .Range("B1").Value = "Тест"
    .Range("B2").Value = "10"
    .Range("B3").Value = "20"

    oChartSource = .Range("A1:B3")
ENDWITH
```

```

* Создайте диаграмму в виде листа диаграммы
oChart = oExcel.Charts.ADD()
oChart.ACTIVATE()
oExcel.ActiveChart.ChartWizard(oChartSource, xlColumn, ;
    autoColumnFormat, xlColumns, autoOneCategoryLabel, ;
    autoOneSeriesLabel, autoHasLegend, ;
    "Пример диаграммы", "", "")

* Сохраните файл с активной таблицей данных.
*oExcel.Sheets["Sheet1"].Activate()
oExcel.Sheets["Диаграмма1"].Activate()
XLSFile = FULLPATH("SampleChart.XLS")
oExcel.ActiveWorkbook.SaveAs(XLSFile)
oExcel.Quit()

RELEASE oExcel, oBook, oChart, oChartSource

* Создайте переменные для демонстрационных целей,
* чтобы вы могли проверить их после завершения программы.
PUBLIC oPowerPoint, oOLEChart

* Создайте объект PowerPoint.
oPowerPoint= CreateObject("PowerPoint.Application")

* Сделайте его видимым и добавьте презентацию и слайды
oPowerPoint.Visible = .T.
oPresentation = oPowerPoint.Presentations.Add()
oSlide = oPresentation.Slides.Add(1, ppLayoutBlank)

* Добавить рабочий лист
oOLEChart = oSlide.Shapes.AddOLEObject( ;
    1.0 * autoIn2Pt, 1.5 * autoIn2Pt, ;
    8.0 * autoIn2Pt, 5.0 * autoIn2Pt, , ;
    XLSFile, , , , .T.)

```

С добавлением диаграмм Excel (и листов тоже) возникла небольшая проблема, поскольку в книге имеется несколько листов. Как PowerPoint узнает, какой лист отображать? Он отображает первый рабочий лист. Если в вашей книге настроено размещение диаграммы на листе, отличном от первого листа, или если ваша книга содержит несколько листов диаграмм, вы можете указать, какой лист будет отображаться, если объект связан (этот параметр недоступен, если объект встроен). Используйте свойство `SourceFullName` объекта `Shape LinkFormat`, чтобы указать имя листа. Задайте для этого свойства объединение имени файла, восклицательного знака и имени листа, как в «C:\My Files\SampleChart.XLS!Chart1». После установки этого свойства обновите ссылку, вызвав метод `Update` объекта `Shape LinkFormat`.

Методы `AddOLEObject` Excel и Word работают аналогично, за исключением того, что их список параметров находится в другом порядке.

```

oObj.AddOLEObject( [cClassName], [cFileName], [lLink],
    [lDisplayAsIcon], [cIconFileName],
    [nIconIndex], [cIconLabel], [nLeft], [nTop],
    [nWidth], [nHeight] )

```


oObj должен быть одним из следующих: объект коллекции Shapes листа Excel, объект коллекции Shape документа Word или объект коллекции InlineShapes документа Word.

Как и в PowerPoint, передайте либо cClassName, либо cFileName, а другое оставьте пустым. Следующий параметр – это логический параметр, определяющий, следует ли связать или внедрить объект (по умолчанию для внедрения объекта он имеет значение False). Далее следуют параметры значка. Последними являются параметры, определяющие местоположение нового объекта в пунктах. Если они не указаны, значения Left и Top по умолчанию равны 0, а высота и ширина по умолчанию соответствуют размерам объекта.

Добавление документа в Excel с использованием следующего синтаксиса:

```
oOLEDoc = oSheet.Shapes.AddOLEObject( , "C:\MyDocs\Test.DOC")
```

добавляет документ с координатой 0,0 относительно активной ячейки. Если выбрана ячейка B3, то новый объект размещается относительно ячейки B3. Размер объекта по умолчанию составляет 468 пунктов в ширину и 717,75 пунктов в высоту. В электронной таблице это занимает примерно 10 столбцов по 57 строк (с учетом ширины столбца и высоты строки по умолчанию). Размер рассчитывается как 6,5 на 9,9 дюйма, что соответствует размерам полей, хранящихся в документе. Если пользователь дважды щелкает объект, он активирует редактирование документа на месте.

Добавление презентации в Excel с использованием следующего синтаксиса:

```
oOLEPpt = oSheet.Shapes.AddOLEObject( , "C:\MyDocs\Test.PPT")
```

добавляет представление с координатой 0,0 относительно активной ячейки. Размер презентации PowerPoint по умолчанию составляет 360 пунктов в ширину и 270 пунктов в высоту (7 столбцов по 22 строки). Двойной щелчок по объекту запускает слайд-шоу.

Добавьте презентацию в Word в верхнем левом углу на полях выбранной в данный момент страницы с синтаксисом, подобным следующему:

```
oOLEPpt = oDoc.Shapes.AddOLEObject( , "C:\MyDocs\Test.PPT")
```

Это добавляет презентацию шириной 360 пунктов и высотой 270 пунктов и запускает слайд-шоу при двойном щелчке пользователем.

Добавьте книгу в Word в верхнем левом углу текущей страницы с помощью команды, аналогичной следующей:

```
oOLEXls = oDoc.Shapes.AddOLEObject( , "C:\MyDocs\Test.XLS")
```

По умолчанию на листе Excel отображается только первая ячейка листа, которая была активна при последнем сохранении файла, независимо от того, сколько ячеек используется на листе (хотя мы видели различия, которые могут быть связаны со свойствами настройки страницы, хранящимися как Excel по умолчанию). Ячейка по умолчанию имеет ширину 52,2 пункта и высоту 13,8 пункта, не очень большую. Однако если лист диаграммы был активным листом при сохранении файла, то диаграмма будет иметь колоссальные размеры 683 на 487 пунктов (9,5 на 6,7 дюйма), что слишком широко для страницы! Для листов Excel представляется разумным указать высоту и ширину и, возможно, заранее определить, что находится на листе (диаграмма или размеры ячеек на листе) перед добавлением объекта.

В Word при связывании объекта существует объект LinkFormat со свойством SourceFullName, но он не позволяет указать имя листа, как это делает PowerPoint.

Кажется, что Word отображает только первый слайд или лист, поэтому учтите это при разработке приложения.

Взаимодействие между приложениями Office, независимо от метода, может добавить к вашему приложению значительную функциональность. Подождите, пока не увидите, что мы можем делать с событиями.

13.2 Информирование о мероприятиях с помощью VFPCOM

На протяжении всей этой книги мы сосредоточились на том, чтобы указывать серверам, что им делать. Связь с сервером осуществляется в форме вызова методов и установки свойств. Связь с сервером принимает форму VFP, иницилирующую запрос свойства или получающую возвращаемое значение от вызовов методов. Общение кажется немного однобоким, и для начала всех разговоров требуется VFP. Это кажется странным по сравнению с элементами управления COM (ActiveX), которые могут обмениваться данными с VFP через код события.

Хотя FoxPro может видеть и обрабатывать события, вызванные элементами управления ActiveX, он не может видеть события, вызванные объектами, созданными с помощью функции `CreateObject()`. Это может привести к некоторым неловким заговоздкам, например, к тому, что VFP будет находиться в цикле в ожидании изменения статуса сервера или вызывать сервер и покидать экран VFP с сообщением: «Нажмите эту кнопку, когда закончите». Напротив, в Visual Basic есть конструкция WITH EVENTS, которая позволяет событиям на серверах, которые вызывает VB, запускать код внутри VB. Вскоре после выхода VFP 6.0 мастера из Microsoft разработали COM-элемент управления под названием VFPCOM, который решает эту проблему. VFPCOM связывает события сервера с созданным вами объектом обработчика событий, который имеет методы с теми же именами, что и события объекта сервера. Когда Word вызывает событие `DocumentBeforeClose`, VFPCOM ищет метод `DocumentBeforeClose` в созданном вами объекте обработчика событий.

Получение VFPCOM.DLL

Вам нужно будет получить VFPCOM с веб-сайта Microsoft по адресу msdn.microsoft.com/vfoxpro или <https://legacyupdate.net/download-center/download/2049/visual-foxpro-vfpcom-utility>. Перейдите по ссылкам на страницу «Обновления продуктов» (на момент написания статьи она находится в разделе «Образцы и загрузки» на главной странице). Затем выберите утилиту VFPCOM для загрузки. Это примерно 156 КБ. Сохраните файл VFPCOM.EXE на диск. Находясь на веб-сайте, обязательно сохраните или распечатайте файл ReadMe, который не включен в EXE-файл.

Когда вы запускаете VFPCOM.EXE для его установки, он помещает DLL и файлы ее примеров в каталог \VFPCOM под основным каталогом VFP (если вы не указали иное). Он также регистрирует DLL.

Использование VFPCOM.DLL

Первым шагом в использовании VFPCOM является создание ссылки на объект. Используйте имя класса в функции `CreateObject()`, например:

```
oVFPCOM = CreateObject("vfpcom.comutil")
```

Чтобы использовать VFPCOM, вам также потребуется ссылка на сервер Office. Для примеров мы будем использовать Word (если вы пытаетесь сделать это в интерактивном режиме, сделайте себе одолжение и сделайте его видимым, чтобы

не потерять невидимый экземпляр сервера):

```
oWord = CreateObject("Word.Application")  
oWord.Visible = .T.
```

Сервер VFPCOM имеет пять методов, но нам важны только два: ExportEvents и BindEvents. Метод ExportEvents дает ответ на вопрос: «Но как мне найти все имена событий для записи этого объекта обработчика событий?» ExportEvents не только извлекает имена событий, но и создает программный файл, содержащий полное определение класса! УХ ТЫ! Метод принимает два параметра: первый – это ссылка на объект, события которого вы хотите экспортировать, а второй – имя файла, в который все это записать. Предполагая, что вы создали oVFPCOM и oWord (как показано ранее), создайте файл WordEvents.PRG следующим образом:

```
oVFPCOM.ExportEvents(oWord, "WordEvents.PRG")
```

И затем, как по волшебству, файл WordEvents.PRG содержит следующее:

```
DEFINE CLASS ApplicationEvents2 AS custom  
  
    PROCEDURE DocumentBeforeClose(Doc, Cancel)  
        * Add user code here  
    ENDPROC  
  
    PROCEDURE DocumentBeforePrint(Doc, Cancel)  
        * Add user code here  
    ENDPROC  
  
    PROCEDURE DocumentBeforeSave(Doc, SaveAsUI, Cancel)  
        * Add user code here  
    ENDPROC  
  
    PROCEDURE DocumentChange  
        * Add user code here  
    ENDPROC  
  
    PROCEDURE DocumentOpen(Doc)  
        * Add user code here  
    ENDPROC  
  
    PROCEDURE NewDocument(Doc)  
        * Add user code here  
    ENDPROC  
  
    PROCEDURE Quit  
        * Add user code here  
    ENDPROC  
  
    PROCEDURE WindowActivate(Doc, Wn)  
        * Add user code here  
    ENDPROC  
  
    PROCEDURE WindowBeforeDoubleClick(Sel, Cancel)  
        * Add user code here  
    ENDPROC
```

```
PROCEDURE WindowBeforeRightClick(Sel, Cancel)
* Add user code here
ENDPROC
```

```
PROCEDURE WindowDeactivate(Doc, Wn)
* Add user code here
ENDPROC
```

```
PROCEDURE WindowSelectionChange(Sel)
* Add user code here
ENDPROC
```

```
ENDDEFINE
```

Круто, нет? Просто измените имя класса с ApplicationEvents2 на WordEvents, измените имена переменных параметров на свое собственное соглашение об именах, затем добавьте свой код, и вы получите определение класса.

Следующий шаг – использовать метод BindEvents, чтобы сообщить VFPCOM, какой объект является сервером и какой объект содержит методы, сопоставленные с событиями. Первый параметр – это ссылка на объект приложения Office; второй – это ссылка на объект (скелетом которого может быть результат ExportEvents).

```
oWordEvents = CreateObject("WordEvents")
oVFPCOM.BindEvents(oWord, oWordEvents)
```

Убедитесь, что вы определили область действия переменных для утилиты VFPCOM и объектов обработчика событий (oVFPCOM и oWordEvents), чтобы они были доступны, пока доступна ссылка на приложение Office.



Давайте рассмотрим небольшой пример, показывающий, как работает VFPCOM. В листинге 4 показана очень урезанная версия того, что необходимо для обработки событий Word. Листинг 4 доступен как VFPCOMandWord.PRG в файлах загрузки для разработчиков, доступных по адресу www.hentzenwerke.com.

Листинг 4. Простой пример того, как VFPCOM обрабатывает события Word. Этот интерактивный пример иллюстрирует, как VFPCOM обрабатывает событие Word перед закрытием.

```
* VFPCOMandWord.prg
* © 2000, Tamar E. Granor and Della Martin
* From: Microsoft Office Automation with Visual FoxPro
* Hentzenwerke Publishing. www.hentzenwerke.com

* Удалите все существующие ссылки на серверы.
* Это предотвращает потерю памяти оставшимися экземплярами.
RELEASE ALL LIKE o*

* Сделайте следующие переменные общедоступными в
* демонстрационных целях, чтобы они были доступны после
* запуска PRG для демонстрации функциональности VFPCOM.
PUBLIC oWord, oWordEvents, oVFPCOM

* Создайте серверное приложение.
oWord = CreateObject("Word.Application")
```

```

* Сделайте его видимым.
oWord.Visible = .T.
oWord.Documents.Add()
oWord.Documents[1].Range.Text = "Это тест."

* Создайте объект VFPCOM.
oVFPCom = CreateObject("vfpcom.comutil")

* Создайте обработчик событий.
oWordEvents = CreateObject("WordEvents")

* Привяжите сервер к обработчику событий.
oVFPCom.BindEvents(oWord, oWordEvents)

* Выведите VFP наверх и сообщите пользователю, что делать дальше.
_SCREEN.AlwaysOnTop = .T.
_SCREEN.AlwaysOnTop = .F.
MessageBox("Когда вы закроете это окно, вы перейдете в Word." + ;
    CHR(13) + ;
    "Закройте Word, чтобы посмотреть, как VFPCOM с этим справится";
    + CHR(13) + ;
    "VFP должен выдать MessageBox 'BeforeClose', " + CHR(13) + ;
    "тогда Word должен открыть диалоговое окно 'Сохранить'." )
oWord.Activate() && Активировать Word.

oVFPCom.UnbindEvents(oWord, oWordEvents)

* Определение урезанного обработчика событий.
DEFINE CLASS WordEvents AS Custom
    PROCEDURE DocumentBeforeClose(lCancel, oDoc)
        * Дешевый способ вывести VFP на первый план.
        _SCREEN.AlwaysOnTop = .T.
        _SCREEN.AlwaysOnTop = .F.

        MessageBox("BeforeClose")

        * Вернитесь в Word, чтобы увидеть, что он представляет
        * (запретить Alt-Tab)
        oWord.Activate()
    ENDPROC
ENDDEFINE

```

Как видите, обработчик событий был урезан для обработки только одного события; просто чтобы сэкономить место в книге, запустите метод ExportEvents, чтобы получить полный код. Запускаемый код представляет собой простой MessageBox, показывающий, где будет выполняться ваш код. Он также позволяет перемещать нужные окна наверх, поэтому вам не придется нажимать Alt-Tab между приложениями.

После выполнения этого кода переменные PUBLIC все еще находятся в области видимости, поэтому обработчик событий продолжает работать. Когда вы закрываете окно сообщения, VFP активирует окно Word. Закройте документ или приложение (оба запускают событие DocumentBeforeClose). Когда событие DocumentBeforeClose срабатывает в объекте WordEvents, VFP выходит вперед,

выдает простое окно сообщения «BeforeClose», а затем возвращает вас в Word, чтобы увидеть, что Word продолжает работу с диалоговым окном «Сохранить».

Представьте себе, что можно перехватить множество событий, а полный доступ к документу открывает множество возможностей. Представьте себе, что вы позволяете пользователю редактировать документ Word, а затем передаете содержимое документа обратно в VFP, чтобы добавить текстовый элемент управления при сохранении документа. Вы можете перехватить щелчок правой кнопкой мыши и запустить код в зависимости от того, где пользователь щелкнул правой кнопкой мыши в документе. Вы можете отслеживать выход из Word и напоминать пользователю, что он не может закрыть Word во время выполнения кода Автоматизации. Круто. Очень круто.

VFPCOM с Excel

VFPCOM хорошо работает с Excel, с одним небольшим отличием: для объекта Application, объекта Workbook и объекта Worksheet существуют разные события. Запустите ExportEvents с помощью oExcel, oBook и oSheet и сравните результаты. Создайте обработчик событий для каждого типа объекта, а затем используйте BindEvents для привязки каждого типа объекта к его обработчику событий.

VFPCOM с PowerPoint

VFPCOM работает с PowerPoint несколько иначе. Несогласованность в VFPCOM и/или объектной модели PowerPoint означает, что VFPCOM не работает так, как заявлено. Одним из первых индикаторов является то, что результат ExportEvents VFPCOM – пустой файл; ни одно из событий не выглядит так, как если бы оно было открыто. Кроме того, если вы попытаетесь использовать BindEvents, вы получите сообщение об ошибке: «Код ошибки OLE 0x80070002: система не может найти указанный файл».

К счастью, есть обходной путь. Мы благодарим Джона В. Петерсена, написавшего DLL на Visual Basic, которая решает эту проблему. DLL, а также ее исходный код доступны в виде серии файлов vbPowerPoint (DLL, VBP, LIB и EXP) в файлах загрузки для разработчиков, доступных по адресу www.hentzenwerke.com. DLL использует VB для создания класса, который является экземпляром PowerPoint. Этот экземпляр VB предоставляет интерфейс событий. Вам необходимо зарегистрировать DLL, прежде чем использовать ее. Запустите следующую команду из пункта меню «Выполнить» кнопки «Пуск»:

```
REGSVR32 C:\YourDriveAndPathHere\vbPowerpoint.dll
```

Убедитесь, что путь соответствует вашему компьютеру. Чтобы удалить эту DLL, введите ту же команду, добавив в конце «/u».



Листинг 5, хранящийся как VFPCOMandPPT.PRG в файлах загрузки для разработчиков, доступных на сайте www.hentzenwerke.com, показывает, как использовать DLL для работы VFP. Основное отличие состоит в том, что oVBPowerPoint создается на основе класса VB и содержит приложение PowerPoint в свойстве oPowerPoint.

Листинг 5. Использование VFPCOM с PowerPoint. Здесь требуется обходной путь с использованием vbPowerPoint.DLL.

```
* VFPCOMandPpt.prg
* © 2000, Tamar E. Granor and Della Martin
* From: Microsoft Office Automation with Visual FoxPro
```

```

* Hentzenwerke Publishing. www.hentzenwerke.com

* Удалите все существующие ссылки на серверы.
* Это предотвращает потерю памяти оставшимися экземплярами.
RELEASE ALL LIKE o*

* Сделайте следующие переменные общедоступными в
* демонстрационных целях, чтобы они были доступны после
* запуска PRG для демонстрации функциональности VFPCOM.
PUBLIC oVFPCom, oVBPowerPoint, oVFPPowerPoint

#DEFINE ppLayoutTitle 1

* Создайте экземпляр VFPCOM.
oVFPCom = CreateObject("vfpcom.comutil")

* Создайте класс VB, который, в свою очередь, создаст PowerPoint.
oVBPowerPoint = CreateObject("vbPowerPoint.Class1")

* Добавьте небольшую презентацию.
oVBPowerPoint.oPowerPoint.Visible = .T.
oPres = oVBPowerPoint.oPowerPoint.Presentations.Add()
oSlide = oPres.Slides.Add(1, ppLayoutTitle)
oSlide.Shapes[1].TextFrame.TextRange.Text = "This is a Test"
oSlide.Shapes[2].TextFrame.TextRange.Text = "It is ONLY a Test."

* Создайте экземпляр объекта обработчика событий.
oVFPPowerPoint = CreateObject("vfpPowerPoint")

* Привяжите PowerPoint к объекту обработчика ошибок.
oVFPCom.BindEvents(oVBPowerPoint, oVFPPowerPoint)

* Пусть пользователь знает, что делать
_SCREEN.AlwaysOnTop = .T.
_SCREEN.AlwaysOnTop = .F.

MessageBox("Когда вы закроете это окно, вы перейдете " + ;
    "в PowerPoint." + CHR(13) + ;
    "Закройте PowerPoint, чтобы посмотреть, как с этим " + ;
    "справится VFPCOM." + CHR(13) + ;
    "VFP должен выдать MessageBox 'BeforeClose', " + CHR(13) + ;
    "тогда PowerPoint должен открыть диалоговое окно 'Сохранить'.")
oVBPowerPoint.oPowerPoint.Activate() && Активировать PowerPoint

DEFINE CLASS vfpPowerPoint AS custom

    PROCEDURE PresentationClose(Pres)
        * Дешевый способ вывести VFP на первый план.
        _SCREEN.AlwaysOnTop = .T.
        _SCREEN.AlwaysOnTop = .F.

        MessageBox("PresentationClose")

```



```
* Активировать PowerPoint  
oVBPowerPoint.oPowerPoint.Activate()  
ENDPROC
```

```
ENDDEFINE
```

13.3 Последнее слово

Взаимодействие между приложениями Office принимает множество форм. Одна из наиболее распространенных – повторное использование данных: взятие диаграммы, созданной для одной части приложения, и встраивание или связывание ее с документом или презентацией. Благодаря VFPCOM серверы могут отправлять информацию обратно в зависимости от событий, предупреждая программу о том, что пользователь завершил изменения, поскольку закрыл сервер (или закрывает его преждевременно). Мы показали вам несколько простых примеров, иллюстрирующих эти концепции; мы знаем, что вы полны идей о том, как масштабировать эти примеры в сложные приложения.

Глава 14. Обработка ошибок Автоматизации

Программирование Автоматизации похоже на другие виды ошибок программирования. Однако ошибки немного отличаются от тех, с которыми вы сталкивались в автономном приложении. Ваше приложение должно быть подготовлено к обработке ошибок, характерных для Автоматизации.

Большинство разработчиков сталкиваются с ошибками, возникающими только в среде их приложений. В FoxPro (или Basic, C или любом другом языке программирования) вы настраиваете свою среду, проверяете, доступны ли все вспомогательные библиотеки, а затем позволяете своему приложению работать. После запуска вашей программы вам не нужно беспокоиться о том, что еще установлено и работает на компьютере пользователя.

Автоматизация - это немного другое. CreateObject() не будет работать, если приложение Office не установлено или если оно правильно установлено в реестре, но исполняемый файл больше не находится на диске (некоторые пользователи до сих пор не научились удалять программы, предпочитая удалять каталоги, чтобы освободить дисковое пространство, что не удаляет настройки реестра). После того как вы создали экземпляр и сделали его видимым, пользователи, похоже, захотят закрыть его до того, как ваше приложение закончит работу с ним. Что еще более досадно, ни один из серверов Office ничего не знает о среде FoxPro, поэтому вы не можете полагаться на SET PATH, SET DEFAULT и другие настройки FoxPro.

14.1 Ошибки, связанные с Автоматизацией

К счастью для нас, существует ограниченное количество ошибок VFP, характерных для Автоматизации. В таблице 1 показан список всех ошибок, связанных с Автоматизацией. Как видно из их описаний, сообщения об ошибках относятся к корням Автоматизации в OLE.

Таблица 1. Ошибки Автоматизации, к которым вам нужно привыкнуть.

Номер ошибки	Описание ошибки
1420	Объект OLE недействителен или поврежден.
1421	Невозможно активировать сервер OLE.
1423	Ошибка создания объекта OLE.
1424	Ошибка копирования объекта OLE в буфер обмена.
1426	Код ошибки OLE 0x «имя».
1427	Код исключения OLE IDispatch «имя».
1428	Код исключения OLE IDispatch «номер» из «сервера»: «имя».
1429	«Ошибка OLE».
1440	Ошибка исключения OLE «имя». Объект OLE может быть поврежден.

Наиболее распространенными ошибками являются 1426-1429, которые обычно указывают на то, что на сервере возникла проблема с устанавливаемым свойством или выполняемым методом. Другие ошибки указывают на проблему с самим серверным приложением, например, на проблему с поиском сервера. Ошибки 1420, 1421 и 1423 возникают при создании экземпляра объекта при вызове CreateObject() или GetObject(). Ошибка 1424 возникает только при копировании в буфер обмена;

если вы не копируете объект в буфер обмена (как описано в главе 13 «Межофисная связь»), вам не нужно беспокоиться об этой ошибке. Ошибка 1440 является новой для VFP 6.0 и указывает на то, что сервер вызвал общую ошибку защиты и экземпляр больше не доступен.

14.2 Война 1426-1429

Нет, речь здесь не о многовековой битве, а о чередовании номеров ошибок. Но мы уверены, что вы часто будете сталкиваться с этими четырьмя ошибками при написании кода Автоматизации. Эти четыре ошибки возникают из-за того, что сервер Автоматизации отправляет обратно код ошибки. Какой из них вы получите, зависит от того, сколько информации готов предоставить сервер. Вот что разочаровывает: между этими ошибками нет никакой категорической разницы, и каждая ошибка имеет множество значений. Эти четыре ошибки возвращают все возможные проблемы с каждого сервера Автоматизации; напротив, FoxPro имеет сотни сообщений об ошибках!

Прежде чем вы подумаете, что это ужасный дизайн, вспомните, что FoxPro получает ошибки в стандартном интерфейсе COM/Automation. Невозможно присвоить уникальный номер каждой возможной ошибке на каждом возможном сервере, даже если бы мы могли гарантировать уникальность ошибок на всех серверах, у нас были бы номера ошибок, превышающие GUID, что вызвало бы другие проблемы.

Так почему же так неприятно работать с этими четырьмя ошибками? Вы не можете отличить фатальную ошибку (сервер отключился), синтаксическую ошибку и ошибку, которую можно исправить (например, ошибку деления на ноль) по номеру. Да, можно вести тщательную базу данных с текстом и номером каждой ошибки для каждого сервера, но из-за большого количества ошибок очень сложно отслеживать каждую ошибку, с которой вы сталкиваетесь, и не отставать от разработки.

Тонкие различия

Как мы уже говорили, разница между этими ошибками заключается в количестве информации, которую сервер готов предоставить. Ошибка 1426 «Код ошибки OLE 0x» является самой простой. Она возвращает только строку символов, содержащую шестнадцатеричное число и короткое сообщение, например: «Код ошибки OLE 0x800706ba: сервер RPC недоступен». В обработчике ошибок ERROR() возвращает 1426, а MESSAGE() возвращает «Код ошибки OLE 0x800706ba: сервер RPC недоступен».

Ошибки 1427, 1428 и 1429 возвращают одинаковую информацию: код исключения, за которым следует сообщение. Хотя FoxPro рассматривает их как разные ошибки, их сообщения кажутся пугающе похожими. Различия могут заключаться в том, что возвращается функцией AERROR().

Функция AERROR() Visual FoxPro возвращает дополнительную информацию об ошибке. Она создает массив из семи столбцов, содержащий различную информацию, в зависимости от того, является ли это ошибкой FoxPro, OLE или ODBC. В справке указано, что большинство ошибок FoxPro возвращают нулевые значения для большинства записей, за исключением ошибок OLE 1427 и 1429 и ошибки ODBC 1526. Фактически, ошибки OLE 1426 и 1428 также эффективно используют AERROR().

Таблица 2 выводит содержимое результирующего массива AERROR(). Ошибка 1426 устанавливает только первые три значения; остальным четырем присваиваются нулевые значения. Ошибки 1427, 1428 и 1429 используют до семи столбцов массива.

Таблица 2. Содержимое массива AERROR() с использованием ошибки 1429, сгенерированной Excel. Описанные позиции относятся только к ошибкам OLE; не общие ошибки FoxPro или ошибки ODBC, которые имеют разные описания.

Позиция	Содержание	Пример
1	Номер ошибки. То же, что ERROR().	1429
2	Текст сообщения об ошибке VFP. То же, что MESSAGE(). Обратите внимание, что это объединение некоторой другой информации в этом массиве.	«Код исключения OLE IDispatch 0 из Microsoft Excel: невозможно установить свойство Bold класса Font»
3	Текст сообщения OLE.	«Невозможно установить свойство Bold класса Font»
4	Приложение, отправляющее сообщение.	«Microsoft Excel»
5	Файл справки приложения, если он доступен (.Null., если нет).	«c:\Program Files\Microsoft Office\Office\1033\xlmain9.chm»
6	Идентификатор контекста справки, если он доступен (.Null., если нет).	0
7	Номер исключения OLE 2.0.	0

Ошибка 1426 устанавливает первые три столбца. Первый – это номер ошибки, который аналогичен запросу ERROR(). Второй – это сообщение об ошибке VFP, которое отображается в диалоговом окне ошибки, например: «Код ошибки OLE 0x80020026: неизвестное имя». Это та же строка, которая возвращается функцией MESSAGE(). Третий столбец содержит только часть текстовой строки; это строка, возвращаемая с OLE-сервера. VFP добавляет стандартный текст для пояснения сообщения; в случае 1426 это «код ошибки OLE 0x». Хотя вы можете проанализировать это из строки, вероятно, проще просто проверить третий столбец массива, который содержит код ошибки, за которым следует сообщение: «80020026: неизвестное имя». На самом деле, еще проще проверить SYS(2018), самую последнюю функцию параметров сообщений об ошибках. Остальные столбцы массива AERROR() имеют значение .Null., как и любая другая ошибка VFP.

Ошибки 1427-1429 заполняют все столбцы массива (при наличии файла справки и идентификатора контекста справки). Как видно из таблицы 1, сообщение VFP представляет собой объединение стандартного сообщения («код исключения OLE IDispatch»), номера исключения (столбец 7), имени приложения (столбец 4) и, наконец, текста OLE IDispatch. сообщение (столбец 3). Текст сообщения (столбец 3) также является значением, возвращаемым SYS(2018), самым последним параметром сообщения об ошибке, которое отличается от ошибки 1426, которая возвращает только текст (как в столбце 3). На самом деле SYS(2018) возвращает только первые 255 символов сообщения, что может иметь значение в более длинных сообщениях, как мы увидим позже.



Когда мы писали эту главу, стало очевидно, что 1426 и 1429 – наиболее

распространенные ошибки. Фактически, в нашем поиске примеров ошибок мы не смогли обнаружить даже живую ошибку 1427 или 1428. Это не означает, что вы никогда не увидите ошибку 1427 или 1428; вместо этого это означает, что мы не столкнулись со всеми ошибками Office Automation (хотя похоже, что они были!). Хотя Office может не использовать эти ошибки, другие COM-объекты могут, поэтому на всякий случай мы настоятельно рекомендуем вам создать обработчик ошибок для обработки 1427 и 1428.

Некоторые интересные наблюдения за ошибками

Чтобы по-настоящему понять, насколько утомительной может быть обработка этих ошибок, приведем несколько примеров. Первый пример показывает диапазон того, насколько информативными могут быть эти ошибки. Например, представьте себе следующую строку кода (да, в ней есть опечатка, чтобы проиллюстрировать нашу точку зрения):

```
oChart.ChartWizrd(oExcel.Sheets(cSheetName).Range(cGraphRange), ;  
    xlColumn, 6, xlColumns, 1, 1, 1, "", "", "Активы", "")
```

Полученное сообщение об ошибке показано на рисунке 1.

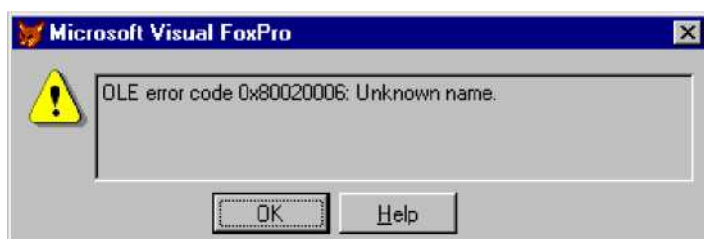


Рисунок 1. Пример ошибки 1426, полученный из-за неправильного написания имени метода. Хотя его краткость достойна восхищения, небольшая информация о том, что именно представляет собой неизвестное имя, была бы полезна.

Просто немного кратко, нет? Тем более, что в нашем примере не менее пяти имен: ссылки на oExcel, oChart, метод ChartWizard (который здесь является виновником), объект Sheets и объект Range. Честно говоря, VFP сообщает о том, что он получил от сервера, поэтому мы можем винить сервер в упущении одной небольшой, но очень полезной информации.

С другой стороны, есть ошибка, возникающая в следующей строке кода. Эта строка ссылается на несуществующий файл:

```
oWord.Documents.Open("Test.Doc")
```

Эта ошибка под номером 1429 показана на рисунке 2.



Рисунок 2. Другая ошибка, 1429, предлагает много информации. Там так много информации, даже есть полоса прокрутки!

Это сообщение настолько необычно длинное и информативное, что в окне сообщения об ошибке даже есть полоса прокрутки, позволяющая прочитать полный текст сообщения! Полный текст сообщения об ошибке выглядит следующим образом:

OLE IDispatch exception code 0 from Microsoft Word: This file could not be found.

Try one or more of the following:

- * Check the spelling of the name of the document.

- * Try a different file name.

(Test.Doc).

При тестировании этой ошибки мы обнаружили, что SYS(2018) содержит только первые 255 символов.

Другая ситуация заключается в том, что один и тот же номер ошибки дает разные виды ошибок, которые необходимо обрабатывать по-разному. Одним из видов ошибок является синтаксическая ошибка, эти надоедливые опечатки, которые обычно улавливает компилятор, или уведомление во время выполнения, информирующее вас о том, что вы поменяли местами эти два параметра. Другой вид ошибок – это когда пользователь уведомляется о том, что его система настроена неправильно или он допустил ошибку. Почти все ошибки FoxPro попадают в ту или иную категорию, поэтому гораздо проще обрабатывать каждую отдельную ошибку.

Ошибки 1426-1429 возвращают ошибки обеих этих категорий (и любой другой категории, о которой вы только можете подумать). Для иллюстрации давайте выберем ошибку 1426. Мы уже говорили о «коде ошибки OLE 0x80020026: неизвестное имя», показанном на рисунке 1, что обычно означает, что в коде есть метод или свойство с ошибкой и разработчику необходимо исправить код. Обработчик ошибок может изящно пропустить эту строку, закрыть модуль или выполнить любое другое подходящее действие (в идеале разработчик исправляет это до того, как ошибка дойдет до пользователя!). В отличие от этой ситуации, на рисунке 3 показана другая ошибка: «Код ошибки OLE 0x800706ba: сервер RPC недоступен». Эта ошибка может появиться в двух случаях: когда вы открываете сервер, и он не запускается правильно (или не запускается вообще), или позже, после того, как ваша программа откроет сервер, а пользователь закроет его до того, как ваша программа закончит с ним работать. Еще одна классическая ошибка 1426 – «Код ошибки OLE 0x80080005: сбой выполнения сервера». Это означает, что сервер зарегистрирован, но реальный EXE-файл не найден (обычно из-за удаления пользователем путем удаления папки вместо использования процедуры удаления). Эти две последние ошибки обрабатываются немного иначе, чем синтаксическая ошибка! Кто знает, сколько еще ошибок возвращается как 1426.

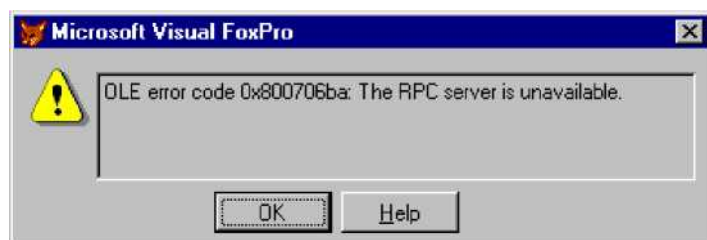


Рисунок 3. Еще одна ошибка 1426, указывающая, что сервер недоступен. Ошибки Автоматизации могут возвращать практически любые ошибки (включая синтаксические ошибки, ошибки времени выполнения и ошибки конфигурации), что усложняет программирование обработчика ошибок.

14.3 Стратегии обработки ошибок

В любой среде программирования существует два способа обработки ошибок: проактивный и реактивный (упреждающий и пассивный). Проактивный подход заключается в написании кода таким образом, чтобы предотвратить возникновение

ошибок. Например, вы можете избежать ошибок «Неверный индекс», проверив `oExcel.Workbooks.Count` и убедитесь, что `oExcel.Workbooks[43]` доступен. Реактивный подход заключается в том, чтобы позволить процедуре `ON ERROR` и/или методу `Error` формы или методу `Error` объекта обработать ошибку. Надежная стратегия обработки ошибок использует оба метода, чтобы гарантировать корректную обработку ошибок.

Насколько именно стратегии обработки ошибок должны быть проактивными или реактивными, зависит от личных предпочтений. Фактически, «личные предпочтения» на самом деле следует определять как «твердо укоренившиеся убеждения, защищаемые с упорством питбуля». У нас обеих были жаркие разговоры с другими о том, как справляться с ошибками, и мы не собираемся вступить в дискуссию о плюсах и минусах конкретных реализаций обработчиков ошибок.

Однако мы скажем, что, поскольку все ошибки Автоматизации сводятся всего к нескольким номерам ошибок, эта задача требует творческого решения проблем. Есть некоторые методы, которые вы можете обдумать и решить, как реализовать в существующей схеме обработки ошибок. Мы также будем настаивать на некоторых упреждающих и превентивных подходах.

ON ERROR и метод Error

Скорее всего, вы добавляете код Автоматизации в существующее приложение или используете существующую платформу приложения. Вы можете выбрать обработку ошибок Автоматизации в процедуре, вызываемой `ON ERROR`, которая отлично работает, особенно если код Автоматизации разбросан по всему приложению.

Если ваше приложение изолирует код Автоматизации, то более подходящим может оказаться локализованный обработчик ошибок. В приложении JFAST код Автоматизации запускается из одной формы, и почти весь код, запускаемый там, представляет собой код Автоматизации (в отличие от операторов VFP). Обработчик ошибок хорошо работает в виде метода `Error`, который переопределяет настройку `ON ERROR`. Другое логичное место для обработки ошибок – это метод `Error` класса-оболочки, который вам следует использовать (см. главу 15, «Подведение итогов работы с серверами»). Просто помните, что класс-оболочка не может перехватывать ошибки в коде, который выполняется вне класса-оболочки.

Если вы работаете в команде разработчиков, прежде чем приступить к оснащению подпрограмм `ON ERROR` вашего приложения новейшими и лучшими средствами Автоматизации обработки ошибок, попробуйте изолировать их и усовершенствовать в методе `Error` формы. Таким образом, вы можете усовершенствовать процедуры обработки ошибок во время тестирования формы, а затем внести одно изменение в обработчик ошибок приложения. Таким образом, вы сможете минимизировать количество изменений кода ошибки приложения (и количество раз, когда члены вашей команды появляются у вашего стола и требуют знать, почему вы это «исправили»).



В листинге 1 (`ErrorMethod.PRG` в файлах для загрузки для разработчиков, доступных на сайте www.hentzenwerke.com) показан пример кода, который можно использовать в качестве отправной точки. Несомненно, вы найдете в своем приложении другие способы обработки определенных разновидностей ошибок 1426 и 1429. Это хорошая отправная точка для дальнейшего развития по мере того, как вы будете лучше знакомиться с типами ошибок, с которыми сталкивается ваше приложение. Этот простой скелет создает строку, указывающую, что произошло, и

отображает ее пользователю. Последний бит метода закрывает выполняющийся код, который вы захотите закомментировать во время разработки, заменив его на SUSPEND или вернув вас к вашему коду (например, Ignore), чтобы найти следующую ошибку.

Листинг 1. Пример формата обработчика ошибок Автоматизации, находящегося в методе Error формы. Этот метод доступен как ErrorMethod.PRG в файлах загрузки для разработчиков.

```
* ErrorMethod.prg
* © 2000, Tamar E. Granor and Della Martin
* From: Microsoft Office Automation with Visual FoxPro
* Hentzenwerke Publishing. www.hentzenwerke.com

LPARAMETERS nError, cMethod, nLine
LOCAL ErrorArray[1], AppName, Instructions, oWkBk

m.ErrorMessage = MESSAGE()

#DEFINE CR CHR(13)
#DEFINE wdDoNotSaveChanges 0

* Получите информацию из массива ошибок - она полезна только для
* ошибок OLE 1426-1429.
= AERROR(ErrorArray)

* Создайте удобное для пользователя сообщение об ошибке.
DO CASE
CASE nError = 1420  && Объект OLE недействителен или поврежден.
    m.Instructions = "Попробуйте переустановить Office."

CASE nError = 1421  && Невозможно активировать OLE-сервер.
    m.Instructions = "Попробуйте переустановить Office."

CASE nError = 1422  && Ошибка при сохранении объекта OLE.
    m.Instructions = "Сообщить об этой ошибке."

CASE nError = 1423  && Ошибка создания объекта OLE.
    m.Instructions = "Сообщить об этой ошибке."

CASE nError = 1424  && Ошибка копирования объекта OLE в буфер
    && обмена.
    m.Instructions = "Сообщить об этой ошибке."

CASE nError = 1426  && Код ошибки OLE 0x"имя".

    * Это хорошее место для проверки наличия ErrorArray[3]
    * и определения того, что делать, на основе
    * LEFT(ErrorArray[3], 8), кода ошибки.
    * Один из примеров показан ниже.

    IF LEFT(ErrorArray[3], 8) = "800706ba" && Сервер RPC
        && недоступен
        m.Instructions = "Сервер был отключен."
```

```
ELSE
    m.Instructions = "Сообщить об этой ошибке"
ENDIF

CASE INLIST(nError, 1427, 1428, 1429 )
    * 1427 = Код исключения OLE IDispatch "имя".
    * 1428 = Код исключения OLE IDispatch "номер"
    *                  из "сервера": "имя".
    * 1429 = "Ошибка OLE"

    * Не забывайте, что вы можете запросить следующее:
    * ErrorArray[3], текст сообщения OLE.
    * ErrorArray[5], файл справки приложения.
    * ErrorArray[6], идентификатор контекста справки.
    * ErrorArray[7], номер исключения OLE 2.0 – это
    *           особенно полезно для обработки отдельных исключений.

    IF NOT ISNULL(ErrorArray[4])
        m.AppName = "Проблема с приложением: " + ErrorArray[4]+CR
    ELSE
        m.AppName = ""
    ENDIF

    m.Instructions = AppName

CASE nError = 1440 && Ошибка исключения OLE "имя".
                    && Объект OLE может быть поврежден.
    m.Instructions = "Сообщить об этой ошибке."

OTHERWISE
    m.Instructions = "Сообщить об этой ошибке."

ENDCASE

= MessageBox("НЕОЖИДАННАЯ ОШИБКА АВТОМАТИЗАЦИИ:" + CR + CR + ;
    m.ErrorMessage + CR + ;
    "Ошибка №: " + TRANSFORM(nError, '9999') + " в " + cMethod + ;
    CR + "в строке " + TRANSFORM(nLine, '9999999') + CR + CR + ;
    m.Instructions, "ОШИБКА АВТОМАТИЗАЦИИ")

* В нашем примере закрываются открытые объекты. Вашему приложению
* может потребоваться другая стратегия, например закрытие модуля,
* закрытие приложения или выдача окна сообщения «Игнорировать,
* Повторить, Возобновить». Измените это в соответствии с вашим
* приложением

IF TYPE("oPowerPoint") = "O" AND NOT ISNULL(oPowerPoint)
    oPowerPoint.Quit()
ENDIF

IF TYPE("oWord") = "O" AND NOT ISNULL(oWord)
    oWord.Quit( wdDoNotSaveChanges )
ENDIF
```

```

IF TYPE("oExcel") = "O" AND NOT ISNULL(oExcel)
    FOR EACH oWkBk IN oExcel.Workbooks
        * Вставьте свой любимый метод сохранения или закрытия
        * каждой книги, чтобы Excel закрывался без сообщения.
    NEXT oWkBk

    oExcel.Quit()
ENDIF

* Удалите переменные - измените их в соответствии с вашим
* приложением.
RELEASE oPowerPoint && и любые другие ссылки на объекты PowerPoint
RELEASE oExcel && и любые другие ссылки на объекты Excel
RELEASE oWord && и любые другие ссылки на объекты Word

ThisForm.Release()

RETURN TO MASTER

```

Почему мы отключаем все в этом обработчике ошибок? При таком большом количестве различных типов ошибок, проходящих через 1426-1429, мы не знаем, повлияет ли эта ошибка только на одну строку (например, невозможность установить свойство полужирного шрифта) или же она вызовет каскад ошибок (как в случае отключения сервера). Это самый безопасный метод предотвращения каскадных ошибок. По мере разработки приложения вы будете находить определенные номера ошибок OLE (1427, 1429) или коды ошибок (1426), которые можно обрабатывать в рамках этой платформы. Например, в некоторых ситуациях, таких как пакетная обработка, может быть достаточно зарегистрировать определенные ошибки и продолжить.

Предотвращение ошибок

Вместо того, чтобы реагировать на возникающие ошибки, имеет смысл попытаться их предотвратить. Многие ошибки можно предотвратить, например, проверив свойство Count коллекции перед указанием индекса, чтобы убедиться, что индекс не выходит за пределы диапазона.

Другая часто встречающаяся ошибка – предположить, что серверное приложение знает о среде FoxPro. Большинство разработчиков полагались на настройки SET PATH и SET DEFAULT для поиска файлов. Каждый раз, когда вы указываете имя файла, который находится в каталоге по умолчанию, обязательно добавляйте к имени файла префикс SYS(5) + SYS(2003), чтобы сервер мог его найти. Если он не находится в каталоге по умолчанию, обязательно укажите диск и путь. Использование FullPath() – это быстрый способ получить путь к файлу, хотя наличие в пути более одного файла с одинаковым именем может привести к неожиданным результатам.

Сервер также не имеет ни малейшего понятия о настройке SET SAFETY. Хуже того, если вы автоматизируете все четыре приложения Office, вы обнаружите, что каждое из них ведет себя по-разному, когда вы пытаетесь перезаписать файл или выйти из приложения с документом в несохраненном состоянии. Проверьте главы «Основы» для каждого сервера, чтобы убедиться, что ваше приложение делает то, что вы хотите.

Если ваши серверы регулярно не открываются, проверьте статус SET OLEOBJECT,

который должен быть включен. Когда SET OLEOBJECT включен, он сообщает VFP выполнить поиск сервера в реестре. Если для него установлено значение OFF, VFP не сможет выполнять поиск в реестре, где находятся все серверы Автоматизации.

Обеспечение существования сервера

Ваше приложение должно убедиться, что на компьютере пользователя установлен Office и что он правильно зарегистрирован. FoxPro находит объекты COM (следовательно, Автоматизации) через реестр. Быстрый просмотр реестра может сказать вам, установлен ли сервер.

Вы можете сказать, что он был установлен. Но что, если пользователю не хватает места на диске, и он решает освободить немного места, удалив всю серию каталогов Office, а не удаляя Office? Записи реестра целы, но указывают на несуществующий файл. Это вам тоже нужно проверить.

Знаете ли вы, что на своих компьютерах пользователи могут иметь приложения Office, и они успешно работают с помощью значков на рабочем столе и меню «Пуск», но Автоматизация не может их найти? Да, такое может случиться, и вы потратите недели, пытаясь это понять. Проблема в реестре. Делла убедилась в этом на собственном горьком опыте, когда пользователь решил, что ему не нравится расположение файлов Office по умолчанию, и вручную переместил каталог, а затем изменил значки и меню «Пуск», чтобы отразить новое расположение. Проблема была в том, что он не менял реестр (это не его вина, он даже не знал, что такое реестр). Сообщение об ошибке «Excel не установлен на этом компьютере должным образом; пожалуйста, переустановите его» ужасно смутило его, поскольку он использовал Excel, поэтому его необходимо установить правильно. Еще одна причина проверить файл, который, по мнению реестра, он должен найти.

Еще одна ошибка: если вы используете Office 97, вам нужно использовать как минимум SR-1 (есть также SR-2 и SR-2B), поскольку он устраняет некоторые проблемы с Автоматизацией. Проверьте номера версий, чтобы убедиться, что у вашего пользователя установлена правильная версия. Хорошей новостью является то, что проверить реестр несложно. Используется несколько функций API для доступа к реестру. Если вы думаете, что API сложно использовать, смиритесь с этим! Но сначала несколько слов о реестре.

Регистр 101

Реестр представляет собой иерархический список, состоящий из *ключей*. На рисунке 4 показан редактор реестра с шестью основными ключами (показана Windows 98). Открытие ключа открывает ряд *подразделов*. На рисунке 5 показан ключ HKEY_CLASSES_ROOT, открытый для раскрытия его первого уровня подразделов. Если вы посмотрите файл справки редактора реестра, вы не найдете ссылки на термин «подключ». Различие между ключом и подразделом полезно при передаче параметров функциям API.

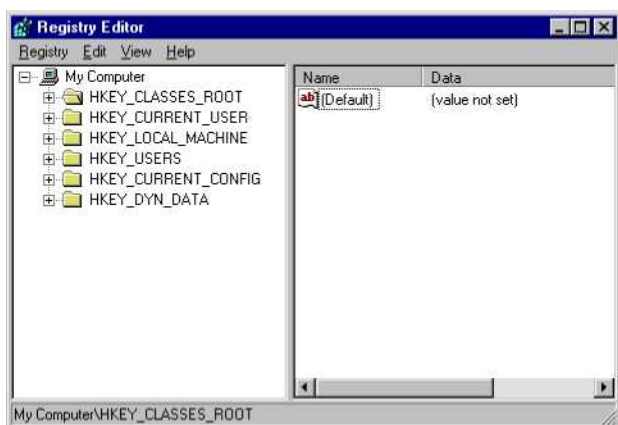


Рисунок 4. Редактор реестра откроется, чтобы отобразить ключи реестра. Ключи реестра различаются в зависимости от версии Windows. Здесь показана Windows 98.

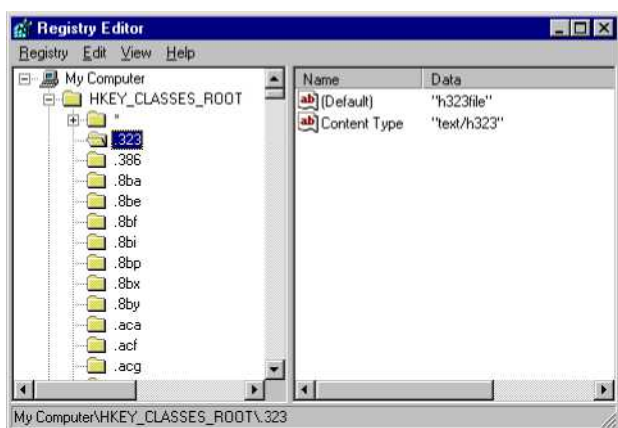


Рисунок 5. Редактор реестра показывает открытый ключ HKEY_CLASSES_ROOT с отображаемыми подразделами. На панели справа показаны значения, связанные с подразделом.

С каждым подразделом связано одно или несколько *значений*. Значение состоит из двух частей: имени и данных. Имя представляет собой строку символов, описывающую содержимое данных. На рисунке 5 показан подраздел .323 с двумя значениями: значением «(Default)» для «h323file» и значением «Тип контента» для «text/h323».

Как мы уже говорили, реестр представляет собой иерархический список ключей. Главный ключ содержит подразделы, подразделы могут иметь подразделы и так далее. Многие программы имеют набор ключей, разбросанных по всему реестру. Для каждого приложения Office, используемого в задачах Автоматизации, существует ряд ключей.

Автоматизация использует ключ имени класса для поиска информации о сервере. Имя класса Excel «Excel.Application» показано на рисунке 6. Ключ имени класса имеет подраздел CLSID, который содержит GUID или глобальный уникальный идентификатор. Идентификаторы GUID, такие как Excel, показанные на правой панели рис. 6, генерируются разработчиком приложения (в данном случае Microsoft). Реестр использует эти GUID для отслеживания программ. Это число гарантирует уникальное представление каждого приложения в реестре.

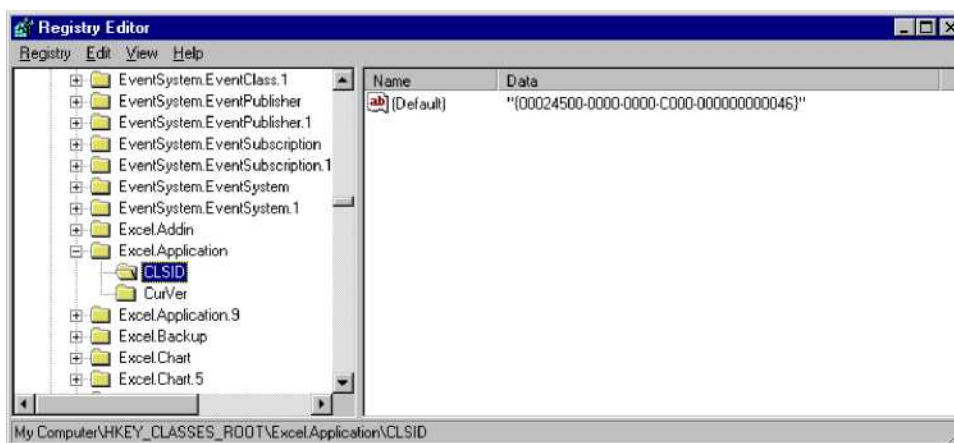


Рисунок 6. Ключ имени класса для Excel: «Excel.Application». Автоматизация ищет ключ имени класса, находит его подраздел CLSID и ищет значение по умолчанию, которое содержит GUID. Это используется для дальнейшего поиска в реестре.

Как только мы получим значение данных ключа CLSID, мы сможем использовать его для поиска ключа, значение которого содержит полное имя файла. Иерархия ключей – HKEY_CLASSES_ROOT, CLSID, GUID (глобальный уникальный идентификатор) и, наконец, LocalServer32. Его единственное значение – это полное имя EXE. На рисунке 7 показан редактор реестра, открытый с помощью этого ключа для Excel.

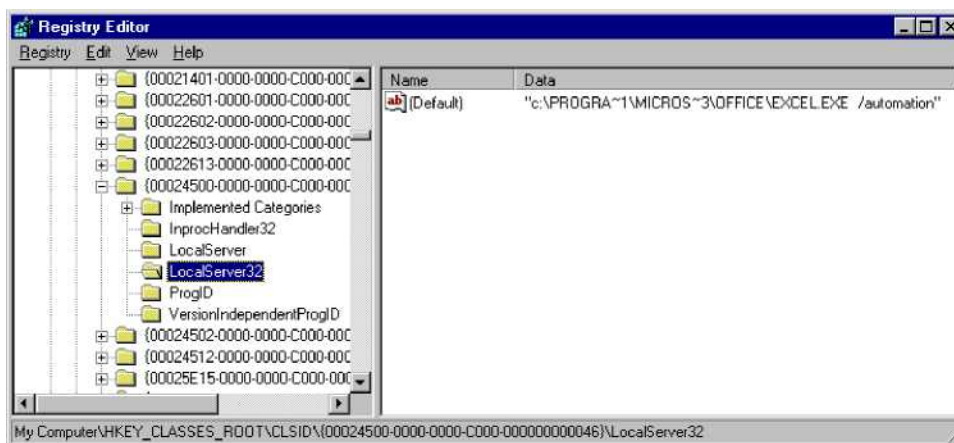


Рисунок 7. Некоторые ключи реестра для Excel. На левой панели показана иерархия подразделов. Родительские ключи, находящиеся вне верхней части экрана, – это HKEY_CLASSES_ROOT\CLSID. Значением по умолчанию для ключа LocalServer32 является исполняемый файл, вызываемый вызовами Автоматизации.

Теперь вы знаете, что искать. Теперь вам нужно знать, как это сделать. Краткое описание API-интерфейсов.

API 101

API означает интерфейс прикладного программирования. Windows поставляется с рядом библиотек DLL, содержащих функции доступа ко всем видам информации системного уровня. Этот набор функций включает в себя Windows API.

Чтобы использовать одну из этих функций API, необходимо сначала зарегистрировать функцию с помощью команды DECLARE. Команда DECLARE принимает несколько аргументов. Первый – это необязательный аргумент, указывающий тип данных значения, возвращаемого функцией. Возможные варианты: SHORT, INTEGER, SINGLE, DOUBLE, LONG и STRING. Следующий аргумент – это имя регистрируемой функции. Этот аргумент чувствителен к

реестру. Если вы получаете ошибки, обязательно проверьте регистр, поскольку большинство вызовов Windows API имеют смешанный регистр. Третий аргумент сообщает имя библиотеки. Вы можете указать DLL или, если это стандартный вызов Win32API (обычно в библиотеках ядра, пользователя и GDI), вы можете указать ключевое слово WIN32API. Обязательно укажите необходимое ключевое слово IN. В случае, если имя функции противоречит существующему ключевому слову FoxPro, вы можете добавить необязательную фразу AS <aliasname>, чтобы переименовать функцию.

Наконец, вам необходимо указать типы данных каждого параметра функции. Передайте список типов данных, разделенных запятыми. Вы можете выбрать INTEGER, SINGLE, DOUBLE, LONG и STRING. Если параметр должен передаваться по ссылке (где функция будет изменять значение переменной), необходимо добавить знак @ после типа данных (и перед запятой). Вам также необходимо использовать @ при вызове функции для передачи по ссылке, но если вы опустите @ в DECLARE, вы не сможете передать функцию по ссылке. Вы также можете добавить необязательное имя после @; FoxPro вообще не использует его, кроме как для документирования, но это достаточно веская причина, чтобы это сделать.

Функция RegOpenKey используется для открытия ключа. После открытия ключа вы можете запросить его значения. Команда DECLARE для функции RegOpenKey:

```
DECLARE RegOpenKey IN ADVAPI32.DLL ;
    INTEGER Key, STRING Subkey, INTEGER @ ReturnedHandle
```

Теперь вы можете использовать эту функцию, как и любую другую функцию, например:

```
hOpenKey = 0
= RegOpenKey(2147483648, ;
    "CLSID\{000209FF-0000-0000-C000-000000000046}\LocalServer32", ;
    @hOpenKey)
```

Просто убедитесь, что у вас определена переменная надлежащего типа, прежде чем передавать переменную по ссылке. Перед вызовом переменная hOpenKey должна быть инициализирована значением 0, как в предыдущем примере.

Функции API не так уж и сложны в использовании. Самое сложное – это знать, как найти доступные функции. К счастью, команда Visual FoxPro предоставляет классы-оболочки (содержащие полный исходный код) для функций API, используемых для доступа к реестру. В VFP 6 доступен базовый класс FoxPro под названием Registry.VCX. В VFP 5 вы найдете образец файла Registry.PRG, содержащий тот же код, что и VFP 6 VCX, за исключением определенного в коде.

Создав экземпляр класса Registry для переменной oRegistry, вы можете воспользоваться преимуществами классов-оболочек. В VFP 6 используйте функцию NewObject(), которая работает аналогично CreateObject(), за исключением того, что вы указываете, в каком VCX находится класс. VCX не обязательно должен быть открыт или находиться в пути к NewObject() для создания экземпляра объекта:

```
oRegistry = NewObject("Registry", HOME() + "FFC\Registry")
```

В VFP 5 вам необходимо установить процедурный файл в Registry.PRG, а затем создать экземпляр класса:

```
SET PROCEDURE TO \vfp5\samples\classes\registry.prg ADDITIVE
oRegistry = CreateObject("Registry")
```

Класс Registry – это специальный класс со множеством методов, упрощающих использование функций API. Он выполняет DECLARE для всех функций

(независимо от того, сколько функций вы собираетесь использовать), а затем использует свойства для хранения многих значений, передаваемых по ссылке. Чтобы открыть ключ, вы передаете по ссылке переменную, которая была инициализирована значением 0. API присваивает этой переменной дескриптор. Затем вы используете этот дескриптор в функции RegEnumValue(), также передавая ему пустые переменные для заполнения. Метод OpenKey класса реестра хранит дескриптор свойства для использования методом EnumKeyValues (или любым другим методом, использующим открытый ключ). Это позволяет вам сосредоточиться на получении информации из реестра, а не на попытках управлять множеством переменных.

Копаем

Мы установили, что искать, где искать и как это найти; теперь давайте соберем это вместе. После открытия ключа с помощью метода OpenKey класса реестра вы можете получить список значений ключа с помощью EnumKeyValues.

Открыв ключ, вы можете перечислить его значения в массиве. EnumKeyValues принимает один параметр – пустой массив, передаваемый по ссылке. Этот метод заполняет ваш массив списком всех значений, помещая имя в столбец 1, а данные в столбец 2. Хотя ключ может содержать множество значений, в обоих ключах, которые вы хотите проверить, возвращается только одно значение. Первая позиция массива содержит имя ключа «(Default)», а вторая позиция содержит данные, которые вы ищете. Эти две функции класса Registry продемонстрированы в листинге 2, который включен как CheckServer.PRG в файлы для загрузки для разработчиков, доступные по адресу www.hentzenwerke.com. Он настроен на использование одного из имен классов сервера Office в качестве параметра, например:

```
? CheckServer("Word.Application")
```

Функция возвращает логическое значение, указывающее, установлен ли сервер. Обратите внимание: как только мы найдем значение ключа, нам нужно удалить некоторые конечные символы (пример строки см. на рисунке 7). Затем проверяем существование файла.

Листинг 2. Передайте имя класса приложения, чтобы узнать, зарегистрировано ли приложение Office и находится ли оно на диске. Попробуйте изменить имя Word.EXE, чтобы проверить его; обязательно переименуйте EXE-файл обратно, когда закончите тестирование.

```
* CheckServer.prg
* © 2000, Tamar E. Granor and Della Martin
* From: Microsoft Office Automation with Visual FoxPro
* Hentzenwerke Publishing. www.hentzenwerke.com
```

```
LPARAMETER cServerName
```

```
LOCAL oRegistry, cClassID, cEXEName, lEXEExists, ;
      aClassIDValues, aClassIDValues, aServerNameValues
```

```
*IF VERSION() = "Visual FoxPro 06"
IF VERSION() = "Visual FoxPro 09"
    oRegistry = NewObject("Registry", HOME() + "FFC\Registry")
ELSE
    SET PROCEDURE TO HOME() + "samples\classes\registry.prg"
ADDITIVE
```

```

oRegistry = CreateObject("Registry")
ENDIF

lEXEExists = .F.
DECLARE aClassIDValues[1], aServerNameValues[1]

WITH oRegistry

    * Найдите CLSID сервера. Сначала ищите ключ класса.
    IF .OpenKey(cServerName + "\CLSID") = 0

        * Ключ класса открыт, теперь перечислите его значения.
        .EnumKeyValues(@aClassIDValues)

        * Часть данных первого (единственного) возвращаемого
        * значения — это CLSID. Найдите ключ LocalServer32 для
        * CLSID.
        IF .OpenKey("CLSID\" + aClassIDValues[1,2] + ;
            "\LocalServer32") = 0

            * Перечислите значения LocalServer32.
            .EnumKeyValues(@aServerNameValues)

            * EXE-файл сохраняется в первом (единственном)
            * возвращенном значении данных.
            cEXEName = aServerNameValues[2]

            * Возвращаемое значение может иметь в конце
            * «-Automation», «/Automation» или «/AUTOMATION» и
            * другие завершающие элементы. Удалите это.
            IF "AUTO" $ UPPER(cEXEName)
                cEXEName = LEFT(cEXEName, ;
                    ATC("AUTO", UPPER(cEXEName)) - 2)
            ENDIF

            * Убедитесь, что файл существует.
            lEXEExists = FILE(cEXEName)

        ENDIF
    ENDIF
ENDWITH

RETURN lEXEExists

```

Версии Office

Поскольку версии 97 и 2000 каждого приложения Office используют один и тот же GUID, код проверяет только существование одной из версий; он не сообщает вам, какой из них (если вы специально не проверите конкретную версию в имени класса, как в «Word.Application.9»). В VFP 6 представлена функция AGetFileVersion(), которая позволяет вам проверять информацию о версии, указанную в файле. Она принимает два параметра: первый — это массив для заполнения информацией о версии, а второй — имя файла, из которого можно извлечь информацию о версии. Функция возвращает количество строк в результирующем массиве, которое в

случае успеха равно 15. Из 15 элементов массива нас интересует четвертый – версия файла (остальные 14 элементов см. в справке). Элемент 4 – это строка символов, которая выглядит примерно так: «9.0.2719».

В листинге 2 имя файла помещается в переменную cEXEName; вы можете добавить следующий сегмент кода в конец кода и передать параметр версии по ссылке, чтобы функция CheckServer() проверила сервер и версию.

```
* Убедитесь, что это путь к Excel на вашем компьютере,  
* или используйте переменную с тем же именем в листинге 2.  
cEXEName = "C:\Program Files\Microsoft Office\Office\Excel.EXE"
```

```
nRows = AGetFileVersion(aVersionInfo, cEXEName)  
cVersion = aVersionInfo[4]
```

Теперь вам решать, что делать со строкой символов: преобразовать ее в число или оставить в качестве символа для сравнений. Имейте в виду, что версия 9 – это Office 2000, а версия 8 – Office 97.

AGetFileVersion() доступен только в VFP 6. VFP 5 имеет функцию GetFileVersion() в FoxTools. Она принимает те же два параметра, но их порядок обратный. Кроме того, перед передачей по ссылке массив уже должен существовать и иметь как минимум 12 строк и один столбец. Вот код, переписанный для VFP 5:

```
cExeName = "C:\Program Files\Microsoft Office\Office\Excel.EXE"  
SET LIBRARY TO FOXTOOLS.FLL  
DECLARE aVersionInfo[12]  
= GetFileVersion(cExeName, @aVersionInfo)  
cVersion = aVersionInfo[4]
```

14.4 Подведем итоги

Обработка ошибок Автоматизации может быть сложной задачей, но не невозможной. Важно включить метод устранения ошибок Автоматизации в общую стратегию обработки ошибок. А еще лучше, посмотрите следующую главу о классах-оболочках и включите свою стратегию обработки ошибок Автоматизации в класс-оболочку.

Глава 15. Подведение итогов работы с серверами

Серверы Office настолько разнообразны, что ими сложно пользоваться. Классы-оболочки упрощают работу и защищают вас от изменений в будущих версиях.

Каждый сервер Office обеспечивает огромную мощность, но его самая сильная сторона одновременно является и самой большой слабостью. Чтобы ими пользоваться, вам нужно их изучить. Для каждого приложения, которое вы хотите автоматизировать, вам придется многому научиться. Несмотря на обещание полиморфизма, работа с несколькими серверами Office означает изучение множества исключений.

Что еще хуже, каждый раз, когда Microsoft выпускает новую версию Office, все меняется. Добавляются новые свойства и методы, свойства получают дополнительные значения и, самое главное, методы принимают новые параметры. В последние годы Microsoft значительно очистила свою деятельность, поэтому изменение порядка параметров теперь является необычным, и маловероятно, что параметр будет обязательным, если раньше он был необязательным, но время от времени даже такие вещи все еще случаются.

Иногда изменения в новой версии являются хорошими и предлагают улучшенные способы выполнения задачи Автоматизации. Но если вы использовали определенный метод во всем своем коде, вам не захочется возвращаться и менять его в десятках или сотнях мест.

Так как же можно использовать Автоматизацию, не превратив ее в рутинную работу на полный рабочий день? Ответ заключается в том, чтобы обернуть операции, которые вы используете, в классы Visual FoxPro.

15.1 Зачем обертывать серверы?

Класс-оболочка, как следует из названия, обходит другой класс или иногда код, который не является объектно-ориентированным (например, что-то вроде FoxTools), чтобы придать ему объектную структуру. Цель, как правило, состоит в том, чтобы обеспечить более понятный интерфейс и предоставить возможность создания подклассов. (Если вы знакомы с шаблонами проектирования, вы можете признать в классах-оболочках реализацию шаблона адаптера. Чтобы узнать о шаблонах, очень полезном инструменте проектирования классов, мы рекомендуем классическую книгу *Паттерны объектно-ориентированного проектирования* Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж.)

При работе с Автоматизацией классы-оболочки делают за вас несколько вещей.

Во-первых, они позволяют упростить. Многие методы серверов Office ожидают множество параметров. В своих классах-оболочках вы можете скрыть все эти дополнительные параметры. Методы, которые вы выбираете для предоставления, принимают только те параметры, которые вам действительно нужны, а с остальным код может справиться самостоятельно. В качестве альтернативы вы можете предоставить набор свойств, позволяющий вызывающей программе устанавливать те параметры, которые ей нужны, а затем использовать вызов метода для установки значений по умолчанию для любых неуказанных параметров.

Во-вторых, классы-оболочки защищают вас от изменений Microsoft. Если порядок параметров серверного метода меняется, вам нужно исправить его только в одном месте вашего класса-оболочки, а не везде, где вы использовали этот метод.

В-третьих, классы-оболочки позволяют централизовать обработку ошибок. Например, вместо того, чтобы писать весь код вашего приложения, чтобы

беспокоиться о том, доступен ли у вас экземпляр сервера, код вашего приложения вызывает методы вашего класса-оболочки, и эти методы беспокоятся о том, доступен ли сервер. Точно так же методы вашего класса-оболочки могут беспокоиться о том, есть ли открытый документ и что делать, если вызов сервера не удался.

15.2 Структурирование классов-оболочек

Оставшаяся часть этой главы посвящена набору классов-оболочек для серверов Office. На вершине иерархии классов находится абстрактный класс `cusWrapServer`, основанный на пользовательском классе Visual FoxPro. Он имеет несколько пользовательских свойств и методов, подробности о которых рассматриваются в следующих разделах. (На самом деле, хотя Custom технически является подходящим классом для этого, поскольку классы-оболочки не нуждаются в контейнерных возможностях Custom, рассмотрите возможность начать с более простого класса, такого как Line, который не использует столько ресурсов, сколько Custom.)

Класс `cusWrapServer` может быть подклассом для создания отдельных классов-оболочек для Word, Excel и PowerPoint (и, возможно, также для некоторых других серверов Автоматизации). Объектная модель Outlook настолько отличается от других серверов Office, что эта структура классов-оболочек для нее не подходит, но, конечно, вы можете разработать класс-оболочку для Outlook.

Все классы, описанные в этой главе, включены в файлы для загрузки для разработчиков, доступные по адресу www.hentzenwerke.com. Поскольку некоторые функции, используемые в классах, уникальны для VFP 6, что предотвращает открытие классов в конструкторе классов в более старых версиях, в дополнение к библиотеке классов предоставляется версия кода.

15.3 Фундаментальные операции

Независимо от того, с каким сервером вы имеете дело, вам необходимы определенные операции: открытие сервера, закрытие сервера, открытие и закрытие документа, проверка, открыт ли сервер, проверка, есть ли у вас документ для работы и т. д. . Методы и свойства для обработки этих операций добавляются в `cusWrapServer`, хотя некоторые из методов являются абстрактными, поскольку требуют кода, специфичного для сервера. В таблице 1 показаны пользовательские свойства `cusWrapServer`. В таблице 2 показаны пользовательские методы класса.

Таблица 1. Свойства `cusWrapServer`. Эти свойства добавляются в абстрактный класс-оболочку.

Свойство	Тип	Описание
<code>cServerName</code>	Character	Имя сервера Автоматизации, например «Word.Application».
<code>lOpenAsNeeded</code>	Logical	Указывает, открывать ли сервер при вызове метода, требующего его использования. Если это .F., некоторые методы могут дать сбой.
<code>oDocument</code>	Object	Ссылка на объект активного документа. «Документ» здесь используется в широком смысле для обозначения основного типа объекта, используемого приложением.
<code>oServer</code>	Object	Ссылка на объект на сервере.

Таблица 2. Методы `cusWrapServer`. Эти пользовательские методы добавляются в абстрактный класс, поэтому они доступны всем классам-оболочкам. Многие из них абстрактны на этом уровне; то есть код, который будет выполняться, должен быть указан в подклассах.

Метод	Описание
<code>CloseDocument</code>	Закрывает документ основного типа для сервера. Абстрактный.
<code>CloseServer</code>	Закрывает экземпляр сервера. Абстрактный.
<code>HideServer</code>	Делает экземпляр сервера невидимым. Абстрактный
<code>IsDocOpen</code>	Указывает, открыт ли документ. Абстрактный.
<code>IsServerOpen</code>	Указывает, доступен ли экземпляр сервера. Абстрактный.
<code>NewDocument</code>	Создает новый документ основного типа для сервера. Абстрактный.
<code>oDocument_Access</code>	Возвращает ссылку на активный документ.
<code>OpenDocument</code>	Открывает документ основного типа для сервера.
<code>OpenServer</code>	Открывает экземпляр сервера.
<code>OpenServerDocument</code>	Серверная часть <code>OpenDocument</code> . Обрабатывает детали, специфичные для сервера. Абстрактный.
<code>SaveDocument</code>	Сохраняет документ основного типа для сервера.
<code>ShowServer</code>	Делает сервер видимым. Абстрактный.
<code>TellUser</code>	Отображает сообщение пользователю.
<code>zReadme</code>	Метод документации.

Разговор с пользователем

Все общение с пользователем осуществляется через один метод `TellUser`. Это обеспечивает централизованные службы сообщений для класса-оболочки. В предоставленной версии `cusWrapServer` `TellUser` довольно прост (и действительно ориентирован больше на разработчиков, использующих класс-оболочку, чем на конечных пользователей). Он использует либо `WAIT WINDOW`, либо `MessageBox` и не принимает никаких входных данных. Хотя эта версия подходит для простого тестирования, вы, вероятно, захотите заменить этот метод тем, который связан со службами сообщений, уже используемыми в ваших приложениях.

```
* cusWrapServer.TellUser
* Передайте сообщение пользователю. Отправляя все
* сообщения с помощью этого метода, можно легко
* изменить механизм, используемый для сообщений.
* Это всего лишь очень простой обработчик сообщений.
```

```
LPARAMETER cMessageText, lDontWait
```

```
IF SYS(2335) = "1" ; && Не в автоматическом режиме.
    AND This.Application.StartMode = 0    && Обычный интерактивный
```

```

&& сеанс VFP.

IF lDontWait
    WAIT WINDOW cMessageText NOWAIT
ELSE
    MessageBox( cMessageText )
ENDIF
ELSE && Невозможно создать диалоговое окно пользовательского
    && интерфейса.
    ERROR cMessageText && генерирует ошибку 1098.
    * Вместо этого вы можете предпочесть использовать
    * COMReturnError().
ENDIF

RETURN

```

Поскольку каждый метод, который взаимодействует с пользователем, вызывает TellUser, было бы довольно легко заменить его более сложным механизмом обмена сообщениями. Одним из очевидных изменений является предложение опции «тихого обмена сообщениями», возвращающего коды ошибок вместо отображения сообщений.

Создание экземпляра сервера

Вероятно, самая сложная и подверженная ошибкам часть Автоматизации – запуск сервера. Возможно, сервер отсутствует или не зарегистрирован, либо Windows по какой-то причине не может его запустить. Чтобы справиться со всеми этими возможностями, cusWrapServer использует отдельный класс, единственной целью которого является создание экземпляра сервера Автоматизации. Этот класс, называемый cusOpenServer, также является производным от Custom. Хотя методы cusOpenServer можно включить в cusWrapServer, имеет смысл вынести их в отдельный класс, поскольку многим процессам необходимо открыть сервер Автоматизации. Вместо того, чтобы заставлять их всех загружать весь абстрактный класс-оболочку, который имеет множество методов, не связанных с задачей открытия сервера, они могут использовать только cusOpenServer.

cusOpenServer имеет два настраиваемых свойства и два настраиваемых метода. Свойство cServerName содержит имя сервера, экземпляр которого будет создан; Свойство oServer содержит ссылку на вновь созданный сервер. Метод OpenServer выполняет фактическую работу по созданию сервера. Метод zReadMe класса является методом документации (как и во всех представленных здесь классах).

Метод Init cusOpenServer принимает два параметра: имя сервера для создания экземпляра и логическое значение, указывающее, следует ли создавать его экземпляр сразу или подождать, пока OpenServer не будет вызван явно. Затем он использует класс реестра (предоставляемый как VCX с VFP 6 и как PRG с VFP 5), чтобы убедиться, что сервер зарегистрирован. В противном случае Init возвращает .F., что предотвращает создание этого объекта (cusOpenServer). Вот код:

```

* cusOpenServer.Init
* Проверьте, допустимо ли указанное имя сервера, и,
* если указано пользователем, создайте сервер.
LPARAMETERS cServerName, lCreateAtInit

LOCAL oRegistry

IF VarType(cServerName) = "C"

```



```

* Проверьте, является ли это действительным именем сервера.
* При разработке/тестировании вам может потребоваться добавить
* Registry.VCX из HOME()+"\FFC\" в ваш путь
oRegistry = NewObject("Registry","Registry")
IF oRegistry.IsKey(cServerName)
    This.cServerName = cServerName
ELSE
    RETURN .F.
ENDIF
RELEASE oRegistry
ELSE
    * Нет смысла создавать этот объект.
    RETURN .F.
ENDIF

IF VarType(lCreateAtInit) = "L" AND lCreateAtInit
    This.OpenServer()
ENDIF

RETURN

```

Метод `OpenServer` класса `cusOpenServer` создает экземпляр сервера и сохраняет ссылку. Вот код:

```

* cusOpenServer.OpenServer
* Откройте сервер, указанный This.cServerName.
* Сохраните новый экземпляр в This.oServer.

IF VARTYPE(This.cServerName) = "C"
    This.oServer = CreateObject(This.cServerName)
ENDIF

RETURN This.oServer

```

На самом деле есть и другие проблемы, которые могут возникнуть при попытке создать экземпляр сервера. Они обсуждаются в главе 14 «Обработка ошибок Автоматизации». Чтобы облегчить чтение этого кода, мы не рассмотрели здесь все из них. Если вы используете этот класс в качестве основы для своего собственного класса-оболочки, обязательно интегрируйте код из главы 14 в этот метод `Init`.

Метод `OpenServer` класса `cusWrapServer` создает экземпляр `cusOpenServer`, передавая `.F.` для параметра `lCreateAtInit`. Если экземпляр `cusOpenServer` создан успешно, `OpenServer` получает ссылку на новый сервер и сохраняет ее в свойстве `oServer`. В противном случае пользователю сообщают, что сервер не может быть запущен.

```

* cusWrapServer.OpenServer
* Создайте экземпляр сервера Автоматизации
* и сохраните ссылку на него в This.oServer.

LOCAL oGetServ

IF EMPTY(This.cServerName)
    This.TellUser("No server specified")
    RETURN .F.
ENDIF

```

```
oGetServ =  
NewObject("cusOpenServer","Automation","",This.cServerName,.F.)  
IF VarType(oGetServ) = "O"  
    This.oServer = oGetServ.OpenServer()  
    IF VarType(This.oServer) <> "O"  
        This.TellUser("Can't create automation server")  
        RETURN .F.  
    ENDIF  
ENDIF  
  
RETURN
```

Заккрытие сервера

В отличие от создания экземпляров серверов, методы выключения разных серверов различаются. В частности, вам необходимо предпринять различные действия, чтобы быть уверенным, что пользователь не увидит диалоговое окно при попытке закрыть серверное приложение. Таким образом, метод CloseServer остается пустым (абстрактным) на уровне cusWrapServer и должен быть закодирован в каждом конкретном подклассе.

Однако мы хотим быть уверены, что серверные приложения не останутся работающими после закрытия нашего класса-оболочки. Итак, метод Destroy содержит такой код:

```
* cusWrapServer.Destroy  
* При необходимости закройте сервер на выходе,  
* чтобы он не работал в фоновом режиме.  
  
IF This.IsServerOpen()  
    This.CloseServer()  
ENDIF  
  
RETURN
```

Открытие документа

Термин «документ» в Windows используется для обозначения основного типа файла, с которым работает приложение. В Excel это книга, а в PowerPoint – презентация. Word запутывает проблему, поскольку его основной тип файла также называется документом. В оставшейся части этой главы термин «документ» используется в смысле Windows, а не в смысле файла Word.

Большая часть того, что вам нужно сделать, чтобы открыть документ, одинакова независимо от приложения. Вам необходимо убедиться, что пользователь указал документ для открытия, убедиться, что документ существует, и убедиться, что сервер открыт или, если необходимо, открыть сервер. Однако фактический процесс открытия документа различается для разных серверов. Класс-оболочка делит код на эти две части. Метод OpenDocument делает все возможное, а затем вызывает метод OpenServerDocument, который закодирован на уровне подкласса. Вот код OpenDocument:

```
* cusWrapServer.OpenDocument  
* Откройте указанный документ и сделайте его активным.  
* Возвращает логическое значение, указывающее,  
* был ли документ успешно открыт.
```

```

LPARAMETER cDocument
* cDocument = имя документа, который необходимо открыть, включая
ПОЛНЫЙ ПУТЬ

LOCAL lIsServerOpen, lSuccess

* Проверьте, был ли передан и действителен ли параметр
IF VarType(cDocument) <> "C"
    This.TellUser("Для открытия необходимо указать документ")
    RETURN .F.
ENDIF

IF NOT FILE(cDocument)
    This.TellUser("Не могу найти " + cDocument)
    RETURN .F.
ENDIF

* Убедитесь, что сервер открыт.
lIsServerOpen = This.IsServerOpen()

DO CASE
CASE NOT lIsServerOpen AND NOT This.lOpenAsNeeded
    This.TellUser("Программа недоступна")
    RETURN .F.

CASE NOT lIsServerOpen
    * Открыть сервер.
    This.OpenServer()

OTHERWISE
    * Сервер открыт и все в порядке.
ENDCASE

* Попытка открыть документ.
* Вызов метода, указанного сервером.
lSuccess = This.OpenServerDocument( cDocument )

RETURN lSuccess

```

Сохранение документа

Методы сохранения документов на серверах Office настолько схожи, что метод SaveDocument можно определить в абстрактном классе. Этот метод принимает два параметра: ссылку на объект сохраняемого документа и имя файла, включая полный путь, по которому его необходимо сохранить. Оба параметра являются необязательными. Если ссылка на документ опущена, активный документ сохраняется. Если имя файла опущено, файл повторно сохраняется в своем текущем местоположении. Как написано, метод перезаписывает существующие файлы, не информируя об этом пользователя.

```

* cusWrapServer.SaveDocument
* Сохраните указанный документ. Если передано имя файла,
* сохраните его с этим именем. В противном случае

```

```
* используйте имя, которое у него уже есть. Если это новый
* документ и имя не указано, произойдет сбой.
* НЕ ПРОВЕРЯЕТ ПЕРЕЗАПИСЬ ФАЙЛА!
```

```
* Если документ не передан, используйте активный документ.
```

```
LPARAMETERS oDocument, cFileName
```

```
LOCAL lReturn, cOldCentury
```

```
IF VarType(oDocument) <> "O"
    oDocument = This.oDocument
ENDIF
```

```
DO CASE
```

```
CASE IsNull(oDocument)
```

```
    * Нет документа для сохранения.
```

```
    * Сообщите пользователю и выйдите.
```

```
    This.TellUser("Нет документа для сохранения.")
```

```
    lReturn = .F.
```

```
CASE VarType(cFileName) = "C"
```

```
    * Просто сохраните его с указанным именем.
```

```
    lReturn = .T.
```

```
    oDocument.SaveAs( cFileName )
```

```
    tFDateTime = This.GetFileDateTime( cFileName)
```

```
    IF NOT FILE(cFileName) OR NOT (tFDateTime + 1 > DATETIME())
```

```
        * Если файл не существует или не был обновлен за последнюю
        * секунду
```

```
        This.TellUser("Не удалось сохранить документ как " + ;
                      cFileName)
```

```
        lReturn = .F.
```

```
    ENDIF
```

```
CASE VarType(cFileName) = "L" AND ;
```

```
    NOT (oDocument.Name == oDocument.FullName)
```

```
    * Имя отсутствует, но сохранено ранее, поэтому сохраните его
```

```
    * с текущим именем.
```

```
    lReturn = .T.
```

```
    oDocument.Save()
```

```
    cFileName = oDocument.FullName
```

```
    tFDateTime = This.GetFileDateTime( cFileName)
```

```
    IF NOT (tFDateTime + 1 > DATETIME())
```

```
        * Если файл не был обновлен за последнюю секунду.
```

```
        This.TellUser("Не удалось сохранить документ.")
```

```
        lReturn = .F.
```

```
    ENDIF
```

```
OTHERWISE
```

```
    This.TellUser("Укажите имя файла")
```

```
lReturn = .F.
```

```
ENDCASE
```

```
RETURN lReturn
```

Этот метод работает для Word и PowerPoint. Excel не позволяет использовать SaveAs для перезаписи существующего файла без запроса пользователя, поэтому подкласс, специфичный для Excel, содержит собственную версию SaveDocument, которая позволяет обойти эту проблему.

Используемый здесь метод GetFileDateTime возвращает время последнего сохранения файла в виде значения даты и времени. Мы были удивлены, что нам пришлось прибегнуть к этому обходному методу проверки успешного сохранения, но, что удивительно, методы Save и SaveAs не возвращают значение, указывающее, удалось ли им сохранить файл. В сети этот тест может завершиться неудачей, даже если файл был сохранен, если часы на сервере не синхронизированы с часами на локальном компьютере.

Другие основные операции

Хотя существуют и другие методы, которые явно необходимы для каждого сервера, такие как печать или закрытие серверного приложения, их необходимо определять в подклассах, поскольку они зависят от конкретных свойств и методов серверов, или способ их выполнения слишком отличается от одного сервера к другому.

15.4 Создание подклассов

Следующим шагом в создании классов-оболочек является создание конкретных подклассов cusWrapServer. Мы хотим, чтобы для каждого сервера приложений был обернут один подкласс. Для Office мы рассмотрим три подкласса: cusWord, cusExcel и cusPowerPoint. Все три наследуют пользовательские свойства и методы cusWrapServer, а также код, добавленный на абстрактном уровне.

На уровне подкласса мы можем иметь дело с особенностями отдельных продуктов, а также с обычными различиями между ними. Первым шагом является завершение определения основных операций.

Каждый из подклассов имеет включаемый файл, в котором #DEFINE определяет соответствующие константы. В отличие от другого кода в этой книге, следующий код не показывает определения констант каждый раз, когда они используются.

Каждый из классов Excel и PowerPoint имеет по одному дополнительному свойству, которое можно добавить в базовые операции. Он называется lShouldBeVisible и отслеживает видимость сервера. Он используется для определения того, есть ли ссылка на сервер или нет. См. раздел «Проверка сервера» далее в этой главе.

Закрытие серверов

Самая большая проблема при отключении серверов Автоматизации заключается в том, что они не отображают какие-либо сообщения пользователю, поскольку файлы не сохранены. Существуют различные способы решения проблемы: представленные здесь версии закрывают файлы без их сохранения. Другие альтернативы – автоматически сохранять все открытые документы или позволить вам передать параметр CloseServer, чтобы решить, сохранять ли открытые документы или запрашивать у пользователя каждый открытый документ, но делать это на стороне VFP, а не на стороне сервера.

Чтобы завершить работу без сохранения открытых документов, необходимо

несколько разных подходов. В Word – самый простой способ. Quit принимает параметр, который сообщает ему о закрытии без сохранения изменений. Вот метод CloseServer для класса cusWord:

```
* cusWord.CloseServer
* Закройте сервер Автоматизации Word.

LOCAL lReturn

IF This.IsServerOpen()
    This.oServer.Quit( wdDoNotSaveChanges )
    This.oServer = .NULL.
    This.oDocument = .NULL.
    lReturn = .T.
ELSE
    * Мог бы сказать пользователю, но зачем?
    * This.TellUser("Word не открыт.")
    lReturn = .F.
ENDIF

RETURN lReturn
```

И в Excel, и в PowerPoint вам придется сделать это с трудом: вам придется просмотреть все открытые документы и закрыть их, прежде чем вы сможете вызвать команду «Выход». Вот код метода CloseServer Excel. PowerPoint – это почти то же самое, просто мысленно замените на презентацию везде, где вы видите рабочую книгу.

```
* cusExcel.CloseServer
* Закройте сервер Автоматизации Excel.

LOCAL lReturn, oWorkbook

IF This.IsServerOpen()
    * Сначала закройте открытые книги.
    FOR EACH oWorkbook IN This.oServer.Workbooks
        This.CloseDocument( oWorkbook.Name )
    ENDFOR
    This.oServer.Quit()
    This.oServer = .NULL.
    This.oDocument = .NULL.
    This.lShouldBeVisible = .F.
    lReturn = .T.
ELSE
    * Мог бы сказать пользователю, но зачем?
    * This.TellUser("Excel не открыт.")
    lReturn = .F.
ENDIF

RETURN lReturn
```

Обратите внимание на использование здесь метода CloseDocument. Без этого вызова CloseServer был бы намного длиннее и сложнее, помимо прочего, CloseDocument скрывает детали закрытия книг без сохранения изменений или запроса пользователя.

Открытие документа, часть 2

Метод OpenDocument (описанный ранее в разделе «Открытие документа» в этой главе) обрабатывает общие аспекты получения указанного документа. Затем он вызывает метод OpenServerDocument для выполнения части задачи, специфичной для сервера. Каждый подкласс имеет код в этом методе. Код всех трех классов очень похож, различаясь в первую очередь именами объектов, с которыми он работает. Вот версия PowerPoint:

```
* cusPowerPoint.OpenServerDocument
* Этот метод выполняет часть открытия документа,
* специфичную для PowerPoint. Он вызывается OpenDocument
* после того, как этот метод проверяет действительный
* сервер, допустимое имя файла и т. д.

LPARAMETER cDocument

* Попытка открыть документ.
This.oDocument = This.oServer.Presentations.Open(cDocument)
IF UPPER(This.oDocument.FullName) <> UPPER(cDocument)
    This.TellUser("Невозможно открыть " + cDocument)
    RETURN .F.
ENDIF

RETURN .T.
```

Заккрытие документа

Когда мы готовы закрыть документ, возникают самые разные вопросы. Какой документ нам следует закрыть? Документ сохранился? Стоит ли его сохранять? Методы Close серверов Office имеют различную готовность обрабатывать несохраненные изменения без запроса пользователя. Однако независимо от желания приложения цель наших методов CloseDocument – избежать всех запросов и выполнить пожелания пользователя.

Реализованные нами методы CloseDocument принимают три параметра, причем все они необязательны. Первый – это имя закрываемого документа; если более одного открытого документа могут иметь одно и то же имя, путь должен быть включен. Второй параметр является логическим и указывает, нужно ли сохранять документ перед его закрытием; в противном случае изменения отбрасываются. Третий параметр – это имя файла, включая путь, который будет присвоен документу при его сохранении. (Вы можете передать третий параметр, чтобы сохранить существующий документ в новом месте перед его закрытием.)

Как и в случае со многими методами, определенными на этом уровне, большая часть различий между классами заключается просто в том, что вы имеете дело с разными объектами. Вот версия CloseDocument для Word:

```
* cusWord.CloseDocument
* Закройте документ в Word. Если передано имя документа,
* закройте его; в противном случае закройте активный документ.
* Сохраните документ перед закрытием,
* если lSaveFirst имеет значение .t.

LPARAMETERS cDocument, lSaveFirst, cFileName
* cDocument = название документа, который нужно закрыть.
*           Включите путь, если несколько документов
```



```
*          могут иметь одну и ту же основу.
* lSaveFirst = сохранить документ перед закрытием?
* cFileName = имя файла, которое нужно назначить при сохранении

LOCAL oDocToClose && документ, который нужно закрыть

* Проверить параметры.
IF NOT INLIST(VarType(cDocument), "C", "L")
    This.TellUser("Укажите имя документа или оставьте " + ;
                  "пустым для активного документа")
    RETURN .F.
ENDIF

IF VarType(lSaveFirst) <> "L"
    This.TellUser("Укажите .Т. чтобы сохранить перед закрытием; "+
                  ".F. или опустите, чтобы закрыть без сохранения")
    RETURN .F.
ENDIF

* Убедитесь, что Word открыт и существует хотя бы один документ.
IF NOT This.IsServerOpen()
    This.TellUser("Word недоступен.")
    RETURN .F.
ENDIF

IF NOT This.IsDocOpen()
    This.TellUser("Нет открытых документов.")
    RETURN .F.
ENDIF

* Пользователь указал документ?
IF VarType(cDocument) = "C"
    * Проверьте наличие соответствующего документа.
    * Здесь необходимо использовать TYPE() вместо VARTYPE(),
    * чтобы избежать вычисления выражения.
    IF TYPE("This.oServer.Documents[cDocument]") = "O"
        oDocToClose = This.oServer.Documents[cDocument]
    ELSE
        This.TellUser(cDocument + " не открыт.")
        RETURN .F.
    ENDIF
ELSE
    oDocToClose = This.oDocument
ENDIF

lSuccess = .T.
IF lSaveFirst
    lSuccess = This.SaveDocument(oDocToClose, cFileName)
ENDIF

IF lSuccess
    * Не закрываться, если запрос на сохранение не выполнен.
    oDocToClose.Close(wdDoNotSaveChanges)
```

```
ENDIF
```

```
RETURN lSuccess
```

Версии Excel и PowerPoint очень похожи; Подробности см. в файлах для загрузки для разработчиков, доступных по адресу www.hentzenwerke.com.

Создание новых документов

Одной из наиболее распространенных операций является создание нового документа любого типа, который использует сервер. Как и последние шаги по открытию документа, это необходимо сделать в подклассах, поскольку детали зависят от иерархии объектов.

И Word, и Excel позволяют указать шаблон, на котором будет основан новый документ, поэтому методы в этих подклассах принимают один необязательный параметр, содержащий имя используемого шаблона. Если он опущен, используется шаблон по умолчанию. Код метода NewDocument для этих двух классов сложен, поскольку он ищет указанный шаблон, чтобы убедиться в его доступности, прежде чем пытаться создать новый документ. Для этого он использует класс cusDirectoryHandler из библиотеки Utilities.VCX (которая включена в файлы загрузки для разработчиков, доступные по адресу www.hentzenwerke.com). Вот версия Excel:

```
* cusExcel.NewDocument
* Создайте новую книгу Excel. Используйте шаблон, если он указан.
LPARAMETERS cTemplate
    * cTemplate - имя шаблона новой книги

LOCAL lFoundIt, cTemplatePath, oDirectoryHandler, cPath
LOCAL lSuccess, lIsServerOpen
    * lFoundIt = указанный шаблон был где-то найден?
    * cTemplatePath = путь к каталогу, указанному Excel для
    *                  шаблонов
    * oDirectoryHandler = ссылка на объект обработки каталога
    * cPath = путь для проверки шаблона
    * lSuccess = возвращаемое значение, указывающее, был ли создан
    *             новый документ.
    * lIsServerOpen = Excel уже открыт?

* Проверить параметр.
IF NOT INLIST(VarType(cTemplate), "C", "L")
    This.TellUser("Укажите имя шаблона или опустите, " + ;
        "чтобы использовать шаблон по умолчанию")

    RETURN .F.
ENDIF

* Убедитесь, что Excel открыт.
lIsServerOpen = This.IsServerOpen()

DO CASE
CASE NOT lIsServerOpen AND NOT This.lOpenAsNeeded
    This.TellUser("Excel недоступен.")
    RETURN .F.

CASE NOT lIsServerOpen
    * Откройте Excel.
```

```
This.OpenServer()
```

```
OTHERWISE
```

```
    * Excel открыт и все в порядке.
```

```
ENDCASE
```

```
IF EMPTY(cTemplate)
```

```
    * Если шаблон не указан, используйте шаблон по умолчанию.  
    * Для этого мы можем просто установить cTemplate в пустую  
    * строку и установить флаг, говорящий, что мы закончили.  
    * Тогда позвольте вызову ниже выполнить всю работу.
```

```
This.oDocument = This.oServer.Workbooks.Add()
```

```
lSuccess = .T.
```

```
ELSE
```

```
    * Существует ли указанный шаблон?  
    * Есть несколько возможностей.  
    * 1) Если вызывающий абонент предоставил полный путь и файл,  
    *    проверьте это и не более.  
    * 2) Вызывающий абонент предоставил относительный путь и  
    *    файл.  
    *    Нам нужно его почистить и проверить.  
    * 3) Вызывающий абонент предоставил только имя файла.  
    *    Нам нужно проверить его в соответствующих каталогах  
    *    шаблонов Excel.  
    * Первые два можно объединить.
```

```
IF NOT EMPTY(JustPath(cTemplate))
```

```
    * Был предусмотрен какой-то путь,  
    * так что мы можем просто пойти туда.  
    cTemplate = FULLPATH(cTemplate)  
    lFoundIt = FILE(cTemplate)
```

```
ELSE
```

```
    * Сначала попробуйте каталог шаблонов  
    * пользователя (и его подкаталоги).  
    oDirectoryHandler = NewObject("cusDirectoryHandler", ;  
                                   "Utilities")  
    cTemplatePath = This.oServer.TemplatesPath  
    IF EMPTY(cTemplatePath)  
        lFoundIt = .F.  
    ELSE  
        IF FILE(ForcePath(cTemplate, cTemplatePath))  
            lFoundIt = .T.  
            cTemplate=ForcePath(cTemplate, cTemplatePath)  
        ELSE  
            * Нужно спуститься по дереву каталогов ниже.  
            nDirectoryCount = ;  
                oDirectoryHandler.BuildDirectoryTree(cTemplatePath)  
            nDir = 1  
            DO WHILE NOT lFoundIt AND nDir <= nDirectoryCount  
                cPath = oDirectoryHandler.GetDirectory(nDir)  
                IF FILE(ForcePath(cTemplate, cPath))
```

```

        lFoundIt = .T.
        cTemplate = ForcePath(cTemplate, cPath)
    ENDIF
    nDir = nDir + 1
ENDDO
oDirectoryHandler.ClearDirectoryTree()
ENDIF

* Если все еще не найдено, попробуйте путь к шаблону
* группы.
IF NOT lFoundIt
    cTemplatePath = This.oServer.NetworkTemplatesPath
    IF EMPTY(cTemplatePath)
        lFoundIt = .F.
    ELSE
        IF FILE(ForcePath(cTemplate, cTemplatePath))
            lFoundIt = .T.
            cTemplate=ForcePath(cTemplate, cTemplatePath)
        ELSE
            * Нужно спуститься по дереву каталогов ниже.
            nDirectoryCount = ;
            oDirectoryHandler.BuildDirectoryTree( ;
                cTemplatePath)
            nDir = 1
            DO WHILE NOT lFoundIt AND ;
                nDir <= nDirectoryCount
                cPath = oDirectoryHandler.GetDirectory(nDir)
                IF FILE(ForcePath(cTemplate, cPath))
                    lFoundIt = .T.
                    cTemplate = ForcePath(cTemplate, cPath)
                ENDIF
                nDir = nDir + 1
            ENDDO
        ENDIF
    ENDIF
ENDIF

ENDIF
ENDIF

* Освободите обработчик каталога. Мы закончили с этим.
RELEASE oDirectoryHandler
ENDIF

* Итак, мы нашли где-нибудь шаблон?
IF lFoundIt
    * Создайте новый документ.
    This.oDocument = This.oServer.Workbooks.Add( cTemplate )
    lSuccess = .T.
ELSE
    This.TellUser("Невозможно найти указанный шаблон.")
    lSuccess = .F.
ENDIF

```

```
ENDIF
```

```
RETURN lSuccess
```

Версия NewDocument для Word очень похожа на показанную здесь. Версия PowerPoint намного проще, поскольку PowerPoint не предлагает возможности указать шаблон, поэтому она просто создает новую презентацию и возвращает индикатор успеха.

Проверка сервера

Многим методам оболочки необходимо убедиться в доступности сервера. Метод IsServerOpen проверяет состояние сервера и возвращает .T. если он открыт и .F. если это не так.

Для проверки состояния сервера необходимы две совершенно разные стратегии, поскольку Word при выключении ведет себя совершенно иначе, чем Excel и PowerPoint. Когда Word закрывается либо путем вызова метода Quit, либо из-за того, что пользователь закрывает его в интерактивном режиме, он исчезает из диспетчера задач, а ссылки на объект приложения Word вызывают ошибки. Итак, чтобы определить, относится ли свойство oServer к допустимому экземпляру Word, мы можем использовать этот код:

```
* cusWord.IsServerOpen
* Проверьте, открыт ли Word

RETURN VarType(This.oServer)<>"0" OR ;
       TYPE("This.oServer.Name")<>"C"
```

Однако PowerPoint и Excel сохраняют объект Application доступным до тех пор, пока все ссылки на них не будут уничтожены. Кроме того, даже несмотря на то, что сервер закрыт, вы можете продолжать ссылаться на его свойства и методы через ссылочную переменную (в данном случае свойство oServer), не получая при этом ошибок. Поэтому необходима другая стратегия.

Наш подход состоит из двух частей. Во-первых, метод CloseServer устанавливает oServer в значение .NULL. после команды Quit, чтобы серверное приложение фактически закрылось. Это обрабатывает случаи, когда сервер закрывается через класс-оболочку. Более сложный случай – это случай, когда пользователь закрывает приложение в интерактивном режиме. Чтобы справиться с этой ситуацией, свойство lShouldBeVisible отслеживает ожидаемую видимость сервера. Оно установлено на .F. при открытии сервера, а затем устанавливается в соответствии с видимостью сервера в методах ShowServer и HideServer (см. «Отображение сервера» далее в этой главе). Когда пользователь закрывает сервер через интерфейс, свойство Visible сервера изменяется, а lShouldBeVisible – нет. Таким образом, IsServerOpen может проверить несоответствие между этими двумя свойствами. Код Excel и PowerPoint немного отличается из-за способа обработки логических значений, но идея идентична. Вот версия Excel:

```
* cusExcel.IsServerOpen
* Проверьте, открыт ли сервер.
* Этот метод сравнивает пользовательское свойство
* lShouldBeVisible с собственным свойством Visible сервера.
* Если они совпадают, сервер открыт. Если нет, то пользователь
* должен закрыть сервер в интерактивном режиме. Причина, по
* которой необходим этот метод, заключается в том, что Excel
* сохраняет сервер в памяти даже после его выключения.
```

```

LOCAL lReturn

IF IsNull(This.oServer)
    * Нет созданного сервера
    lReturn = .F.
ELSE
    * Сравните фактическое значение видимости с отслеживаемой
    * видимостью.
    IF This.oServer.Visible = This.lShouldBeVisible
        * Они совпадают, значит сервер открыт и исправен.
        lReturn = .T.
    ELSE
        * Видимость не соответствует. Пользователь должен отключить
        * сервер.
        lReturn = .F.
    ENDIF
ENDIF

RETURN lReturn

```

Чтобы это работало, в методе OpenServer необходим еще один шаг: свойство lShouldBeVisible должно быть установлено в .F. Итак, для подклассов Excel и PowerPoint OpenServer содержит такой код:

```

* cusExcel.OpenServer / cusPowerPoint.OpenServer
DoDefault()

* Если открытие сервера прошло успешно,
* установите отслеживаемую видимость.
IF NOT IsNull(This.oServer)
    This.lShouldBeVisible = .F.
ENDIF

```

Проверка открытых документов

Точно так же, как нам нужно знать, есть ли ссылка на сервер, многим методам необходимо знать, открыт ли документ. Этой цели служит метод IsDocOpen. Он закодирован на уровне подкласса из-за различий в иерархии объектов, но по сути идентичен в каждом случае. Вот версия PowerPoint:

```

* cusPowerPoint.IsDocOpen
* Проверка, открыта ли презентация в PowerPoint
* Этому методу всегда должен предшествовать вызов IsServerOpen.

RETURN This.oServer.Presentations.Count > 1

```

Отслеживание активного документа

Ряд методов класса-оболочки работают с активным документом (независимо от того, означает ли «документ» книгу, презентацию или документ для сервера). Важно, чтобы у класса-оболочки всегда была возможность ссылаться на активный документ сервера, какого бы типа он ни был. Поскольку имя свойства сервера для этого объекта варьируется в зависимости от того, какой сервер вы используете, необходимость ссылаться на это свойство сервера каждый раз, когда оно вам нужно, неудобно.

Вместо доступа к активному документу через свойство сервера cusWrapper

использует свойство oDocument. Чтобы поддерживать актуальность, метод Access этого свойства содержит код на уровне подкласса. (Поскольку методы доступа были представлены в VFP 6, этот метод нелегко перенести в VFP 5. Если вы используете эту версию, вы должны самостоятельно отслеживать активный документ. Хотя это и обременительно, это не представляет проблемы, если только пользователю не разрешено взаимодействовать с экземпляром сервера Автоматизации.)

Код oDocument_Access очень похож для каждого подкласса, различается только то, на какое свойство сервера он ссылается. Вот версия oDocument_Access для Excel:

```
* oDocument_Access
* Убедитесь, что свойство указывает на активный в данный момент
* документ.
IF This.IsServerOpen() AND This.IsDocOpen()
    This.oDocument = This.oServer.ActiveWorkbook
ELSE
    This.oDocument = .NULL.
ENDIF

RETURN This.oDocument
```



PowerPoint распознает активные объекты только в том случае, если сам PowerPoint видим, поэтому код, аналогичный показанному здесь, не работает, когда PowerPoint.Visible = .F. Эта небольшая досадная разница может действительно повлиять на ваш код. Дополнительную информацию см. в разделе «Обертка PowerPoint – стандартизация текста» далее в этой главе.

Отображение сервера

Методы ShowServer и HideServer позволяют отображать и скрывать сервер. Между версиями Excel и Word есть небольшие различия, а версия ShowServer для PowerPoint идентична версии Excel. Однако когда дело доходит до HideServer и PowerPoint, существует большая разница. Это потому, что как только вы сделаете PowerPoint видимым, вы не сможете снова его скрыть. PowerPoint этого не позволяет.

Единственная разница между версиями этих методов для Word и Excel заключается в том, что версия Excel отслеживает пользовательское свойство lShouldBeVisible, необходимое для метода IsServerOpen. Вот версии обоих методов в Excel:

```
* cusExcel.ShowServer
* Покажите сеанс Автоматизации Excel.

LOCAL lReturn
* lReturn = возвращаемое значение -
* .T. если Excel открыт; .F. в противном случае

* Excel открыт?
IF This.IsServerOpen()
    This.oServer.Visible = .T.
    This.lShouldBeVisible = .T.
    lReturn = .T.
ELSE
    lReturn = .F.
ENDIF
```



```

RETURN lReturn

* cusExcel.HideServer
* Скройте сеанс Автоматизации Excel.

* Excel открыт?
IF This.IsServerOpen()
    This.oServer.Visible = .F.
    This.lShouldBeVisible = .F.
ENDIF

RETURN

```

15.5 Использование основных операций



Определив все эти основные операции, мы можем использовать их вместо прямого вызова серверов. В листинге 1 показана программа, которая открывает Word, создает новый документ, помещает в него небольшой текст, сохраняет его и закрывает Word. Программа делает все за кулисами, вообще без пользовательского интерфейса (кроме звука работающего жесткого диска). Чтобы проверить результаты, откройте Word и файл Sample.DOC (хранящийся в каталоге временных файлов, указанном SYS(2023)). Программа включена в файлы для загрузки для разработчиков, доступные по адресу www.henzenwerke.com как UseWordWrapper.PRG.

Листинг 1. Использование класса-обертки. Эта программа использует класс-оболочку Word для открытия Word, создания документа и его сохранения. Единственный код в этой программе, который напрямую взаимодействует с Word, — это две строки, которые фактически помещают текст в документ. Все остальное проходит через класс-оболочку.

```

* UseWordWrapper.PRG
* © 2000, Tamar E. Granor and Della Martin
* From: Microsoft Office Automation with Visual FoxPro
* Hentzenwerke Publishing. www.hentzenwerke.com

```

```

* Демонстрация класса-оболочки Word.
#DEFINE CR CHR(13)

```

```

LOCAL oWord

```

```

* Открыть Word.
oWord = NewObject("cusWord","automation")

```

```

WITH oWord
    .OpenServer()

```

```

    IF NOT .IsServerOpen()
        WAIT WINDOW "Не удалось открыть Word."
        RETURN .F.
    ENDIF

```

```

* Создайте документ
.NewDocument()

* Проверьте, открыт ли документ.
IF .IsDocOpen()
    * Поместите некий текст в новый документ
    WITH .oDocument.Range()
        .InsertAfter("Это наш новый документ Word." + CR)
        .InsertAfter("Только эти строки напрямую ссылаются" + ;
            " на методы Word." + CR)
    ENDWITH

    * Сохраните новый документ
    .SaveDocument( ,FULLPATH("") + "sample.doc")

ELSE
    WAIT WINDOW "Не удалось создать новый документ."
ENDIF

* Закройте Word.
.CloseServer( )
ENDWITH

RETURN

```

Поскольку `OpenServer` и `NewDocument` возвращают логические значения, указывающие на успех или неудачу, этот код на самом деле можно несколько сократить. Весь процесс открытия сервера и создания нового документа можно свести к одной строке:

```
IF .OpenServer() and .NewDocument()
```

Мы не сделали этого по двум причинам. Во-первых, мы хотели продемонстрировать больше методов. Фактически, в более длинном примере вы, скорее всего, довольно часто будете использовать `IsServerOpen` и, особенно, `IsDocOpen`. Во-вторых, объединение этих двух методов в одну строку не позволяет вам выдавать конкретные сообщения об ошибках. Все, что вы знаете, это то, что процесс не удался. Поскольку класс-оболочка генерирует сообщения об ошибках, этого может быть достаточно. (На самом деле, если ваша версия класса-оболочки предоставляет сообщения от `TellUser`, возможно, вы не захотите включать какие-либо сообщения об ошибках в программу, использующую этот класс.)

Обратите внимание, что переменная `oWord` здесь относится не к самому объекту `Word Automation`, как в примерах предыдущих глав, а к объекту-оболочке `cusWord`.

Хотя этот пример не кажется намного короче, чем примеры из предыдущих глав, поймите, что большая разница состоит в том, что эта версия включает в себя множество скрытых операций обработки ошибок.

15.6 Добавление операций, специфичных для сервера

До сих пор все определенные нами методы были общими для всех серверов. Хотя они и облегчают написание кода Автоматизации, это только начало. Следующим шагом является перенос операций, специфичных для отдельных серверов.

Но на серверах Office есть десятки методов. Как узнать, какие из них следует обернуть? Это легко – оберните те, которые вы используете. Будьте внимательны,

когда пишете код Автоматизации. Имейте в виду, что цель переноса серверов – сделать вашу жизнь проще. Подумайте о том, чего вы пытаетесь достичь, используя тот или иной метод сервера.

Во второй раз, когда вы обнаружите, что выполняете определенную задачу, пришло время надеть на нее оболочку. Но только потому, что вы его обернули, не думайте, что вы с ним закончили. Когда вы увидите, как вы на самом деле его используете, вам, вероятно, придется его почистить. Среди вопросов, которые вам нужно задать, – какие параметры он должен иметь, нужны ли классу-оболочке какие-либо свойства для поддержки этого метода, действительно ли метод нужно разбить на несколько методов и следует ли сгруппировать несколько методов сервера в один метод-оболочку, потому что вы всегда будете использовать их вместе. В итоге вы хотите получить набор методов со значимыми именами и небольшим количеством параметров, каждый из которых выполняет одну задачу, но которые вы можете объединить для выполнения сложных операций.

Чтобы немного усложнить вашу работу, окажется, что ваши классы-оболочки, вероятно, должны будут работать по-разному для разных приложений. На это мы намекали ранее, когда говорили о методе `CloseServer` – в некоторых приложениях уместно закрывать все открытые документы, не сохраняя их; в других нужно сохранять каждый открытый документ; а в других этот выбор должен сделать пользователь. То же самое и с другими задачами, специфичными для сервера: набор необходимых вам операций и способ их работы различаются в зависимости от того, что вы пытаетесь выполнить.

К счастью, в объектно-ориентированной среде, такой как VFP, с такими вариациями легко справиться. Создание подклассов предоставляет один из способов сделать это, хотя это может привести к головной боли при обслуживании, если вам нужно слишком много вариантов. Альтернативой является использование свойств и параметров для удовлетворения потребностей конкретного приложения. Если конкретную операцию необходимо реализовать несколькими совершенно разными способами, вы можете даже захотеть использовать для ее обработки отдельный класс. Затем подкласс, специфичный для сервера, может определить подходящую реализацию во время выполнения и создать экземпляр необходимого класса. (Это шаблон *Strategy* («Стратегия») – подробности см. в *Паттерны объектно-ориентированного проектирования* Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж.)

Для начала приведем методы-оболочки для одной общей операции на каждом из серверов Office.

Обёртка Word - от таблицы к таблице

Как мы обсуждали в главе 5 «Промежуточный Word», таблицы Word удобны для отображения данных FoxPro. Одна из вещей, которые вы, скорее всего, сделаете при отправке данных VFP в Word, – это преобразуете какой-либо курсор VFP в таблицу Word. Здесь метод, `DBFToTable`, принимает данные в рабочей области, независимо от того, представляют ли они курсор, таблицу или представление, и создает и заполняет таблицу Word, содержащую эти данные. Метод принимает четыре параметра (все необязательные): диапазон, в котором должна быть размещена таблица, псевдоним курсора и два логических флага. Первый флаг указывает, следует ли оставлять пустую строку вверху таблицы для заголовков, а второй определяет, используется ли метод `AutoFit` для изменения размера таблицы после ее заполнения. Вот код:

```
* cusWord.DBFToTable
```

```
* Этот метод помещает данные из курсора, представления или
```

```
* таблицы в таблицу Word. Он создает одну строку таблицы
```

- * для каждой записи с одним столбцом для каждого поля.
- * Возвращает ссылку на вновь созданную таблицу;
- * .null. если таблица не может быть создана.

```

LPARAMETERS oRange, cAlias, lAddHeader, lAutoFit
* oRange - куда поместить новую таблицу. Если опущено,
*          добавить в текущей точке вставки.
* cAlias - псевдоним рабочей области, содержащей
*          курсор/представление/таблицу. Если опущено,
*          использовать текущую рабочую область..
* lAddHeader - включить строку заголовка в таблицу?
* lAutoFit - автоматически изменять размер столбцов?

LOCAL nOldSelect, nRows, nCols, oTable, nCurRow, nCol, nActualCol
* nOldSelect = рабочая область выбирается при вызове
* nRows      = количество добавляемых строк
* nCols      = количество добавляемых столбцов
* oTable     = содержит ссылку на вновь созданную таблицу
* nCurRow   = текущая строка, получающая данные
* nCol       = счетчик столбцов
* nActualCol = указывает, какой столбец вставить в таблицу

* Проверьте параметры
nOldSelect = SELECT()
IF EMPTY( cAlias )
    cAlias = ALIAS()
    IF EMPTY( cAlias )
        This.TellUser("Необходимо указать курсор.")
        RETURN .NULL.
    ENDIF
ELSE
    IF USED( cAlias )
        SELECT (cAlias)
    ELSE
        This.TellUser("Такой курсор не используется.")
        RETURN .NULL.
    ENDIF
ENDIF

IF VarType( lAddHeader ) <> "L"
    This.TellUser("Используйте логическое значение, чтобы " + ;
        "указать, следует ли добавлять строку"
заголовка.")
    SELECT (nOldSelect)
    RETURN .NULL.
ENDIF

IF VarType( lAutoFit ) <> "L"
    This.TellUser("Используйте логическое значение, чтобы " + ;
        "указать, следует ли изменять размер столбцов.")
    SELECT (nOldSelect)
    RETURN .NULL.
ENDIF

```

```

* Теперь убедитесь, что есть документ.
IF NOT (This.IsServerOpen() AND This.IsDocOpen())
    This.TellUser("Word должен быть открыт с готовым документом.")
    SELECT (nOldSelect)
    RETURN .NULL.
ENDIF

* Проверьте параметр oRange
IF VarType( oRange ) <> "O"
    oRange = This.oServer.Selection.Range()
ENDIF

* Готово к работе. Добавить таблицу в текущей точке вставки
WITH This.oDocument
    nRows = RECCOUNT()
    IF lAddHeader
        nRows = nRows + 1
    ENDIF

    * Определите количество столбцов, которые нужно создать.
    * Пропустите любые поля General.
    nColumns = 0
    FOR nCol = 1 TO FCOUNT()
        IF TYPE(FIELD(nCol)) <> "G"
            nColumns = nColumns + 1
        ENDIF
    ENDFOR
    oTable = .Tables.Add(oRange,nRows,nColumns)
ENDWITH

WITH oTable
    IF lAddHeader
        * Установите первую строку как строку заголовка,
        * которая будет повторяться на последующих страницах.
        .Rows(1).HeadingFormat = .T.
        nCurRow = 2
    ELSE
        nCurRow = 1
    ENDIF

    * Теперь добавьте данные в строки, пропуская поля General.
    SCAN
        nActualCol = 1
        FOR nCol = 1 TO .Columns.Count
            * Eliminate general fields
            DO WHILE TYPE(FIELD(nActualCol)) = "G"
                nActualCol = nActualCol + 1
            ENDDO

            .Cell[ nCurRow, nCol ].Range.InsertAfter( ;
                TRANSFORM(EVAL(FIELD(nActualCol))))
            nActualCol = nActualCol + 1
        ENDFOR
    ENDSCAN
ENDWITH

```

```
ENDFOR
```

```
nCurRow = nCurRow + 1
```

```
ENDSCAN
```

```
IF lAutoFit
```

```
.Columns.AutoFit()
```

```
ENDIF
```

```
ENDWITH
```

```
RETURN oTable
```



Сам код довольно прост. Что делает этот метод достойным написания, так это то, что он берет задачу, которую мы, вероятно, будем выполнять довольно часто, и превращает ее в один вызов. В листинге 2 представлена еще одна программа, использующая класс-оболочку Word. Как и программа в листинге 1, она создает сервер Автоматизации и открывает новый документ. Затем она создает курсор на основе таблицы «Поставщик» из TasTrade, отправляет эти данные в новый документ, затем сохраняет документ и закрывает сервер. Эта программа включена как BuildSupplierTable.PRG в файлы загрузки для разработчиков, доступные на сайте www.hentzenwerke.com. На рисунке 1 показана часть полученного документа.

Листинг 2. Использование метода, специфичного для Word. Эта программа создает таблицу Word с помощью одного вызова метода из класса-оболочки Word.

```
* BuildSupplierTable.PRG
* © 2000, Tamar E. Granor and Della Martin
* From: Microsoft Office Automation with Visual FoxPro
* Hentzenwerke Publishing. www.hentzenwerke.com
```

```
* Создайте таблицу Word, используя класс-оболочку.
#DEFINE CR CHR(13)
```

```
LOCAL oWord
```

```
* Откройте Word.
```

```
oWord = NewObject("cusWord", "automation")
```

```
WITH oWord
```

```
.OpenServer()
```

```
IF NOT .IsServerOpen()
```

```
WAIT WINDOW "Не удалось открыть Word."
```

```
RETURN .F.
```

```
ENDIF
```

```
* Создайте документ.
```

```
.NewDocument()
```

```
* Проверьте, открыт ли документ.
```

```
IF .IsDocOpen()
```

```
* Поместите некий текст в новый документ
```

```

OPEN DATABASE _SAMPLES + "TasTrade\Data\TasTrade"
SELECT Company_Name, Contact_Name, Contact_Title, Phone ;
FROM Supplier ;
ORDER BY 1 ;
INTO CURSOR SupplierList

```

```

oRange = .oDocument.Range()
.DBFToTable(oRange, ,.F., .T.)

```

```

USE IN Supplier
USE IN SupplierList
ENDIF

```

```

* Сохраните новый документ.
.SaveDocument( ,FULLPATH("") + "suppliers.doc")

```

```

* Закройте Word
.CloseServer()

```

```

ENDWITH

```

```

RETURN

```

Aux joyeux ecclésiastiques	Guylene Medier	Sales Manager	(1) 03.03.00.68
Bigfoot Breweries	Cheryl Saylor	Regional Account Rep.	(503) 555 9931
Cooperativa de Quesos 'Las Cabras'	Antonio del Valle Caavedra	Export Administrator	(98) 598 76 54
Escaragots Boussaux	Marie Delamare	Sales Manager	85.57.00.07
Exceller Liquids	Charlotte Cooper	Purchasing Manager	(71) 555-2222
Formaggi Fortini S.r.l.	Elio Rossi	Sales Representative	(0544) 60313
Forêts d'Orchères	Chantal Goulet	Accounting Manager	(514) 555 2055
G'day, Mate	Wendy Mackenzie	Sales Representative	(02) 555-5914
Gai pâturage	Eliane Noz	Sales Representative	38.76.99.06
Grandma Kelly's Homestead	Regina Murphy	Sales Representative	(313) 555 5735

Рисунок 1. Список поставщиков. Эта таблица была создана с помощью листинга 2, в котором используется класс-оболочка Word. Метод DBFToTable обрабатывает все детали перемещения данных из курсора в таблицу Word.

Обёртка Excel - от таблицы к графику

Как мы видели в главе 9 «Графический механизм Excel», Excel имеет замечательный графический механизм. Метод ChartWizard имеет множество параметров, большинство из которых не меняются в ходе работы приложения. Здесь метод MakeGraph принимает диапазон данных и создает лист диаграммы, отображающий данные. Он принимает три параметра: один обязательный (диапазон данных в качестве объекта) и два необязательных (строка для имени листа диаграммы и заголовков графика).

Ряд пользовательских свойств содержит значения по умолчанию для диаграммы. Свойство oChart хранит ссылку на объект диаграммы на случай, если вы захотите напрямую редактировать диаграмму. nChartType определяет тип диаграммы (линейчатая, гистограмма и т. д.), а nChartFormat определяет, какой из предопределенных форматов Excel используется (полное описание см. в разделе «Типы диаграмм» в главе 9). По умолчанию для этих двух типов используется тип

столбца, использующий первый предопределенный формат. Ваша программа может легко изменить эти свойства для создания различных типов диаграмм (или добавить дополнительные параметры для переопределения настроек свойств). Свойство `nChartPlotBy` определяет, расположены ли ряды в строках или столбцах; по умолчанию – столбцы. Некоторые другие параметры вызова `ChartWizard` жестко запрограммированы, но их также можно переключить на ссылку к свойствам объекта `cusExcel` или к параметрам, передаваемым в метод `MakeGraph`. Вот код:

```
* cusExcel.MakeGraph
* Этот метод является оболочкой метода Excel ChartWizard.
* Он создает график из диапазона данных.

LPARAMETERS oGraphRange, cSheetName, cTitle

* Проверьте oGraphRange.
IF VarType(oGraphRange) <> "O"
    This.TellUser("Диапазон графика должен быть объектом.")
    RETURN .F.
ENDIF

* Проверьте cSheetName
IF VarType(cSheetName) <> "C"
    cSheetName = .NULL.
ENDIF

* Проверьте cTitle
IF VarType(cTitle) <> "C"
    cTitle = ""
ENDIF

WITH This.oServer

    * Создайте диаграмму.
    .Charts.Add()
    IF NOT ISNULL(cSheetName)
        .ActiveChart.Name = cSheetName
    ENDIF

    * Единственные константы заключаются в том, что есть
    * одна строка и один столбец для меток рядов и категорий,
    * а также легенда (соответствует трем единицам).
    This.oChart = .ActiveChart.ChartWizard(oGraphRange, ;
        This.nChartType, This.nChartFormat, This.nChartPlotBy, ;
        1, 1, 1, cTitle, "", "", "")

ENDWITH
```

Первая часть кода выполняет серию проверок, чтобы убедиться, что вы передали правильные данные. Затем он добавляет лист диаграммы, переименовывает его (если передано допустимое имя), а затем форматирует диаграмму с помощью метода `ChartWizard`. Что делает это таким особенным, так это то, что теперь вам нужно запомнить три параметра – диапазон для построения графика, как назвать результирующую диаграмму и как назвать диаграмму – вместо длинной серии параметров `ChartWizard`. Если вашим графиком требуется специальное

форматирование (например, включение корпоративных цветов в фон или изменение шрифта, размера и цвета осей), вы можете включить его здесь. Представьте, сколько времени вы сэкономите как разработчик, написав этот код один раз.



В листинге 3 показана программа, использующая класс-оболочку Excel со специфичным для Excel методом MakeGraph. Она создает экземпляр сервера Автоматизации и открывает новый документ (в случае Excel – рабочую книгу). Затем она создает курсор на основе таблицы Product из TasTrade, выбирая только записи Confections. Она отправляет эти данные в новую книгу, создает график, сохраняет документ и закрывает сервер. Эта программа включена как BuildGraph.PRG в файлы загрузки для разработчиков, доступные по адресу www.hentzenwerke.com. На рисунке 2 показана часть полученного документа.

Листинг 3. Использование метода, специфичного для Excel. Эта программа создает график Excel с помощью одного вызова метода из класса-оболочки Excel.

```
* BuildPptGraph.prg
* © 2000, Tamar E. Granor and Della Martin
* From: Microsoft Office Automation with Visual FoxPro
* Hentzenwerke Publishing. www.hentzenwerke.com

* Продемонстрируйте класс-оболочку Excel с помощью
* метода MakeGraph, специфичного для Excel.

LOCAL oExcel, oRange, cRow

* Открыть Excel
oExcel = NewObject("cusExcel","automation")

WITH oExcel
    .OpenServer()

    IF NOT .IsServerOpen()
        WAIT WINDOW "Не удалось открыть Excel."
        RETURN .F.
    ENDIF

    * Создайте книгу.
    .NewDocument()

    * Проверьте, открыта ли книга
    IF .IsDocOpen()

        * Получить курсор данных о продукте
        IF NOT DBUSED("TasTrade")
            OPEN DATABASE (_SAMPLES + "TASTRADE\DATA\Tastrade")
        ENDIF
        IF NOT USED("Products")
            USE Products IN 0
        ENDIF

        SELECT English_Name, ;
```

```
        Unit_Price, ;
        Unit_Cost ;
    FROM Products ;
    WHERE Category_ID = "      3" ;
        AND NOT Discontinued ;
    INTO CURSOR GraphProduct

WITH .oDocument.Sheets[1]
    * Поместите заголовки данных.
    .Range("A1").Value = "Название продукта"
    .Range("B1").Value = "Цена за единицу"
    .Range("C1").Value = "Стоимость единицы"

    * Поместите данные курсора в электронную таблицу.
    SELECT GraphProduct
    SCAN
        cRow = ALLTRIM(STR(RECNO() + 1))
        .Range("A" + cRow).Value = GraphProduct.English_Name
        .Range("B" + cRow).Value = GraphProduct.Unit_Price
        .Range("C" + cRow).Value = GraphProduct.Unit_Cost
    ENDSCAN

    * Настройка диапазона.
    oRange = .Range("A1:C" + cRow)

ENDWITH

    * Постройте график.
    .MakeGraph(oRange, "Кондитерские изделия", ;
        "Цена и стоимость кондитерских изделий")

ENDIF

RELEASE oRange

    * Сохраните новую книгу.
    * .SaveDocument(,SYS(2023) + "\sample.xls")
    .SaveDocument( , FULLPATH("") + "sample.xls")

    * Закройте Excel.
    .CloseServer()
ENDWITH

    * Очистить данные.
CLOSE DATA ALL

RETURN
```

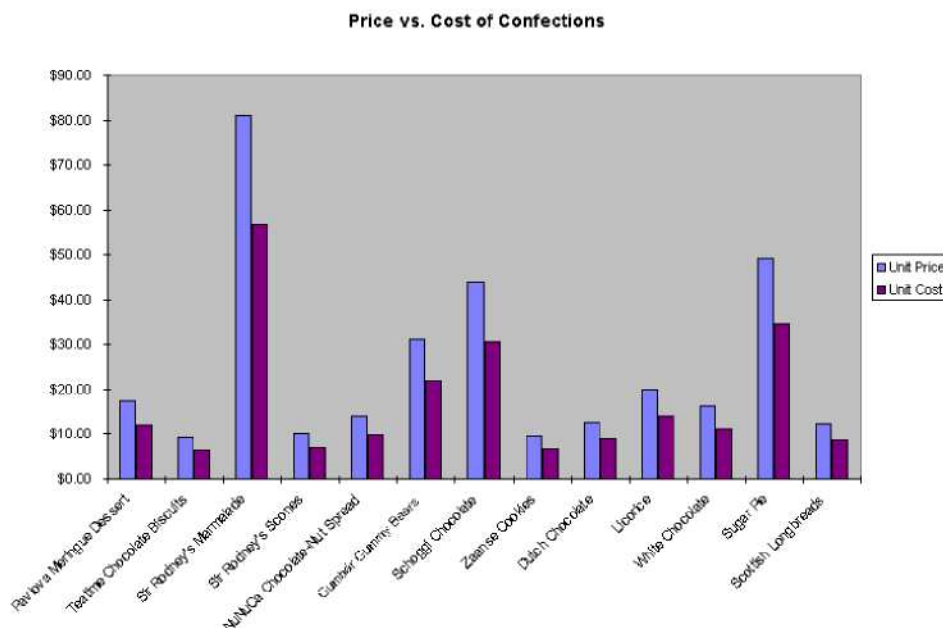


Рисунок 2. График, показывающий зависимость цены от стоимости кондитерских изделий. Этот график был создан с помощью листинга 3, в котором используется класс-оболочка Excel. Метод MakeGraph обрабатывает все детали создания графика.

В примере показан еще один специфический для Excel метод, который вам, вероятно, понадобится, — метод отправки данных из курсора VFP на рабочий лист. При таком методе код в листинге 3 можно было бы упростить еще больше.

Обёртка PowerPoint - стандартизация текста

В главе 10 «Основы PowerPoint» мы обсуждаем множество свойств, доступных для текстовых фреймов. Хороший дизайн презентации требует, чтобы шрифт, размер и цвет оставались постоянными на протяжении всей презентации, чтобы избежать эффекта требования выкупа (чрезмерного количества соседствующих шрифтов). Конечно, бывают случаи, когда необходимо изменить одно или несколько свойств, чтобы привлечь внимание к фразе. Мы создали метод AddTextFrame для стандартизации свойств текста, но при этом допускаем быстрые изменения.

Прежде чем мы перейдем к методу AddTextFrame, мы должны обсудить серьезное несоответствие между PowerPoint и другими серверами. На других серверах всегда есть какой-либо активный документ (ActiveDocument в Word, ActiveWorkbook, ActiveSheet и ActiveChart в Excel), который всегда доступен. PowerPoint работает немного по-другому. Если PowerPoint не отображается, свойство ActivePresentation недоступно (фактически, доступ к нему приводит к ошибке). Кроме того, хотя свойство ActiveSlide отсутствует, вы можете запросить у View свойство SlideIndex. Однако, когда PowerPoint не отображается, View недоступны (в этом случае доступ к View приводит к ошибке). Мы подозреваем, что PowerPoint любит быть видимым (поскольку это очень наглядный продукт), а когда он не виден, он просто не принимает наличие активных представлений или презентаций, поскольку никто не может их просмотреть.

Чтобы все работало правильно, нам нужно внести несколько изменений. Первое — метод oDocument_Access, который проверяет, что oDocument по-прежнему указывает на действительный документ (см. «Отслеживание активного документа» ранее в этой главе). Мы добавили проверку, виден ли сервер, прежде чем устанавливать для oDocument значение ActivePresentation, чтобы невидимые

экземпляры не вызывали ошибку. Здесь есть побочный эффект: если вы запускаете PowerPoint невидушно, вам придется поддерживать актуальность oDocument самостоятельно (обновляя его при каждом изменении презентаций). Вот измененный код метода oDocument_Access:

```
* Убедитесь, что свойство указывает на активный
* в данный момент документ.
IF This.IsServerOpen() AND This.IsDocOpen()

    * PowerPoint не имеет активных свойств, если он не виден.
    * Убедитесь, что вы не меняете активную презентацию,
    * когда PowerPoint не отображается (если вы это делаете,
    * обновите oDocument вручную).
    IF This.oServer.Visible <> 0
        This.oDocument = This.oServer.ActivePresentation
        * В противном случае предположим, что разработчик
        * поддерживает oDocument, и предположим, что все,
        * на что он указывает, действительно.
    ENDIF

ELSE
    This.oDocument = .NULL.
ENDIF

RETURN THIS.oDocument
```

Еще одно изменение, которое нам нужно сделать, — это отслеживать текущий слайд. Для этого требуется собственное свойство oCurrentSlide и собственный метод AddSlide. oCurrentSlide, как следует из названия, отслеживает текущий слайд. Новый метод AddSlide используется для добавления новых слайдов и обновления oCurrentSlide. Он принимает два параметра: значение макета слайда (список значений см. в разделе «Использование макетов слайдов» в главе 10) и индекс нового слайда в слайд-шоу. Вот код:

```
* cusPowerPoint.AddSlide
* Оберните метод Slides.Add, чтобы отслеживать текущий слайд.

LPARAMETER nSlideLayout, nNewIndex
* nSlideLayout = Номер макета слайда. Необязательный.
*                 (Соответствует константам ppLayout.)
*                 Если этот параметр опущен,
*                 используется ppLayoutBlank.
* nNewIndex = Индекс нового слайда. Необязательный.
*              Если этот параметр опущен, используется индекс
*              слайда в oCurrentSlide. Если в oCurrentSlide нет
*              слайда, используется значение 1.

WITH This

    * Проверьте параметры.
    IF PARAMETERS() = 0
        nSlideLayout = ppLayoutBlank
    ENDIF

    IF PARAMETERS() <= 1
```

```

    IF TYPE(".oCurrentSlide") = "O"
        nNewIndex = .oCurrentSlide.SlideIndex
    ELSE
        nNewIndex = 1
    ENDIF
ENDIF

* Добавить слайд.
.oCurrentSlide = .oDocument.Slides.Add(nNewIndex, nSlideLayout)
ENDWITH

```

Метод AddTextFrame помещает текстовое поле в текущую презентацию (на которую ссылается свойство oDocument) на текущем слайде (на который ссылается свойство oCurrentSlide). AddTextFrame использует пользовательские свойства TextFrameFont, TextFrameFontSize и TextFrameTextColor, чтобы задать внешний вид текста. Идея состоит в том, что вы устанавливаете эти свойства один раз и меняете их только при необходимости (не забывая установить их обратно, когда закончите). Та же концепция справедлива для TextFrameHeight и TextFrameWidth, которые задают высоту и ширину текстового фрейма по умолчанию. Сохранив все эти свойства, вы можете сконцентрироваться на нужных вам элементах, которые передаются в качестве параметров: текст, расстояние от верха слайда в пунктах и расстояние от левого края слайда в пунктах. При желании вы можете передать высоту и ширину фрейма (в пунктах), чтобы переопределить свойства по умолчанию. Метод возвращает ссылку на новый фрейм на случай, если вам потребуется дополнительно манипулировать текстовым фреймом. Вот код метода AddTextFrame:

```

* cusPowerPoint.AddTextFrame
* Добавьте текстовое поле на текущий слайд текущей презентации.
LPARAMETERS cTextString, nLeftPoints, nTopPoints, ;
              nWidthPoints, nHeightPoints

* cTextString      = Строка для размещения в текстовом фрейме
* nLeftPoints      = Левый край, в пунктах
* nTopPoints       = Верхний край, в пунктах
* nWidthPoints     = Ширина текстовой рамки в пунктах.
*                  Необязательный; использует свойство
*                  TextFrameWidth, если оно не передано.
* nHeightPoints    = Высота текстового фрейма в пунктах.
*                  Необязательный; использует свойство
*                  TextFrameHeight, если оно не передано.

LOCAL oNewFrame, nCurrentSlide
* oNewFrame        = Ссылка на объект нового TextFrame.
* nCurrentSlide    = Индекс текущего слайда

* Если текстовой строки нет, просто вернитесь.
IF EMPTY(cTextString)
    RETURN .F.
ENDIF

* Проверьте левую и верхнюю части, чтобы убедиться, что они
* числовые.
IF TYPE("nLeftPoints") <> "N"
    This.TellUser("Необходимо указать левый край в пунктах.")

```

```
    RETURN .F.
ENDIF

IF TYPE("nTopPoints") <> "N"
    This.TellUser("Необходимо указать верхний край в пунктах.")
    RETURN .F.
ENDIF

* Проверьте ширину и высоту — если они не переданы, установите
* их по умолчанию; если переданы, убедитесь, что они числовые.
IF TYPE("nWidthPoints") <> "N"

    nWidthPoints = This.TextFrameWidth
ENDIF

IF TYPE("nHeightPoints") <> "N"
    nHeightPoints = This.TextFrameHeight
ENDIF

* Проверьте состояние oCurrentSlide.
IF TYPE("This.oCurrentSlide") <> "O"
    This.TellUser("Свойство oCurrentSlide должно быть
установлено.")
    RETURN .F.
ENDIF

* Поместите текстовый фрейм.
oNewFrame = This.oCurrentSlide.Shapes.AddTextBox(1, ;
    nLeftPoints, nTopPoints, ;
    nWidthPoints, nHeightPoints)

WITH oNewFrame
    * Сделайте его невидимым, чтобы максимизировать
    * производительность.
    .Visible = .F.

    * Добавьте текст в текстовый фрейм и установите остальные
    * свойства.
    WITH oNewFrame.TextFrame
        WITH .TextRange
            .Text = cTextString
            .Font.Name = This.TextFrameFont
            .Font.Size = This.TextFrameFontSize
            .Font.Color.RGB = This.TextFrameTextColor
        ENDWITH

        .MarginLeft      = 0
        .MarginTop       = 0
        .MarginRight     = 0
        .MarginBottom    = 0

    ENDWITH
```



```

.Visible = .T.
ENDWITH

RETURN oNewFrame

```



Листинг 4 представляет собой программу, использующую класс-оболочку PowerPoint. Она создает экземпляр сервера Автоматизации и открывает новый документ (в случае PowerPoint – презентацию). Затем она добавляет новый слайд с помощью метода AddSlide и использует метод AddTextFrame для добавления трех текстовых фреймов. Эта программа включена как BuildPptSlide.PRG в файлы загрузки для разработчиков, доступные по адресу www.hentzenwerke.com. На рисунке 3 показана часть полученного документа.

Листинг 4. Использование подкласса, специфичного для PowerPoint. Эта программа создает слайд PowerPoint с помощью одного вызова класса-оболочки PowerPoint.

```

* BuildPptSlide.prg
* © 2000, Tamar E. Granor and Della Martin
* From: Microsoft Office Automation with Visual FoxPro
* Hentzenwerke Publishing. www.hentzenwerke.com

* Демонстрация класса-оболочки PowerPoint с использованием
* методов, специфичных для PowerPoint.

#DEFINE CR CHR(13)
#DEFINE ppLayoutTitle 1
#DEFINE msoIn2Pts 72

LOCAL oPowerPoint, OrigTextColor

* Открыть PowerPoint.
oPowerPoint = NewObject("cusPowerPoint","automation")

WITH oPowerPoint
    .OpenServer()

    IF NOT .IsServerOpen()
        WAIT WINDOW "Не удалось открыть PowerPoint."
        RETURN .F.
    ENDIF

    * Создайте документ
    .NewDocument()

    * Проверьте, открыт ли документ
    IF .IsDocOpen()

        * Добавьте слайд, используя класс-оболочку.
        * Отсутствие параметров означает, что в качестве первого
        * слайда будет добавлен пустой слайд.
        .AddSlide()

        * Поместите несколько текстовых фреймов на новый слайд.

```

```
.AddTextFrame("First text box.", 2 * msoIn2Pts, ;
              2 * msoIn2Pts)
.AddTextFrame("Second text box.", 6 * msoIn2Pts, ;
              2 * msoIn2Pts)

OrigTextColor = .TextFrameTextColor
.TextFrameTextColor = RGB(0, 255, 255) && Полубой
.AddTextFrame("Third text box.", 4 * msoIn2Pts, ;
              4 * msoIn2Pts)
.TextFrameTextColor = OrigTextColor
```

```
ENDIF
```

```
* Сохраните новый документ.
*.SaveDocument(, SYS(2023) + "\sample.ppt")
.SaveDocument(, FULLPATH("") + "sample.ppt")

* Закройте PowerPoint
.CloseServer()
```

```
ENDWITH
```

```
RETURN
```

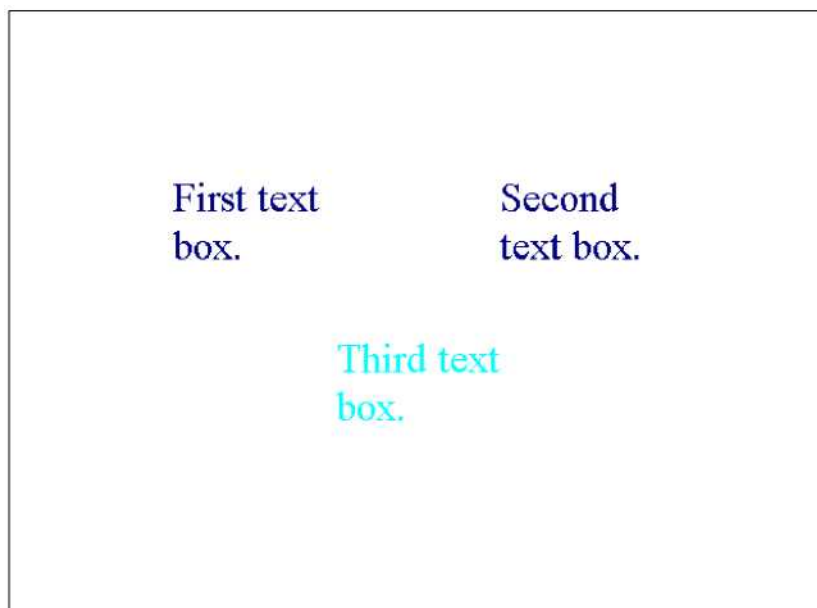


Рисунок 3. Слайд, показывающий простые текстовые фреймы, добавленные с помощью метода AddTextFrame, специфичного для PowerPoint. Этот слайд создан с помощью листинга 4, в котором используется класс-оболочка PowerPoint.

15.7 Подводя итоги

Создание собственных классов-оболочек может показаться непосильной задачей. Просто помните, что вам не обязательно делать все это сразу. Секрет в том, чтобы сохранять бдительность и, как только вы поймаете себя на мысли: «Я уже писал этот код раньше», остановитесь и найдите время, чтобы завершить его.

В идеальном мире, конечно, вы должны обернуть каждый серверный метод перед его использованием, чтобы вас не застали врасплох, когда следующая версия серверного приложения внесет изменения. Но мы не пишем код в идеальном мире. Оберните методы, которые вы регулярно используете, и приложите максимум

усилий к коду, который вы используете чаще всего.

Если вы обнаружите, что метод становится слишком сложным или требует большого количества параметров, рассмотрите возможность его разделения на несколько методов.

Что бы вы ни упаковывали, тщательно документируйте это. Вероятно, вы помните, как использовать методы, которые используете каждый день, но вам не обязательно помнить имена и параметры методов, которые вы используете только раз в месяц.

Приложение А. Интернет-сообщества пользователей

Когда вы действительно застряли, лучшее место, где можно получить помощь, – это какая-нибудь онлайн-группа пользователей.

Мы впервые встретились в Интернете. Прежде чем мы действительно встретились лицом к лицу, мы довольно долго переписывались на FoxForum CompuServe. Таким же образом мы встретились с техническим редактором этой книги и с издателем. Фактически, большинство наших лучших возможностей в сообществе FoxPro появились благодаря отношениям, которые начались в Интернете. Поэтому мы твердо верим, что за помощью в решении проблем с программированием можно обратиться в режиме онлайн.

Единственное, что отличает один ресурс от другого, – это качество помощи, которую вы получаете. По большей части вся доступная помощь исходит от волонтеров, которым нравится отвечать на вопросы, чтобы отточить свои навыки или поблагодарить людей, которые помогали им на этом пути. Многие из людей (включая Тамар), которые отвечают на вопросы в перечисленных нами местах, являются самыми ценными профессионалами Microsoft, получателями ежегодной награды, вручаемой тем, кто тратит слишком много времени, помогая другим в Интернете.

Сложность листинга онлайн-ресурсов заключается в том, что детали могут измениться. Некоторые места остаются на месте в течение длительного времени, но другие, похоже, регулярно перемещаются. Даже те, которые мы считали бы постоянными, иногда подвергаются пересмотру. Пока мы писали эту книгу, CompuServe внесла серьезные изменения в структуру своего форума. Итак, хотя этот список верен на момент его написания (а это последняя часть книги, которую мы пишем), мы с грустью осознаем, что в ту минуту, когда мы перенесем его на бумагу, что-то может измениться. Поэтому выполняйте поиск по ссылкам, которые мы перечисляем, или используйте поисковые системы для поиска сайтов с похожими названиями, если указанные нами точные местоположения больше не действительны.

Мы разделили этот список на две части: ресурсы FoxPro и ресурсы Office. Как разработчики FoxPro, мы гораздо лучше знакомы с первым, но мы сделали все возможное, чтобы откопать и второе. Кроме того, прежде чем мы начнем перечислять сообщества пользователей, мы должны напомнить вам, что Microsoft предлагает значительные онлайн-ресурсы на своем веб-сайте MSDN, msdn.microsoft.com. Этот сайт предоставляет доступ к базе знаний Microsoft, множеству документации и технических статей, а также множеству других вспомогательных документов. Часто это хорошее место для начала поиска помощи.

А1. Ресурсы FoxPro

Их дедушкой, местом, где все началось, является CompuServe. Компания Fox присутствовала там почти с тех пор, как появились продукты Fox. Начав с одного раздела на форуме с участием нескольких поставщиков, такого как Topsy, FoxForum от Fox Software «рос и разрастался», так что ему понадобился целый форум. Второй, менее технический форум (FoxUsers) был добавлен в начале 90-х. Оба пережили переход к спонсорству Microsoft, а затем, наконец, к третьей стороне, когда Microsoft покинула CompuServe и перешла в Интернет. В течение короткого славного периода, когда впервые появился Visual FoxPro, существовало даже три форума FoxPro. Вскоре это снова превратилось в два форума, примерно разделенных на один для Visual FoxPro и один для FoxPro 2.x и общих обсуждений. Затем, в декабре 1999 года, два форума FoxPro были объединены с несколькими

другими форумами, посвященными программному обеспечению Microsoft, и образовали форум MSDevApps. Четыре его раздела посвящены специально FoxPro и Visual FoxPro, а ряд других разделов посвящен более общим вопросам разработки и темам, связанным с разработкой, ориентированной на Microsoft (например, HTML Help). Форумы CompuServe приятно использовать из-за их богатой многопоточной модели обмена сообщениями. За каждым исходным сообщением следуют ответы на это сообщение, а также ответы на ответы и так далее. Помимо создания цепочки, на опубликованные вами сообщения легко найти ответы, что сложно сделать во многих веб-системах обмена сообщениями, даже в тех, которые имеют цепочку сообщений. Кроме того, CompuServe делает огромную работу по предотвращению спама: вы не найдете оскорбительных сообщений или рекламы схем быстрого обогащения или порносайтов. С апреля 2000 года форумы CompuServe открыты для широкой публики, поэтому вы можете участвовать в них через www.compuserve.com, даже если вы не являетесь участником CompuServe.

Microsoft спонсирует группы новостей для FoxPro и Visual FoxPro на своем общедоступном сервере msnews.microsoft.com (Microsoft заверила нас, что если произойдут какие-либо изменения в адресах любого из их сайтов FoxPro, информация появится на msdn.microsoft.com/vfoxpro – или, по крайней мере, этот адрес будет перенаправлен на новый адрес). Ищите группы, имена которых начинаются с «microsoft.public.fox». Хотя Microsoft старается не допускать навязчивых предложений и других сообщений не по теме, здесь такого материала больше, чем на любом другом сайте FoxPro, который мы перечисляем.

Одним из самых оживленных мест для обсуждения FoxPro в Интернете является Universal Thread на сайте www.universalthread.com. Как и CompuServe, он предлагает цепочку сообщений, позволяющую находить ответы на ваши собственные сообщения. Он находится в частной собственности и предлагает бесплатный доступ ко всем сообщениям с ограниченным пользовательским интерфейсом. За разумную ежемесячную плату вы получаете членство Premier, которое дает доступ к некоторым более продвинутым инструментам для его использования.

Новый сайт, который привлек много внимания: www.foxforum.com, где используется другой подход к просмотру сообщений и ответов на них. Он объединяет их все в одно сообщение, так что вам не придется постоянно нажимать «Далее», «Далее», «Далее», чтобы просмотреть всю ветку.

Уникальной в сообществе FoxPro является сестра форума – вики FoxForum, fox.wikis.com. Это не дискуссионная группа или форум, а развивающаяся база знаний, где каждый может публиковать информацию по теме, а другие могут добавлять к ней или редактировать ее. Там вы можете найти записи обо всем, от объектно-ориентированных фреймворков до соглашений об именах и практически обо всем, что связано с FoxPro и сообществом Fox, о чем вы только можете подумать. Если его там нет, вы можете добавить его самостоятельно.

Advisor Media спонсирует форумы по различным продуктам, включая FoxPro, на своем веб-сайте www.advisor.com. Выберите форумы Advisor на главной странице.

Совершенно новым, когда мы составляли этот список, является www.vfp guru.com. Сайт выглядит впечатляюще, на нем есть форумы и зона чата, а также место для скачивания и статей. Еще неизвестно, привлечет ли он достаточно пользователей, чтобы на него стоило заглянуть.

Также новым дополнением является дискуссионная группа по Visual FoxPro, спонсируемая техническими публикациями Fawcette. Однако, в отличие от vfp guru.com, devx.com хорошо известен в сообществе разработчиков своими другими дискуссионными группами, поэтому мы ожидаем, что эта группа быстро

станет активной. Вы найдете его на news.devx.com.

В дополнение ко всему этому, ряд компаний и частных лиц, работающих с Visual FoxPro, предоставляют полезную информацию и файлы для загрузки на своих веб-сайтах. Ссылки на них вы найдете либо в качестве рекламы на некоторых сайтах, перечисленных здесь, либо в самих сообщениях.

A2. Ресурсы Office

Как и в случае с FoxPro, вы можете найти справку по продуктам Office на CompuServe. Как и в случае с FoxPro, форумы, поддерживающие Office, были объединены в конце 1999 года. Новым домом для Office стал MSOForum, и он включает разделы для Word, Excel, PowerPoint, Outlook, Access и ряда других сопутствующих продуктов. И что самое приятное, как и в случае с FoxPro, люди, которые там тусуются, теплые, дружелюбные и отзывчивые. Чтобы попасть туда в Интернете, наведите курсор на go.compuserve.com/msofficeforum?loc=US.

Общедоступные группы новостей Microsoft на сайте msnews.microsoft.com включают группы для всех продуктов Office. Они разбросаны по всему серверу, но ищите группы с различными названиями продуктов в именах и строках типа «vba», «programming» или «automation». Не будьте слишком разборчивы, хотя, похоже, для всех вопросов PowerPoint существует одна группа под названием microsoft.public.powerpoint.

Вуди Леонхард писал о Word и других продуктах Office практически еще до того, как мы о них услышали. Его веб-сайт, www.wopr.com предлагает группы обсуждения продуктов Office, а также пару информационных бюллетеней *Woody's Office Watch* и *Woody's Windows Watch* и многое другое.

У многих Microsoft MVP есть веб-сайты, на которых можно найти советы, подсказки и код. Вместо того, чтобы пытаться предоставить здесь полный список сайтов, связанных с Office, мы указываем вам на сайт, который содержит такой список и время от времени обновляется. Мы благодарим Карла Петерсена за ведение этого списка. Посетите www.mvps.org/links.html чтобы найти ссылки на сайты, посвященные не только Office, но и практически любому продукту Microsoft, о котором вы только можете подумать.

Наконец, если все остальное не помогло, не забудьте попробовать поисковые системы, особенно те, которые архивируют сообщения групп новостей. Поскольку инструменты, с которыми мы работаем, распространены по всему миру, редко когда вы сталкиваетесь с проблемой, которую никто не увидел или не решил.

A3. Онлайн-этикет

В заключение мы приведем несколько комментариев о том, как вести себя на форумах и в группах новостей. Главное - запомнить слово «сообщество», которое мы использовали в названии этого приложения. Ведите себя так же, как в своем районе, и вы прекрасно поладите.

Что это значит? Во-первых, почти каждый, кто отвечает на вопросы онлайн, является волонтером и делает это потому, что ему это нравится. Это не его или ее работа, поэтому не ожидайте, что кто-то «должен» вам ответить. Человек, который отвечает на ваш вопрос, делает вам одолжение. Подтвердите эту услугу своим отношением.

Аналогично, причина, по которой онлайн-сообщества работают, заключается в том, что люди находят время, чтобы ответить на вопросы других людей. Если можете, оплатите тем же и ответьте на чужой вопрос.

В рамках данного форума или группы новостей задавайте свой вопрос только один раз. Завсегда и читают все разделы форума или все группы по своей теме, значит, увидят. Публикация одного и того же вопроса во многих группах (так называемая «перекрестная публикация») считается грубостью.

Когда вы решили свою проблему, опубликовать короткую заметку «Спасибо, это помогло» можно, но это не является строгой необходимостью. Не пишите «спасибо» всем, кто вам ответил. Достаточно одного «спасибо всем, кто помог». Если проблема была особенно сложной, опубликуйте одно сообщение с объяснением того, как вы наконец ее решили, чтобы другие могли поучиться на вашем опыте.

Не отправляйте электронное письмо тому, кто ответит на ваш вопрос, если он или она специально не попросит вас об этом. Одним из преимуществ общественных форумов и групп новостей является то, что многие люди учатся там. Когда вы переключаетесь на частную электронную почту, другие не смогут извлечь уроки из вашего обсуждения. Кроме того, многие из тех, кто отвечает на вопросы, просто слишком заняты, чтобы отвечать на вопросы по электронной почте.

Используйте свое настоящее имя. Технические форумы – не место милым ручкам.

Большинство интернет-сообществ имеют набор правил. Например, правила поведения в группах новостей Microsoft довольно просты: их можно найти по адресу support.microsoft.com/support/news/rules.asp. В Universal Thread список гораздо шире: доступ к правилам можно получить по ссылке на главной странице. Куда бы вы ни пошли, вы сможете найти правила; если вы не можете их найти, спросите. Соблюдение местных обычаев – такая же хорошая идея, когда вы посещаете онлайн-сообщество, как и во время путешествия в физическом мире.

Приложение В. Значения ChartWizard

Метод Excel ChartWizard принимает два параметра и использует их комбинацию для установки свойства ChartType (и, возможно, других свойств).

В главе 9 «Графический механизм Excel» в таблице 1 показаны некоторые значения переменных nChartFormat и nChartType для метода ChartWizard. В этом приложении показан полный ряд значений.

Метод ChartWizard принимает два параметра: nChartType и nChartFormat, и их комбинация устанавливает свойство ChartType объекта Chart. Значение, которое нужно передать в качестве параметра nChartType, показано в заголовке, выделенном жирным шрифтом, вместе с его константой. Например, в первой строке указано «Для типа диаграммы: **xlArea (1)**». Используйте значение 1 или константу xlArea для свойства nChartType, чтобы выбрать диаграмму с областями. Для параметра nChartFormat выберите значение из второго столбца («Параметр формата диаграммы») на основе показанного описания.

Метод ChartWizard заполняет свойство ChartType значением, указанным в столбце «Свойство типа диаграммы». В столбце «Константа типа диаграммы» указано соответствующее имя константы.

Обратите внимание, что ChartWizard задает больше свойств, чем просто ChartType; линии сетки, метки и т.п. также задаются как часть параметра nChartFormat.

Элементы, отмеченные звездочкой (*), чрезвычайно похожи на одноименные элементы выше в таблице, настолько, что практически неотличимы (хотя должно быть какое-то отличие, возможно, одно-единственное свойство, которое не сразу бросается в глаза).

Описание формата диаграммы	Параметр ChartFormat	Свойство ChartType	Константа типа диаграммы
Для типа диаграммы: xlArea (1) (с областями)			
Диаграмма с областями с накоплением	1	76	xlAreaStacked
Нормированная диаграмма с областями	2	77	xlAreaStacked100
Диаграмма с областями с накоплением с вертикальными блоковыми линиями	3	76	xlAreaStacked
Диаграмма с областями с накоплением с черными линиями сетки	4	76	xlAreaStacked
Диаграмма с областями с накоплением с метками серий	5	76	xlAreaStacked
Для типа диаграммы: xlBar (2) (линейчатая)			
Линейчатая диаграмма	1	57	xlBarClustered
Линейчатая диаграмма *	2	57	xlBarClustered
Линейчатая диаграмма с накоплением	3	58	xlBarStacked

Описание формата диаграммы	Параметр ChartFormat	Свойство ChartType	Константа типа диаграммы
Линейчатая диаграмма с перекрытием, установленным на 30	4	57	xlBarClustered
Нормированная линейчатая диаграмма	5	59	xlBarStacked100
Линейчатая диаграмма с черными вертикальными линиями сетки	6	57	xlBarClustered
Линейчатая диаграмма с метками данных	7	57	xlBarClustered
Линейчатая диаграмма с 0 перекрытий и 0 шириной зазора	8	57	xlBarClustered
Линейчатая диаграмма с накоплением с последовательными линиями	9	58	xlBarStacked
Нормированная линейчатая диаграмма с последовательными линиями	10	59	xlBarStacked100
Для типа диаграммы: xlColumn (3) (гистограмма)			
Обычная гистограмма с шириной зазора, установленной на 150.	1	51	xlColumnClustered
Обычная гистограмма с шириной зазора, установленной на 150.	2	51	xlColumnClustered
Гистограмма с накоплением	3	52	xlColumnStacked
Обычная гистограмма с перекрытием, равным 30	4	51	xlColumnClustered
Нормированная гистограмма	5	53	xlColumnStacked100
Обычная гистограмма с горизонтальными линиями сетки	6	51	xlColumnClustered
Обычная гистограмма с метками данных	7	51	xlColumnClustered
Обычная гистограмма с 0 перекрытий и 0 шириной зазора	8	51	xlColumnClustered
Гистограмма с накоплением с последовательными линиями	9	52	xlColumnStacked
Нормированная гистограмма с последовательными линиями	10	53	xlColumnStacked100
Для типа диаграммы: xlLine (4) (график)			

Описание формата диаграммы	Параметр ChartFormat	Свойство ChartType	Константа типа диаграммы
Линия с маркерами	1	65	xlLineMarkers
Линия без маркеров данных	2	4	xlLine
Только маркеры данных (без соединительных линий)	3	65	xlLineMarkers
Линии с маркерами и пунктирной горизонтальной сеткой	4	65	xlLineMarkers
Линии с маркерами и пунктирные линии сетки	5	65	xlLineMarkers
Линии с маркерами и логарифмической шкалой	6	65	xlLineMarkers
Маркеры данных только с линиями min-max	7	65	xlLineMarkers
График акций	8	88	xlStockHLC
График акций *	9	88	xlStockHLC
Сглаженная линия без маркеров	10	4	xlLine
Для типа диаграммы: xlPie (5) (круговая)			
Круговая диаграмма без надписей	1	5	xlPie
Круговая диаграмма с надписями, с выделением первого клина	2	5	xlPie
Круговая диаграмма без надписей	3	5	xlPie
Разрезанная круговая диаграмма	4	69	xlPieExploded
Круговая диаграмма с метками	5	5	xlPie
Круговая диаграмма с метками данных в процентах	6	5	xlPie
Круговая диаграмма с метками и метками данных в процентах	7	5	xlPie
Для типа диаграммы: xlRadar (-4151) (лепестковая)			
Лепестковая диаграмма с маркерами и метками категорий	1	81	xlRadarMarkers
Лепестковая диаграмма метками категорий	2	-4111	xlCombination
Лепестковая диаграмма без осей, с метками категорий	3	-4111	xlCombination

Описание формата диаграммы	Параметр ChartFormat	Свойство ChartType	Константа типа диаграммы
Лепестковая диаграмма с метками категорий и линиями сетки	4	-4111	xlCombination
Лепестковая диаграмма с логарифмической осью	5	-4111	xlCombination
Заполненная лепестковая диаграмма с метками категорий	6	82	xlRadarFilled
Для типа диаграммы: xlXYScatter (-4169) (точечная)			
Точечная диаграмма	1	-4169	xlCombination
Точечная диаграмма с соединительными линиями	2	74	xlXYScatterLines
Точечная диаграмма с черными линиями сетки	3	-4169	xlCombination
Точечная диаграмма по логарифмической оси, горизонтальные линии	4	-4169	xlCombination
Точечная диаграмма по логарифмической оси, обе линии сетки	5	-4169	xlCombination
Сглаженная линия без маркеров	6	73	xlCombination
Для типа диаграммы: xl3DArea (-4098) (объемный вариант диаграммы с областями)			
Объемная диаграмма с областями с накоплением	1	78	xl3DAreaStacked
Объемная диаграмма с областями с накоплением с метками серий	2	78	xl3DAreaStacked
Объемная диаграмма с областями с накоплением с вертикальными блочными линиями, без сетки	3	78	xl3DAreaStacked
Объемная диаграмма с областями с накоплением с сеткой	4	78	xl3DAreaStacked
Объемная диаграмма с областями, без линий сетки	5	-4098	xl3DArea
Объемная диаграмма с областями, обе линии сетки	6	-4098	xl3DArea
Объемная диаграмма с областями, вертикальные линии сетки	7	-4098	xl3DArea

Описание формата диаграммы	Параметр ChartFormat	Свойство ChartType	Константа типа диаграммы
Объемная диаграмма с областями с накоплением с меткой данных, без стен	8	78	xl3DAreaStacked
Для типа диаграммы: xl3DBar (-4099) (объемный вариант линейчатой диаграммы)			
Объемная линейчатая диаграмма	1	60	xl3DBarColumn
Объемная линейчатая диаграмма с накоплением	2	61	xl3DBarStacked
Объемная нормированная линейчатая диаграмма	3	62	xl3DBarStacked
Объемная линейчатая диаграмма, вертикальные линии сетки	4	60	xl3DBarColumn
Объемная линейчатая диаграмма, вертикальные линии сетки, без стен	5	60	xl3DBarColumn
Для типа диаграммы: xl3DColumn (-4100) (объемный вариант гистограммы)			
Объемная гистограмма	1	54	xl3DColumnClustered
Объемная гистограмма с накоплением	2	55	xl3DColumnStacked
Объемная нормированная гистограмма	3	56	xl3DColumnStacked100
Объемная гистограмма, горизонтальные линии сетки	4	54	xl3DColumnClustered
Трехмерная гистограмма, без сеток	5	-4100	xl3DColumn
Трехмерная гистограмма, обе сетки	6	-4100	xl3DColumn
Трехмерная гистограмма, вертикальные сетки	7	-4100	xl3DColumn
Объемная гистограмма, горизонтальные линии сетки, без стен	8	54	xl3DColumnClustered
Для типа диаграммы: xl3DLine (-4101) (объемный график)			
Объемный график, без сетки	1	-4101	xl3DLine
Объемный график с обеими линиями сетки	2	-4101	xl3DLine
Объемный график с вертикальными линиями сетки	3	-4101	xl3DLine

Описание формата диаграммы	Параметр ChartFormat	Свойство ChartType	Константа типа диаграммы
Объемный график, логарифмическая ось	4	-4101	xl3DLine
Для типа диаграммы: xl3DPie (-4102) (объемный вариант круговой диаграммы)			
Объемная круговая диаграмма без надписей	1	-4102	xl3DPie
Объемная круговая диаграмма с надписями, выделенный клин	2	-4102	xl3DPie
Объемная круговая диаграмма без надписей	3	-4102	xl3DPie
Разрезанная объемная круговая диаграмма без надписей	4	70	xl3DPieExploded
Объемная круговая диаграмма с метками	5	-4102	xl3DPie
Объемная круговая диаграмма с метками процентных данных	6	-4102	xl3DPie
Объемная круговая диаграмма с метками и метками процентных данных	7	-4102	xl3DPie
Для типа диаграммы: xl3DSurface (-4103) (поверхность)			
Поверхность с цветом	1	83	xlSurface
Каркасная 3D-поверхность без цвета	2	84	xlSurfaceWireFrame
Контурная диаграмма (вид сверху на поверхностную диаграмму) с цветом	3	85	xlSurfaceTopView
Контурная диаграмма (вид сверху на поверхностную диаграмму) без цвета	4	86	xlSurfaceTopViewWire frame
Для типа диаграммы: xlDoughnut (-4120) (кольцевая)			
Кольцевая диаграмма	1	-4120	xlDoughnut
Разрезанная кольцевая диаграмма, выделена точка данных	2	-4120	xlDoughnut
Разрезанная кольцевая диаграмма	3	-4120	xlDoughnut
Разрезанная кольцевая диаграмма *	4	80	xlDoughnutExploded
Кольцевая диаграмма с метками данных	5	-4120	xlDoughnut

Описание формата диаграммы	Параметр ChartFormat	Свойство ChartType	Константа типа диаграммы
Кольцевая диаграмма с метками процентных данных	6	-4120	xlDoughnut
Кольцевая диаграмма с метками и метками процентных данных	7	-4120	xlDoughnut
Для типа диаграммы: xlCombination (-4111) (составная)			
Столбцы и линия	1	-4111	xlCombination
Столбец и линия, вторая ось – валюта	2	-4111	xlCombination
Линии, вторая ось – валюта	3	-4111	xlCombination
Столбец и площадь	4	-4111	xlCombination
Запас	5	88	xlStockHLC
Столбец и min-max	6	-4111	xlCombination