АНДРЕЙ МИХАЙЛОВ

bhv®

1С: ПРЕДПРИЯТИЕ 7.7/8.0

СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ

ПРИМЕНЕНИЕ ТЕХНОЛОГИЙ OLE, COM, ActiveX

ИСПОЛЬЗОВАНИЕ СКРИПТОВ WSH И WMI

ПРИМЕНЕНИЕ ADO И SQL-DMO ПРИ РАБОТЕ С БАЗАМИ ДАННЫХ

ИСПОЛЬЗОВАНИЕ ACTIVE DIRECTORY SERVICE INTERFACES

PRO

ПРОФЕССИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ

+CD

УДК 681.3.06
ББК 32.973.26-018.2
М69

Михайлов А. В.

М69 1С:Предприятие 7.7/8.0: системное программирование. — СПб.: БХВ-Петербург, 2005. — 336 с.: ил.

ISBN 5-94157-536-X

Содержится информация, предназначенная для практического применения скрытых возможностей операционной системы и разнообразных COM-объектов при разработке собственных информационных систем на платформе "1С:Предприятие" версий 7.7 и 8.0. Рассмотрены особенности использования объектов ActiveX, администрирования Windows средствами WMI и WSH, взаимодействия с базами данных посредством интерфейсов ADO и SQL-DMO, применения технологии OLE Automation и др. при работе в среде "1С:Предприятие". Материал сопровождается большим количеством наглядных и подробно прокомментированных примеров, полные тексты которых приведены на компакт-диске.

Для программистов

УДК 681.3.06
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. гл. редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Владимир Красильников*
Компьютерная верстка	*Натальи Караваевой*
Корректор	*Зинаида Дмитриева*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 20.12.04.
Формат 70×100$^{1}/_{16}$. Печать офсетная. Усл. печ. л. 27,09.
Тираж 4000 экз. Заказ № 722
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

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

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

ISBN 5-94157-536-X

Содержание

ОАО «ТЕХНОЛОГИИ УСПЕХА»

ФУНДАМЕНТ™

НАДОЕЛИ ПРОГРАММЫ-ПОЛУФАБРИКАТЫ?

Поставщики корпоративных информационных систем предлагают Вам «почти готовые» системы «быстрого приготовления». Вам уже надоело сражаться с «почти готовностью» к настоящей работе? Давайте оставим работу с полуфабрикатами программистам. Вам нужен действительно готовый продукт, созданный именно для Вас.

ОАО «Технологии Успеха» специализируется на проектах разработки и внедрения заказных информационных систем на платформе «1С:Предприятие 7.7/8.0». Это означает, что каждое Ваше требование будет учтено и реализовано точно в срок. Мы гарантируем Вам постоянную полную готовность системы к работе независимо от количества Ваших пожеланий по ее усовершенствованию.

НЕ коробочные программные продукты

Вы можете запросить дополнительную информацию о предлагаемых услугах, а также договориться о проведении презентации и бесплатного экспресс-обследования для подготовки коммерческого предложения, обратившись по нашим контактным телефонам либо электронной почте.

Наши телефоны: в Москве — (095) 975-0442;
в Великом Новгороде — (8162) 112-120
Адрес для связи: info@tehuspeh.ru
Наш сайт в Интернете: www.tehuspeh.ru

НЕ коробочные программные продукты

Введение

Зачем нужна эта книга

Предметом изучения в данной книге является широкий спектр вопросов по профессиональному программированию в системе "1С:Предприятие" версий 7.7 и 8.0, а также использование скрытых и неочевидных возможностей данной системы.

Книга представляет собой практическое руководство, предназначенное для программистов, разработчиков и администраторов системы "1С:Предприятие". В ней рассматриваются следующие темы: работа с технологиями ActiveX, COM, WMI и WSH; OLE Automation; интеграция с базами данных через интерфейсы ADO и SQL-DMO; работа с Интернетом, электронной почтой и FTP; интеграция с внешними приложениями и многое другое.

Главная цель книги — дать читателю основы понимания того, что возможности системы "1С:Предприятие" не ограничиваются решением задач для автоматизации документооборота. В этой связи особо подчеркивается, что в системе "1С:Предприятие" можно использовать абсолютно разные технологии программирования, которые активно используются программистами в других средах и языках программирования.

Книга не является традиционным учебником по программированию в среде "1С:Предприятие". Она сосредотачивает внимание читателя на неявных, скрытых, но не менее важных ее возможностях, а также возможностях операционной системы и разнообразных COM-объектов.

В книге содержится большое количество примеров, демонстрирующих варианты использования различных приемов и технологий программирования. Причем примеры рассматриваются для обеих версий системы (7.7 и 8.0).

Структура книги

Глава 1 "Применение технологий COM и ActiveX". В этой главе обсуждаются основы технологий COM и ActiveX. Здесь можно найти краткий обзор их возможностей, а также получить всю необходимую информацию по использованию

этих технологий. Материал этой главы будет особенно полезен начинающим разработчикам, которые пока еще не работали с COM и ActiveX.

Теоретическая часть главы подкрепляется большим количеством примеров использования данных технологий. В частности, рассматриваются приемы работы с Internet Explorer, Microsoft Winsock, ActiveBarCode, Windows Media Player ActiveX, Microsoft Agent и многие другие.

Глава 2 "Администрирование Windows средствами WMI". В этой главе рассматривается, каким образом возможно автоматизировать службу системного администратора средствами Windows Management Instrumentation (WMI). В этой связи показывается, как выполнять различные операции с учетными записями пользователей; доменами; операционной системой; файлами; системным реестром; оборудованием, входящим в локальную вычислительную сеть. Кроме этого, здесь также можно будет узнать о дополнительных функциях, включенных в интерфейс WMI.

Глава 3 "Использование скриптов WSH". Данная глава посвящена рассмотрению возможностей стандартного компонента операционной системы Windows Script Host (WSH). В связи с этим, в ней подробно рассмотрена объектная модель WSH, с помощью которой можно работать с файловой системой, системным реестром, специальными папками и ярлыками Windows, ресурсами локальной сети, а также запускать процессы и контролировать ход их выполнения.

Если даже читатель уже знаком с интерфейсом WSH, рекомендуется прочитать данную главу, чтобы выяснить все возможности интерфейса и закрепить знания на подробно разобранных примерах.

Глава 4 "Использование OLE Automation". Глава посвящена практическому использованию технологии OLE Automation. В ней приводится обзор самых популярных OLE-серверов, таких как Microsoft Word, Excel, PowerPoint и Adobe Photoshop. Помимо этого рассматриваются возможности использования системы "1С:Предприятие" версий 7.7 и 8.0 в качестве OLE Automation сервера, а также приемы работы с COM-соединением в версии 8.0.

Помимо практической части в главе раскрываются понятия OLE-объекта, контейнера приложения, сервера приложения и другие основополагающие термины и принципы обработки данных с помощью технологии OLE Automation.

Глава 5 "Использование ADO". Эта глава книги раскрывает вопросы интеграции разрабатываемых конфигураций с произвольными базами данных при помощи технологии Microsoft ActiveX Data Objects (ADO).

В ней рассматривается объектная модель интерфейса ADO, используемая для соединения с источником данных для организации их чтения, добавления, удаления и модификации.

Помимо ADO в главе содержится описание технологии ADO Extensions for Data Definition and Security (ADOX), которая представляет собой набор

объектов, позволяющих манипулировать метаданными в базах данных и управлять объектами, отвечающими за безопасность системы.

Глава 6. "Использование SQL-DMO". Здесь описывается еще одна технология доступа к базам данных Microsoft SQL Server с помощью технологии SQL Distributed Management Objects (SQL-DMO).

В ней приводятся примеры манипуляции с базами данных Microsoft SQL Server, таблицами, представлениями, хранимыми процедурами и SQL-запросами. Также приводятся примеры работы с сервисными возможностями Microsoft SQL Server, такими как резервирование и восстановление баз данных и настройка ограничений доступа к данным.

Глава 7 "Команды управления Windows, использующие rundll32". В этой главе читатель узнает возможности использования утилиты командной строки rundll32, которая позволяет запускать некоторые команды, заложенные в DLL-файлах.

Помимо этого рассматриваются примеры запуска элементов панели управления, работа с сетью Интернет, принтерами и многое другое.

Глава 8 "Интернет-технологии". Данная глава посвящена использованию интернет-технологий в конфигурациях, разрабатываемых в среде системы "1С:Предприятие" версий 7.7 и 8.0.

Также здесь рассматриваются практические примеры работы с электронной почтой, включая как объекты доступа к почтовым ящикам, посредством интерфейса Messaging Application Program Interface (MAPI), так и объекты, предназначенные для прямой работы с почтовыми серверами.

Помимо этого рассматриваются различные приемы работы с протоколом FTP.

Глава 9 "Использование Active Directory Service Interfaces". Это последняя глава книги, в которой рассматриваются вопросы работы с технологией Active Directory Service Interfaces (ADSI).

В ней приводятся примеры манипуляции службой WinNT Provider, которая позволяет работать с доменами, рабочими станциями, пользователями и группами локальной сети.

Программные требования

Для запуска приведенных в книге практических примеров необходима установка системы "1С:Предприятии" версий 7.7 или 8.0. При этом рекомендуется использовать операционную систему Microsoft Windows 2000 или XP.

Для выполнения отдельных примеров необходимо наличие установленных приложений или некоторых COM-объектов, которые располагаются на прилагаемом к данной книге компакт-диске.

Описание сопроводительного CD-ROM

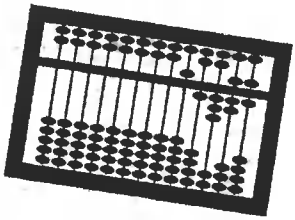
В прилагаемом CD-ROM содержатся все исходные тексты примеров, а также COM-объекты, которые рассмотрены в книге. Помимо примеров, на компакт-диске содержится Offline-версия интернет-проекта ERP Group (**www.erpg.ru**), основателем которого является автор книги.

Источники информации

При написании книги использовалось множество интернет-ресурсов, основными из которых являются:

- **http://www.osp.ru** — издательство "Открытые системы";
- **http://msdn.microsoft.com** — Microsoft Developer Network;
- **http://rusproject.narod.ru** — "Русский проект", автор А. Климов;
- **http://admtech.mrsu.ru** — Мордовский государственный университет им. Н. П. Огарева "Технологии администрирования Windows";
- **http://avb1c.narod.ru** — домашняя страница Алексея Бажитова;
- **http://www.erpg.ru** — проект Андрея Михайлова "ERP Group".

Автор книги выражает искреннюю благодарность авторам данных информационных ресурсов.



Глава 1

Применение технологий COM и ActiveX

Технологии COM и ActiveX

Аббревиатура *COM* расшифровывается достаточно просто. По существу это краткая запись понятия — Component Object Model (компонентная объектная модель). Иногда говорят — "модель COM".

Построение компонентной объектной модели осуществляется при помощи соответствующей информационной технологии. Суть ее заключается в том, что программы строятся из компонентов, которые в свою очередь состоят из объектов. Эти компоненты и объекты могут представлять собой непосредственно исполняемый двоичный код или файлы (DLL, EXE), которые никак не надо "связывать" с проектом, для которого они построены. Их достаточно зарегистрировать в операционной системе и они будут доступны любой программе, исполняющейся на данной машине. Таким образом, использование их в программе производится без применения операций сборки модуля. Более того, модель COM позволяет вызывать объекты какого-либо компонента из своей программы без указания того, где они расположены. Здесь достаточно знать только имя объекта.

Объекты COM можно создавать на любом языке, способном поддерживать данный интерфейс. После этого обращаться к методам этих объектов можно будет на любом языке, в том числе и на встроенном языке системы *"1С:Предприятие"*, позволяющем осуществлять вызовы данного двоичного интерфейса. Ни объект, ни его клиент не знают (да и зачем им это) — на каком языке написаны другие.

Технология COM (в дальнейшем просто — COM) разработана Microsoft, а поэтому первоначально была доступна для программ, работающих под управлением Microsoft Windows 9*x* и Microsoft Windows NT. Теперь Microsoft предоставляет поддержку COM и для Macintosh. Хотя Microsoft не поддерживает COM на других операционных системах, этот пробел заполнен

третьими фирмами. Несколько компаний, больших и малых, предоставляют реализации COM и основанных на ней технологий для широкого диапазона операционных систем.

Технология *ActiveX* построена на основе компонентов COM. ActiveX — это технология, разработанная фирмой Microsoft с целью стандартизации программных компонент. Это системная технология, представляющая совокупность средств, с помощью которых объекты, разработанные различными разработчиками на разных языках программирования и работающие в разных средах могут взаимодействовать друг с другом без какой-либо модификации их исполняемых модулей (двоичных кодов).

Все ActiveX-компоненты можно разделить на два типа:

- визуальные (например, `Microsoft Agent`, `Progress Control`, `Slider Control`, календарь, таймер и пр.);
- не воспринимаемые визуально, но обеспечивающие какую-либо функцию интеграции разных по происхождению компонент.

Преимущества использования технологии ActiveX заключаются в следующем:

- *быстрое написание программного кода*, которое выражается в том, что разработка приложений становится очень похожей на программирование, в котором используются отдельные "программные кирпичики" (элементы ActiveX);
- *открытость и мобильность*, являющиеся следствием спецификации данной технологии, недавно переданной в Open Group, как основы открытого стандарта (кроме того, Microsoft совместно с компаниями Metrowerks и Bristol заканчивает реализацию технологий ActiveX для платформ Macintosh и UNIX);
- *возможность написания приложений с использованием любых средств разработки*, выражающееся в том, что программные элементы ActiveX могут использоваться в Visual Basic, Visual C++, Borland Delphi, Borland C++, 1С:Предприятие и других средствах разработки приложений;
- *большое количество уже существующих программных элементов ActiveX*, причем некоторые из разработанных компонентов являются бесплатными и могут применяться в приложениях независимых разработчиков;
- *стандартность*, выражающаяся в том, что технология ActiveX основана на широко используемых стандартах Интернета (TCP/IP, HTML, Java), с одной стороны, и стандартах, введенных в свое время Microsoft и необходимых для сохранения совместимости с COM и OLE, с другой.

Как ни странно, при рассмотрении проблемы расширения встроенного языка системы "1С:Предприятие", речь заходит только о внешних компонентах, в основу которых положена идеология COM, но которые должны быть написаны по единому стандарту "1С". Следствием этого является невозмож-

ность использования компонентов, которые написаны в средах других языков программирования. В связи с этим, в данной главе будут рассмотрено большое число возможностей использования стандартных COM- и ActiveX-объектов в системе "1С:Предприятие".

В системе "1С:Предприятие" (версия 7.7) новый экземпляр COM-объекта создается с помощью конструкции `Идентификатор = СоздатьОбъект("ИмяОбъекта")`. Для версии 8.8 оператор `СоздатьОбъект` заменяется на `Новый COMОбъект`. Методы и свойства COM-объектов в дальнейшем становятся доступными через созданный объект.

Работа с объектом Internet Explorer

COM-объект, предоставляемый браузером Microsoft Internet Explorer, позволяет выполнять множество функций. Например, получение данных с различных Web-сайтов и использование полученной информации в своих целях, автоматическая регистрация сайта во всех поисковых системах сразу, решение задач посылки GET и POST-запросов и др. Поскольку данный браузер присутствует в любой современной операционной системе семейства Windows, то использование данного компонента является одним из самых универсальных способов получения сторонних HTML-данных.

Рассмотрим ряд приемов использования данного COM-объекта.

Определение текущего разрешения экрана

Для того чтобы определить текущее экранное разрешение, находясь в браузере, необходимо запустить объект `Internet Explorer` и определить, как это показано в листинге 1.1, текущее разрешение.

Листинг 1.1. Определение текущего разрешения экрана для версии 7.7

```
objIE = СоздатьОбъект("InternetExplorer.Application");
objIE.Visible = 0; // Отключаем видимость
objIE.Navigate("about:blank"); // Загружаем пустую страницу
Пока objIE.readyState < 4 Цикл // Ждем пока она загрузится
КонецЦикла;
oScreen = objIE.document.parentWindow.screen;
Сообщить("Текущее разрешение экрана: " + oScreen.width + "x" +
oScreen.height);
```

Приведенный код работает в системе "1С:Предприятие 7.7". Для версии 8.0 необходимо в первой строке заменить "`СоздатьОбъект`" на "`Новый COMОбъект`".

Внимание

Если в цикле ожидания готовности объекта `objIE` условие `objIE.readyState < 4` заменить на `objIE.busy` (распространенная ошибка), то в Windows 9*x* приложение будет завершаться с ошибкой примерно один раз из четырех.

Несмотря на всю привлекательность использования Internet Explorer, вышеописанный подход имеет и значительные недостатки, основной из которых — это существенное понижение общей производительности приложения. Для более эффективного определения текущего разрешения экрана рекомендуется использовать объекты WSH или WMI, о которых будет подробно рассказано в последующих главах.

Анализ HTML-страниц

Среди множества методов и свойств Microsoft Internet Explorer есть весьма ценное свойство `Document`, которое позволяет получить доступ к содержимому объектной модели HTML документа (то есть возвращает ссылку на тот же объект, что и строка клиентского JScript: window.document). Это позволяет в свою очередь легко и просто разбирать HTML-документы безо всяких регулярных выражений.

Рассмотрим примеры использования Internet Explorer для анализа всех ссылок на HTML-странице, получая информацию с сайта автора **www.erpg.ru** (листинги 1.2 и 1.3). Данную информацию можно использовать, например, для автоматического определения наличия необходимой ссылки на свой сайт при обмене кнопками и баннерами (рекламными блоками) с дружественными сайтами.

Листинг 1.2. Получение всех ссылок на сайте для версии 7.7

```
objIE = СоздатьОбъект("InternetExplorer.Application");
objIE.Visible = 0;
objIE.Silent = 0;
objIE.Navigate(Адрес);
Пока objIE.readyState<4 Цикл // Ждем пока она загрузится
КонецЦикла;
objDoc = objIE.Document;
AllTags = objDoc.all.tags("A"); // Получаем всю коллекцию ссылок
Для Ном = 0 По AllTags.length-1 Цикл
  Сообщить(AllTags.item(Ном).href);
КонецЦикла;
objIE.Quit(); // Закрываем приложение
objIE = "";
```

Листинг 1.3. Получение всех ссылок на сайте для версии 8.0

```
objIE = Новый COMОбъект("InternetExplorer.Application");
objIE.Visible = Ложь; // Отключаем видимость
objIE.Silent = Ложь;
objIE.Navigate(Адрес);
Сообщить("Идет загрузка страницы. Ждите...");
Пока objIE.readyState<4 Цикл // Ждем пока загрузится страница
КонецЦикла;
objDoc = objIE.Document;
Для каждого a Из objDoc.all.tags("A") Цикл // Перебираем все ссылки
Сообщить(a.href);
КонецЦикла;
objIE.Quit(); // Закрываем приложение
objIE = Неопределено;
```

Существует несколько важных отличий в реализации данной задачи на разных версиях системы "1С:Предприятие".

1. Создание COM-объекта в версии 7.7 происходит с помощью конструкции `СоздатьОбъект`, а в версии 8.0 — `Новый COMОбъект`.
2. Конструкция `objDoc.all.tags("A")` возвращает коллекцию COM-объектов с ссылками HTML-страницы, заключенные в тег <A>, но версия 7.7 не позволяет напрямую работать с подобными коллекциями. Поэтому для этой версии сначала определяется количество элементов в коллекции с помощью структуры `AllTags.length`, а только потом каждый элемент получаем методом `item(<НомерЭлемента>)`. В версии 8.0 обход элементов любой коллекции происходит с помощью конструкции `Для каждого...Из...Цикл...КонецЦикла`, причем каждый полученный элемент цикла уже будет соответствовать очередному элементу коллекции COM-объектов.

Работа с POST-запросами в HTML-формах

Данный тип HTTP-запросов чаще всего используется для транслирования сторонней информации в тех случаях, когда для доступа к ней требуется либо идентификация прав доступа (предварительный ввод имени пользователя и пароля), либо автоматическое заполнение определенной HTML-формы и отправки этой информации куда-либо, например, на форум.

Обычно, для передачи данных POST-запроса, во всех популярных языках программирования применяется метод `Navigate` объекта InternetExplorer со

специальным параметром `PostData`. Однако, несмотря на это, его практическое использование в среде "1С:Предприятие" сопряжено с некоторыми трудностями. Дело в том, что данный аргумент должен иметь тип — указатель на структуру `SafeArray`, который не поддерживается системой "1С:Предприятие" версии 7.7.

Поэтому самым простым способом отправить POST-запрос посредством `InternetExplorer.Application` можно считать "эмуляцию" нажатия кнопки **Submit** (управляющая кнопка, используемая для передачи данных формы на сервер) на какой-либо реальной HTML-форме. Этот способ особенно интересен с учетом того, что многие скрипты, при обработке POST-запросов, как раз для защиты от подобной автоматизации, учитывают HTTP_REFERER, т. е. с какой страницы пришел запрос, и (или) выставляют специальные сессионные переменные или cookie на той странице, где находится форма, посылающая POST-запрос.

Листинги 1.4 и 1.5 демонстрируют примеры подобного программного эмулирования "интерактивного пользователя". При этом программа, написанная для версии 7.7, приведенная в листинге 1.4, демонстрирует заполнение HTML-формы (рис. 1.1) форума, расположенного по адресу **http://forum.erpg.ru**.

Рис. 1.1. Элементы HTML-формы форума

Листинг 1.4. Заполнение HTML-формы сообщения на форуме для версии 7.7

```
objIE = СоздатьОбъект("InternetExplorer.Application");
objIE.Visible = 0;
objIE.Silent = 0;
objIE.Navigate("http://erpg.ru/forum/posting.php?mode=reply&t=34");
Пока objIE.readyState<4 Цикл // Ждем пока она загрузится
КонецЦикла;
objDoc = objIE.Document;
objForm = objDoc.forms(0); // Получаем первую форму в документе
```

```
Попытка
  // Заполняем поле формы "Имя пользователя"
  objForm.elements("username").value = СокрЛП(Имя);
Исключение
КонецПопытки;
objForm.elements("subject").value = СокрЛП(Тема);
objForm.elements("message").value = СокрЛП(Описание);
objIE.Visible = 1;
Сообщить("HTML-форма форума заполнена. Для отправки сообщения необходимо
нажать кнопку Отправить в Internet Explorer.");
```

К сожалению, версия 7.7 не позволяет выполнить эмуляцию нажатия определенной кнопки HTML-формы из-за того, что система не опознает метод `click()` элемента формы, поэтому для окончательной отправки сообщения на форум пользователю необходимо вручную нажать на кнопку **Отправить** в окне Internet Explorer. Но если на форме имеется только одна кнопка, используемая по умолчанию, то для программной эмуляции ее нажатия можно использовать метод `submit`: `objForm.submit()`.

Версия 8.0 системы "1С:Предприятие" более корректно работает с COM-объектами, поэтому процедура отправки сообщения на форум полностью автоматизирована. Рассмотрим процедуру автоматического заполнения той же формы форума и эмуляции нажатия кнопки **Отправить** (см. листинг 1.5).

Листинг 1.5. Отправка сообщений на форум (метод `Post`) для версии 8.0

```
objIE = Новый COMОбъект("InternetExplorer.Application");
objIE.Visible = Ложь;  // Отключаем видимость
objIE.Silent = Ложь;
objIE.Navigate("http://erpg.ru/forum/posting.php?mode=reply&t=34");
Сообщить("Идет загрузка страницы. Ждите...");
Пока objIE.readyState<4 Цикл // Ждем пока загрузится сраница
КонецЦикла;
objDoc = objIE.Document; // Получаем объект HTML-документ
objForm=objDoc.forms(0); // Получаем первую форму в документе
Попытка
  // Заполняем поле формы "Имя пользователя"
  objForm.elements("username").value = Имя;
Исключение
КонецПопытки;
objForm.elements("subject").value = Тема;
```

```
objForm.elements("message").value = Описание;
objForm.elements("post").click(); // Нажимаем на кнопку "Отправить"
Сообщить("Ваше сообщение отправлено на форум по адресу: " +
"http://erpg.ru/forum/viewtopic.php?t=34");
objIE.Quit(); // Закрываем приложение
objIE = Неопределено;
```

По-моему приведенный код должен быть понятен без дополнительных объяснений, за исключением использования конструкции `Попытка...Исключение...КонецПопытки` при заполнении имени пользователя на форме (элемент с именем `username`). Это делается для того, чтобы предотвратить ошибку системы в том случае, если поле `username` не существует. На форуме используется система авторизации пользователей, и если пользователь был авторизирован ранее, то поле `username` не появляется на форме.

Работа с объектом "Microsoft Winsock"

Microsoft Winsock — это объект ActiveX, который обеспечивает простой доступ к *TCP* или *UDP* сетевым операциям. Чаще всего его используют для того, чтобы подсоединиться к удаленному компьютеру и обмениваться данными в обоих направлениях.

Объект Winsock имеет следующие свойства:

- `ByteReçived` — возвращает количество полученных данных, находящихся в данный момент в принимающем буфере;
- `LocalHostName` — возвращает имя местного компьютера;
- `LocalIP` — возвращает IP-адрес местного компьютера в формате (xxx.xxx.xxx.xxx);
- `LocalPort` — возвращает или устанавливает местный порт использования;

Примечание

Для клиента порт используется для отправки данных. Укажите порт 0, если приложение не требует специфического порта. В этом случае элемент управления выберет случайный порт. После установления соединения этот местный порт используется для TCP-соединений. Для сервера этот порт является принимающим. Если указан порт 0, будет использоваться случайный порт. После вызова функции `Listen` свойство содержит порт, который был выбран.

- `Protocol` — возвращает или устанавливает протокол, используемый Winsock (либо TCP, либо UDP);
- `RemoteHost` — возвращает или устанавливает имя или IP-адрес удаленного компьютера, с которым нужно обмениваться данными (можно указать IP-адрес, имя компьютера или FTP-адрес);

- RemoteHostIP — возвращает IP-адрес удаленного компьютера;
- RemotePort — возвращает или устанавливает порт удаленного компьютера для соединения (порты, устанавливаемые по умолчанию: 80 — HTTP, 26 — FTP);
- State — возвращает состояние элемента управления в числовом выражении.

Определение IP-адреса локального компьютера

Для определения IP-адреса локального компьютера используется свойство LocalIP объекта Winsock. В листинге 1.6 приведен фрагмент программы, показывающий вариант реализации этого действия.

Листинг 1.6. Определение IP-адреса локального компьютера

```
Winsock = Новый COMОбъект("MSWinsock.Winsock");
Сообщить(Winsock.LocalIP);
Winsock = Неопределено;
```

Определение IP-адреса удаленного компьютера

Для определения IP-адреса удаленного компьютера используется свойство RemoteHostIP объекта Winsock. В листинге 1.7 приведен фрагмент программы, показывающий вариант реализации этого действия.

Листинг 1.7. Определение IP-адреса удаленного компьютера

```
Winsock = Новый COMОбъект("MSWinsock.Winsock");
Winsock.Connect(ИмяСервера, 139);
Пока Winsock.State = 4 Цикл // Ждем пока произойдет соединение
КонецЦикла;
Сообщить(Winsock.RemoteHostIP);
Winsock = Неопределено;
```

Код довольно простой, но в нем есть некоторые особенности.

- Для TCP-соединения с удаленным сервером используется порт 139, который, как правило, доступен всегда.
- Метод Connect является асинхронным, то есть возврат из него осуществляется сразу же, не дожидаясь установления соединения.
- Для ожидания соединения с удаленным компьютером в цикле анализируется состояние соединения State.

Создание штрихкодов с использованием ActiveX *ActiveBarCode*

Для эффективного учета движения товаров в организации необходима технология, позволяющая присваивать каждому товару уникальный код и обеспечивать быстрое его считывание при минимальных ошибках. Именно этим условиям и удовлетворяет технология штрихового кодирования товаров. На данный момент насчитывается около 20 форматов штрихкодов (включая их модификации), различающихся, в основном, областями применения. Некоторые форматы имеют механизм контроля корректности, заключающийся в вычислении одной части кода по другой.

Безусловно, система "1С:Предприятие" позволяет использовать технологию штрихкодирования при автоматизации предприятия с использованием входящего в поставку многих конфигураций ActiveX-элемента — `ActiveBarCode`. Он позволяет формировать образ штрихкода непосредственно в "1С:Предприятии". Этот элемент поддерживает практически все известные форматы штриховых кодов, управление цветом и шрифтом выводимого штрихкода, автоматическое определение формата кода, а также вычисление контрольных цифр.

Компонент `ActiveBarCode` имеет множество свойств, методов и событий. Рассмотрим основные свойства, которые чаще всего используются:

- `Text` — текст выводимый под штрихкодом (обычно номер);
- `Type` — тип штрихкода (полный список кодов можно найти на сайте производителя **http://www.activebarcode.com**);
- `AutoType` — признак автоматического определения типа исходя из номера штрихкода;
- `ShowText` — признак включения или отключения вывода текста под штрихкодом;
- `ForeColor` — цвет текста штрихкода;
- `BackColor` — цвет фона;
- `Font` — шрифт текста штрихкода.

Фактически данный компонент представляет из себя файл Barcode.ocx, который может располагаться где угодно, но перед использованием его обязательно необходимо зарегистрировать в системе командой `regsvr32 Barcode.ocx`.

`ActiveBarCode` является визуальным компонентом, т. е. его можно разместить в диалоге или таблице. Однако следует заметить, что использование данного ActiveX-компонента несколько отличается в разных версиях системы "1С:Предприятие". Основное различие состоит в том, что в версии 7.7 невозможно размещать ActiveX-компоненты на формах (разрешено только в табличных документах). В восьмой версии таких ограничений нет.

Есть еще одна крайне неприятная особенность версии 7.7. Она заключается в том, что в этой версии нельзя штатными средствами вставить данный элемент в табличный документ. Чтобы все-таки это сделать, необходимо вставить `ActiveBarCode` в любом другом приложении (например в MS Word, "1С:Предприятие" 8.0 и др.), а потом через буфер обмена перенести его в табличный документ 7.7. Далее, для динамического вывода нужного номера на штрихкоде надо в свойствах данного объекта прописать необходимые свойства объекта (рис. 1.2).

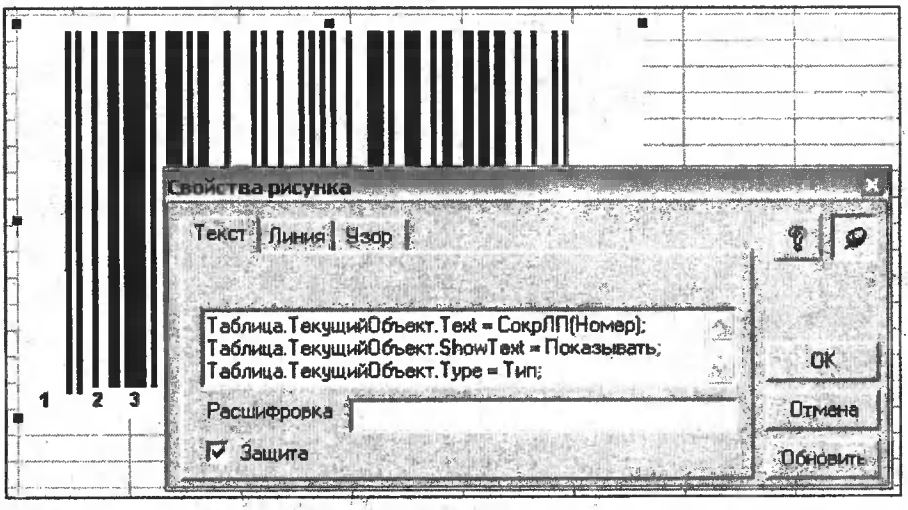


Рис. 1.2. Настройка параметров динамического формирования `ActiveBarCode`

Для вывода табличного документа используется стандартный код "1С:Предприятия" (листинг 1.8).

Листинг 1.8. Вывод табличного документа для версии 7.7

```
Таблица.ИсходнаяТаблица("Таблица");
Таблица.ВывестиСекцию("ШтрихКод");
Таблица.Показать("Штрихкод");
```

В данном примере предполагается, что:

1. Элемент `ActiveBarCode` в табличном документе находится в секции `ШтрихКод`.
2. Переменные `Номер`, `Тип` и `Показывать`, введенные в свойствах элемента, объявляются и заполняются в коде до вывода таблицы (например, из диалога или справочника).

В версии 8.0 системы "1С:Предприятие" все намного проще. В этом случае ActiveX-элементы можно устанавливать как на форму (**Форма | Вставить ActiveX** и затем выбрать `ActiveBarCode`), так и в табличный документ (**Таблица | Рисунки | Вставить объект**).

Так, если элемент `ActiveBarCode` находится на форме и имеет имя `ШтрихКод`, то для его формирования достаточно написать код, приведенный в листинге 1.9.

Листинг 1.9. Формирование штрихкода средствами `ActiveBarCode` для версии 8.0

```
ЭлементыФормы.ШтрихКод.Type = Тип;
ЭлементыФормы.ШтрихКод.Text = Номер;
ЭлементыФормы.ШтрихКод.ShowText = Показывать;
ЭлементыФормы.ШтрихКод.AutoType = Автоопределение;
```

В результате выполнения этого кода на форме появится необходимый номер со штрихкодом (рис. 1.3).

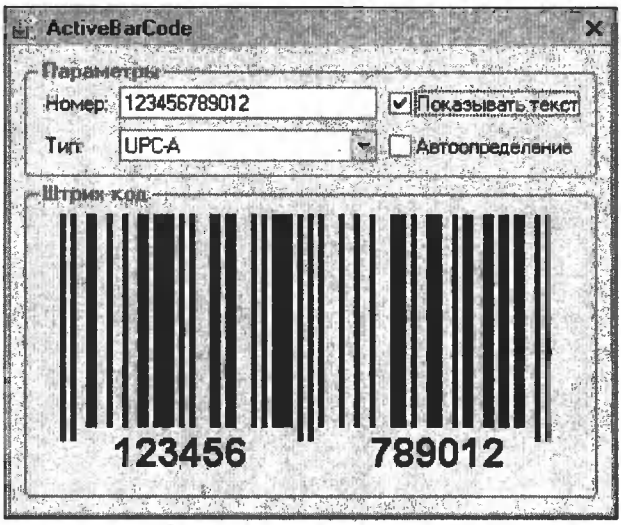


Рис. 1.3. Результат формирования штрихкода для версии 8.0

Использование ActiveX Windows Media Player

ActiveX-компонент Microsoft Windows Media Player позволяет просматривать видео, прослушивать аудиофайлы, а также просматривать рисунки различных форматов (JPG, BMP и др.). Данный компонент бывает очень полезен, когда необходимо просмотреть определенную мультимедийную информацию о товаре или поставщике.

Существует несколько версий Windows Media Player. Для определенности будем рассматривать версию 9.0, которую можно загрузить на сайте производителя **http://www.microsoft.com/windows/windowsmedia/download/default.asp**.

Примечание

К сожалению, в "1С:Предприятие" версии 7.7 невозможно использование данного компонента, поэтому все дальнейшее рассмотрение будет актуально только для восьмой версии платформы.

Для работы Windows Media Player в системе "1С:Предприятие" необходимо выполнить всего два условия.

1. Для размещения ActiveX-компонента на форме в меню **Форма** следует выбрать пункт **Вставить ActiveX**. В появившемся списке выбрать **Windows Media Player**.
2. Написать код, который бы передавал файл с мультимедийной информацией в ActiveX-компонент (листинг 1.10).

Листинг 1.10. Запуск мультимедиафайла в ActiveX Windows Media Player

```
Попытка
  ЭлементыФормы.WMP.settings.autoStart = True; // Автовоспроизведение
  ЭлементыФормы.WMP.URL = ИмяФайла;
Исключение
  Предупреждение(ОписаниеОшибки(), 10, "Ошибка");
КонецПопытки;
```

В данном примере применена конструкция `Попытка..Исключение..КонецПопытки` для предотвращения ошибок, связанных с отсутствием в системе данного компонента. Элемент формы `WMP` как раз и является ActiveX-компонентом.

С помощью ActiveX Windows Media Player можно также просматривать свойства файла (`tag`, `Bitrate` и пр.). В листинге 1.11 демонстрируется пример просмотра свойств произвольного файла.

Листинг 1.11. Получение свойств файла с помощью ActiveX Windows Media Player

```
Попытка
  ЭлементыФормы.WMP.settings.autoStart = False;
  ЭлементыФормы.WMP.URL = ИмяФайла;

  // Получаем объект, предназначенный для работы с выбранным файлом
```

```
  СМ = ЭлементыФормы.WMP.currentMedia;

  // Получение значения свойства по имени
  Сообщить("Полное имя файла: " + СМ.getItemInfo("SourceURL"));

  // Получение всего списка свойств
  Для Ном = 0 по СМ.attributeCount-1 Цикл
    ИмяСвойства = СМ.getAttributeName(Ном);
    ЗначениеСвойства = СМ.getItemInfo(СМ.getAttributeName(Ном);
    Сообщить(ИмяСвойства + " = " + ЗначениеСвойства);
  КонецЦикла;
Исключение
  Предупреждение(ОписаниеОшибки(), 10, "Ошибка");
КонецПопытки;
```

Использование помощников Microsoft Agent

Понятие MS Agent

Microsoft Agent — это набор программных средств, которые поддерживают воспроизведение интерактивных анимированных персонажей. Разработчики могут использовать эти персонажи для создания различных интерактивных приложений, справочных систем, в том числе и в системе "1С:Предприятие".

Также Microsoft Agent предлагает поддержку распознавания голосовой информации (на основе Microsoft Speech Application Programming Interface, SAPI 4.0), позволяя управлять приложениями на уровне голосовых команд. Персонажи могут воспроизводить синтезированную речь (text-to-speech, TTS), аудиофрагменты или текст.

Персонажи отображаются в собственных окнах, что делает интерфейс более удобным и гибким. В состав Microsoft Agent входит компонент ActiveX, доступный из системы "1С:Предприятие".

Эта технология напрямую используется в Office 2000 и включена в состав ядра Windows 2000. Для установки Microsoft Agent в Windows 98 и Windows NT 4.0 требуется загрузить соответствующее программное обеспечение с Web-узла фирмы Microsoft (**http://www.microsoft.com/msagent/**).

В настоящее время на сайте Microsoft доступны четыре стандартных персонажа, которые можно использовать. Это маг — Merlin, джин — Genie, робот — Robby и попугай — Peedy (рис. 1.4).

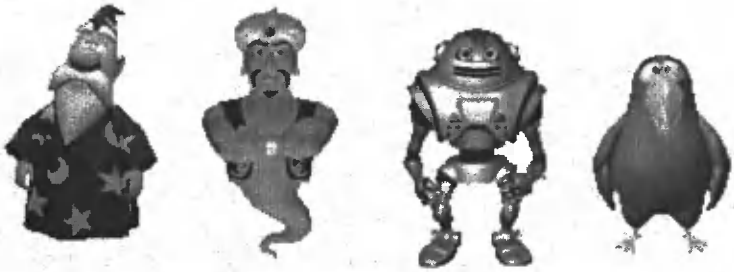


Рис. 1.4. Основные персонажи Microsoft Agent

Кроме стандартных персонажей в сети Интернет существуют персонажи, разработанные другими разработчиками на любой вкус.

Примечание

На том же сайте Microsoft можно загрузить Microsoft Text-To-Speech для того, чтобы "агенты" воспроизводили речь (русский голосовой модуль — Lernout&Hauspie TTS3000), а также редактор "агентов" и официальную документацию Microsoft. Так, при написании материала данной главы были использованы Microsoft Agent Control 2.0 и персонаж Merlin.

Практическое применение MS Agent

Microsoft Agent очень удобно применять в приложениях, где необходимо выводить какие-либо сообщения, подсказки и указания пользователю, но не хочется ограничиваться штатными методами системы. Microsoft Agent позволяет все это сделать красиво, удобно и необычно. Наверняка работать со столь дружественным интерфейсом, как показано на рис. 1.5, будет приятно любому пользователю.

В системе "1С:Предприятие" доступ к объекту Microsoft Agent создается с помощью конструктора `СоздатьОбъект("Agent.Control.2")`, для версии 7.7, и `Новый СОМОбъект("Agent.Control.2")`, для версии 8.0. После создания объекта минимальное, что нужно сделать, — это подключиться к созданному объекту, загрузить необходимый персонаж и вывести его на экран (листинг 1.12).

Листинг 1.12. Создание объекта MS Agent и вывод персонажа

```
Попытка
  // Устанавливаем ссылку на объект
  Agent = Новый СОМОбъект("Agent.Control.2");
```

```
Исключение
  Сообщить("MS Agent не установлен!");
  Сообщить("Загрузить MS Agent можно с сайта Microsoft —
http://www.microsoft.com/msagent/");
  Возврат;
КонецПопытки;

Agent.Connected = Истина; // Подключаемся к объекту
Agent.Characters.Load("Merlin", "Merlin.acs"); // Загружаем персонаж
Agent.Characters("Merlin").Show(); //  Выводим на экран
```

Рассмотрим код, приведенный в листинге 1.12, более подробно.

Прежде чем приступить к анимации персонажа, необходимо загрузить его, используя метод `Load`, и в качестве аргументов передать два параметра: идентификатор персонажа и полное имя файла персонажа или его HTTP-адрес. Microsoft Agent по умолчанию ищет персонажи в каталоге %WinDir%\MSAgent\Chars (где %WinDir% — это переменная окружения, которая указывает путь к каталогу операционной системы Microsoft Windows). Если файл вашего персонажа находится в другом месте, указывайте для него полный путь. Microsoft Agent поддерживает два формата для хранения данных доступных анимаций: одиночный файл (ACS) для локального применения и множественный формат (ACF, ACA), который хранит отдельные виды анимаций и используется для загрузки анимации отдельно через http-сервер.

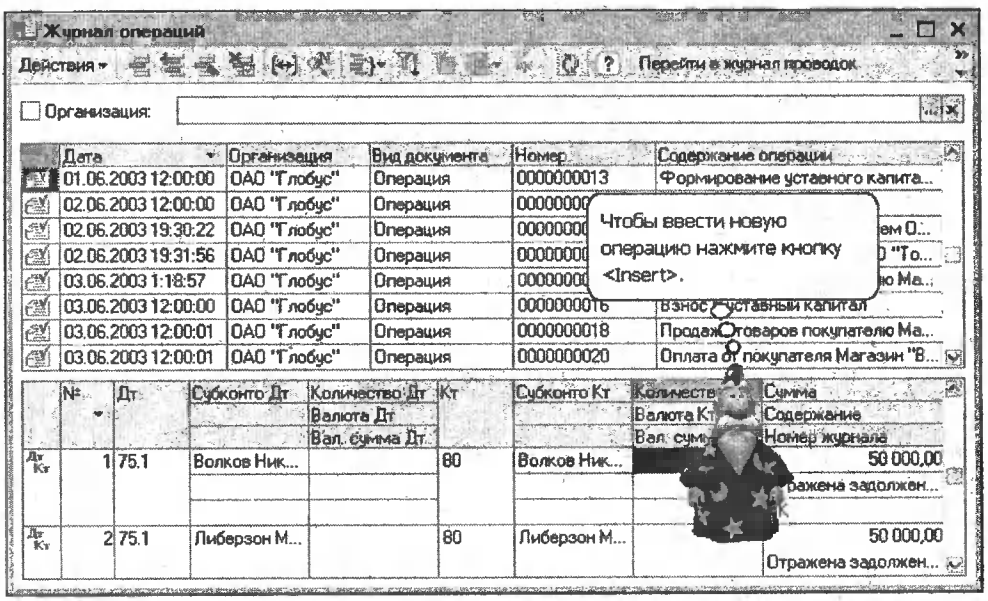


Рис. 1.5. Пример вывода сообщения Microsoft Agent

Приложение может загрузить только единственный экземпляр заданного персонажа. Но при этом можно использовать одновременно разные персонажи.

Метод `Show` выводит заданный персонаж на экран с одновременным проигрыванием соответствующей анимацией и не имеет параметров. Для скрытия персонажа существует метод `Hide`, который тоже параметров не имеет.

Вместо загрузки заданного персонажа, жестко определяя его имя, вы можете загрузить персонаж по умолчанию (листинг 1.13). Метод `ShowDefaultCharacterProperties` объекта `Agent` выводит панель настройки свойств для выбора персонажа по умолчанию. В качестве аргументов в метод можно (но не обязательно) передавать пару координат (X, Y), задающие горизонтальную и вертикальную координаты экрана в пикселах для вывода окна.

Листинг 1.13. Создание объекта MS Agent и вывод персонажа по умолчанию

```
Попытка
 Agent = Новый COMОбъект("Agent.Control.2");
Исключение
  Сообщить("MS Agent не установлен!");
  Возврат;
КонецПопытки;

Agent.Connected = Истина; // Подключаемся к объекту
// Вывод панели настройки свойств для выбора персонажа по умолчанию
Agent.ShowDefaultCharacterProperties(4, 4);
Agent.Characters.Load(,); // Загружаем персонаж по умолчанию
Agent.Characters("").Show(); //  Выводим на экран персонаж по умолчанию
```

После вывода персонажа на экран было бы очень хорошо "оживить" его, т. е. анимировать. Для анимации персонажа MS Agent предоставляет два метода: `MoveTo` и `Play`. Важно отметить, что эти методы не объекта Agent, а методы конкретного загруженного персонажа (листинг 1.14).

Листинг 1.14. Анимация персонажа MS Agent

```
Попытка
  Agent = Новый COMОбъект("Agent.Control.2");
Исключение
  Сообщить("MS Agent не установлен!");
```

```
  Возврат;
КонецПопытки;

Agent.Connected = Истина; // Подключаемся к объекту
Agent.Characters.Load("Merlin", "Merlin.acs"); // Загружаем персонаж
Characters = Agent.Characters("Merlin"); // Получаем ссылку на персонаж
Characters.Show(); //  Выводим на экран
// Перемещаем персонаж
Characters.MoveTo(100, 100); // Перемещаем в координаты (X, Y)
Characters.MoveTo(200, 0, 500); // 500 — скорость анимации
Characters.MoveTo(300, 100, 0); // 0 — без анимации
// Проигрываем заданную анимацию
Для Ном = 1 По 5 Цикл
  // Волшебник постучится в монитор
  Characters.Play("GetAttentionContinued");
КонецЦикла;
Characters.Play("Search"); // Волшебник покажет волшебный шар
```

Метод `MoveTo` перемещает персонаж в указанное место и имеет три параметра. Первые два являются обязательными и задают новые координаты персонажа (X, Y) относительно всего экрана. Третий параметр необязательный и означает время перемещения персонажа. По умолчанию используется 1000 миллисекунд. При использовании нулевого значения персонаж перемещается без анимации.

Метод `Play` проигрывает заданную анимацию для заданного персонажа и имеет всего один параметр — строку, содержащую имя анимации. Каждый персонаж имеет свой набор анимаций, поэтому перед использованием данного метода рекомендуется прочитать документацию к вашему персонажу. Для того чтобы получить весь список возможных имен анимаций для персонажа, существует метод `AnimationNames`, который возвращает коллекцию этих имен. К сожалению, версия 7.7 системы "1С:Предприятие" не позволяет получить все элементы этой коллекции. В версии 8.0 код вывода всех возможный анимаций персонажа будет выглядеть так, как это показано в листинге 1.15.

Листинг 1.15. Получение всех возможных имен анимаций персонажа MS Agent

```
Попытка
  Agent = Новый COMОбъект("Agent.Control.2");
Исключение
  Сообщить("MS Agent не установлен!");
```

```
  Возврат;
КонецПопытки;

Agent.Connected = Истина; // Подключаемся к объекту
Agent.Characters.Load("Merlin", "Merlin.acs"); // Загружаем персонаж
Characters = Agent.Characters("Merlin"); // Получаем ссылку на персонаж
// Получение всех возможных анимаций. Перебор элементов коллекции.
Для каждого Имя Из Characters.AnimationNames() Цикл
  Сообщить(Имя);
КонецЦикла;
```

Microsoft Agent выполняет все вызовы по порядку. Поэтому каждая новая команда начнет выполняться только после завершения предыдущей. Для того чтобы узнать, выполнена ли заданная команда, вы можете послать запрос объекту `Request`, который используется для получения информации о состоянии выполнения.

Microsoft Agent имеет также одну интересную особенность. Например, если статус персонажа не меняется несколько секунд, то Agent начинает самостоятельно проигрывать некоторые виды анимаций.

Для того чтобы остановить анимацию персонажа, применяется метод `Stop`, а для того чтобы остановить все (анимацию, перемещение, речь) — метод `StopAll`.

Помимо простой анимации Microsoft Agent имеет возможность общаться с пользователем, а именно: показывать текстовые подсказки, сообщения и даже воспроизводить речь.

Для того чтобы персонаж вывел какой-либо текст, существует метод `Think`, а для воспроизведения текста — метод `Speak` (листинг 1.16).

Листинг 1.16. Вывод и воспроизведение текстов персонажем MS Agent

```
Попытка
  Agent = Новый COMОбъект("Agent.Control.2");
Исключение
  Сообщить("MS Agent не установлен!");
  Возврат;
КонецПопытки;

Agent.Connected = Истина; // Подключаемся к объекту
Agent.Characters.Load("Merlin", "Merlin.acs"); // Загружаем персонаж
```

```
Characters = Agent.Characters("Merlin"); // Получаем ссылку на персонаж
Characters.Show(); //  Выводим на экран

Characters.Think("Добрый день, приятной работы!"); // Вывести текст
// Воспроизвести одну фразу из заданных
Characters.Speak("Хорошая погодка!|Не пора ли на обед?");
Characters.Speak(" ", "welcome.wav"); // Воспроизвести WAV-файл
```

Метод `Think` выводит заданный текст в особом окне в виде облачка (как в комиксах).

Метод `Speak` воспроизводит заданный текст или звуковой файл для заданного персонажа и имеет два параметра. Первый задает воспроизводимую строку и является необязательным. Второй — строка, содержащая местоположение звукового файла (WAV или LWV). Здесь указывается либо местоположение файла (абсолютный или относительный адрес), либо URL-адрес (если анимация персонажа реализуется через http-протокол).

Примечание

Хотя оба параметра являются необязательными, но один из них непременно должен присутствовать. Используйте пробел в первом параметре, чтобы воспроизвести звуковой файл без вывода текста.

Используйте вертикальную черту (|) в первом параметре в качестве разделителя между фразами. Это позволит выбирать в случайном порядке текст при каждом вызове операции. Можно также использовать специальные теги — Speech Output Tags, предназначенные для изменения голоса (громкость, тембр и т. п.). Пример использования тегов, характеристика которых приводится ниже, приведен в листинге 1.17.

- Тег `\Chr\` — определяет голос, каким произносится текст и может принимать значения `Normal` (нормальный голос), `Monotone` (монотонно) и `Whisper` (шепотом). По умолчанию Normal.
- Тег `\Ctx\` — определяет контекст произносимого текста. Параметр может принимать значения Address — адреса или телефонные номера, E-mail — адреса электронной почты и Unknown — неизвестный контекст. По умолчанию Unknown.
- Тег `\Lst\` — позволяет повторить текст, вызванный ранее.
- Тег `\Map\` — позволяет произносить один текст, а показывать другой.
- Тег `\Pau\` — позволяет выдерживать паузу в произносимой фразе. Длительность паузы задается в миллисекундах (`\Pau=1000\`).
- Тег `\Pit\` — позволяет управлять высотой голоса персонажа. Частота произносимого текста в герцах (`\pit=50\`).

- Тег \Spd\ — позволяет изменять скорость чтения текста. Число, указываемое как параметр, определяет количество произносимых слов в минуту (\Spd=100\).
- Тег \Vol\ — позволяет определять громкость произносимого текста. Параметр может принимать значения от 0 до 65 535 (\Vol=65535\).

Листинг 1.17. Использование тегов при воспроизведении текстов в MS Agent

```
// Текст, воспроизводимый шепотом
Characters.Speak("\Chr=""Whisper""\Тихо, всем спать...");
// Указываем, что воспроизводим электронный адрес
Characters.Speak("Пишите мне: \Ctx=""E-mail""\mav@erpg.ru");
// Повторяем предыдущий текст
Characters.Speak("\Lst\");
// Выводим один текст, а воспроизводим другой
Characters.Speak("\map=""Посетите наш сайт!""=""Добро пожаловать на сайт www.erpg.ru""\");
// Выдерживаем паузу между фразами
Characters.Speak("Привет!\Pau=1000\Почему не отвечаешь?");
// Плавно изменяем интонацию голоса
Characters.Speak("\pit=50\Не делай этого! \pit=300\Стой! \pit=500\Не надо!");
// Воспроизводим текст в медленном режиме
Characters.Speak("\Spd=100\Как я тебе завидую!");
// Воспроизводим текст с пониженной громкостью
Characters.Speak("\Vol=20000\Не ори, все спят!");
```

До сих пор мы рассматривали только методы объектов. Помимо методов MS Agent имеет еще свойства и события. На событиях останавливаться не будем, т. к. их невозможно создать средствами системы "1С:Предприятие". Рассмотрим основные свойства MS Agent.

- Name — возвращает имя персонажа (Джинн, Маг, Робби, Кеша);
- Description — возвращает описание персонажа;
- ExtraData — возвращает расширенные сведения о персонаже;
- Version — возвращает версию персонажа;
- Visible — возвращает 0, если персонаж скрыт методом Hide, и −1, если персонаж видим на экране;
- AutoPopupMenu — с помощью этого свойства можно включить или выключить показ контекстного меню, появляющегося при щелчке правой

кнопки мыши на персонаже или иконке в трэйбаре (возвращает текущее состояние свойства);

- `LanguageID` — возвращает или устанавливает языковой идентификатор для персонажа;
- `Height` — устанавливает или возвращает значение высоты кадра для заданного персонажа;
- `Width` — устанавливает или возвращает значение ширины кадра для заданного персонажа;
- `Left` — получает или устанавливает левый край кадра персонажа;
- `Top` — получает или устанавливает верхний край кадра персонажа;
- `OriginalHeight` — возвращает значение высоты, определенное как настоящее (оригинальное) для заданного персонажа;
- `OriginalWidth` — возвращает значение ширины, определенное как настоящее (оригинальное) для заданного персонажа;
- `TTSModeID` — возвращает или устанавливает идентификатор голосового движка Text-to-Speech (TTS) для персонажа.

В листинге 1.18 показан пример вывода свойств.

Листинг 1.18. Пример работы со свойствами персонажа MS Agent

```
Попытка
  Agent = Новый COMОбъект("Agent.Control.2");
Исключение
  Сообщить("MS Agent не установлен!");
  Сообщить("Загрузить MS Agent можно с сайта Microsoft -
http://www.microsoft.com/msagent/");
  Возврат;
КонецПопытки;

Agent.Connected = Истина; // Подключаемся к объекту
Agent.Characters.Load("Merlin", "Merlin.acs"); // Загружаем персонаж
Characters = Agent.Characters("Merlin"); // Получаем ссылку на персонаж
Characters.Show(); //  Выводим на экран

// Вывод основных свойств
Сообщить("Имя = " + Characters.Name);
Сообщить("Описание = " + Characters.Description);
Сообщить("Дополнительные сведения = " + Characters.ExtraData);
```

```
Сообщить("Версия = " + Characters.Version);
Сообщить("Видимость = " + Characters.Visible);
Сообщить("Показ контекстного меню  = " + Characters.AutoPopupMenu );

// Увеличиваем персонаж вдвое
Characters.Height = Characters.Height * 2;
Characters.Width = Characters.Width * 2;
// Выключаем показ меню
Characters.AutoPopupMenu = Ложь;
```

Рассмотрим еще один объект — `Balloon`.

Объект `Balloon` — это область вывода текстов персонажа, который имеет ряд свойств.

- `BackColor` — цвет фона;
- `BorderColor` — цвет рамки;
- `Enabled` — возвращает признак включения окна сообщений;
- `FontCharSet` — номер набора таблицы символов (например, 0 — ANSI);
- `FontName` — имя шрифта;
- `FontBold` — признак полужирного выделения шрифта;
- `FontItalic` — признак курсивного выделения шрифта;
- `FontSize` — размер шрифта;
- `FontUnderline` — признак подчеркивания текста;
- `ForeColor` — цвет текста;
- `NumberOfLines` — количество строк для вывода текста;
- `Style` — стиль окна сообщений;
- `Visible` — признак видимости окна сообщений.

Свойства `BackColor`, `BorderColor`, `FontBold`, `FontItalic`, `FontUnderline`, `ForeColor` являются свойствами только для чтения. Изменить их значения можно лишь на странице свойств Microsoft Agent, причем только одновременно для всех персонажей. Пример изменения свойств окна сообщений приведен в листинге 1.19.

Листинг 1.19. Пример изменения свойств окна сообщений MS Agent

```
Characters.Balloon.FontName = "Comic Sans MS";
Characters.Balloon.FontSize = 24;
Characters.Think("Добрый день, приятной работы!");
```

Интерактивное взаимодействие с MS Agent

Все, о чем говорилось ранее, так или иначе связано с односторонним общением персонажа с пользователем. Чтобы организовать интерактивное взаимодействие пользователя с MS Agent (организация предупреждения, ввода значений, выбора из списков и т. п.), необходимо прибегнуть к сторонним разработкам, т. к. сама по себе технология MS Agent не позволяет этого делать.

Существует множество подобных разработок, большая и лучшая часть которых являются коммерческими, хотя и не дорогими (обычно не более $10).

Для однозначности остановимся на разработке BalloonDialog 6.5 ActiveX компании SommyTech.

Примечание

Загрузить демо-версию BalloonDialog 6.5 можно на сайте производителя — http://www.sommytech.com.ar.

Компонент `BalloonDialog` позволяет:

- работать совместно с MS Agent;
- выводить модальные предупреждения (замена стандартного оператора `Предупредить`);
- выводить сообщения с флажком (замена стандартного оператора `Сообщить`);
- выводить вопросы с предложением различных вариантов ответов (замена стандартного оператора `Вопрос`);
- показывать диалог ввода текста (замена стандартного оператора `ВвестиСтроку`);
- выводить сложные диалоговые окна, состоящие из текстовых полей, полей для ввода текстов, комментариев, кнопок и радиокнопок одновременно.

Рис. 1.6. Вывод модального окна предупреждения `BalloonDialog` совместно с Microsoft Agent

Начнем по порядку. Приведем пример создания объекта `BalloonDialog`, вывода предупреждения и связи с персонажем MS Agent (листинг 1.20). На рис. 1.6 показан вид окна предупреждения.

Листинг 1.20. Интеграция BalloonDialog с MS Agent и вывод предупреждения

```
Попытка
  // Устанавливаем ссылку на объект
  Agent = Новый COMОбъект("Agent.Control.2");
Исключение
  Сообщить("MS Agent не установлен!");
  Сообщить("Загрузить MS Agent можно с сайта Microsoft -
http://www.microsoft.com/msagent/");
  Возврат;
КонецПопытки;

Попытка
  // Создаем объект BalloonDialog
  Balloon = Новый COMОбъект("BlnDialogT.Balloon");
  // Сброс всех установок свойств в значения по умолчанию
  Balloon.ResetProperties();
Исключение
  Сообщить("BalloonDialog не установлен!");
  Сообщить("Загрузить BalloonDialog можно с сайта SommyTech -
http://www.sommytech.com.ar");
  Возврат;
КонецПопытки;

Agent.Connected = Истина; // Подключаемся к объекту
Agent.Characters.Load("Merlin", "Merlin.acs"); // Загружаем персонаж
Characters = Agent.Characters("Merlin"); // Получаем ссылку на персонаж
Characters.Show(); //  Выводим на экран

// Вывод модального предупреждения
Balloon.MsgBalloon("Вы не имеете прав для входа в данный модуль!",,
"Ошибка!", Characters);
```

Создание объекта `Balloon` в версии 8.0 системы "1С:Предприятие" выполняется с помощью конструктора — `Новый COMОбъект("BlnDialog.Balloon")`, в версии 7.7 — `СоздатьОбъект("BlnDialog.Balloon")`. Если вы используете

демо-версию, то имя COM-объекта будет "BlnDialogT.Balloon", при этом каждый раз при инициализации будет появляться окно с информацией о производителе и просьбе зарегистрироваться.

Метод MsgBalloon выводит рядом с персонажем диалог и ждет, пока пользователь нажмет кнопку. Он возвращает целое число — номер, соответствующий нажатой кнопке и имеет четыре параметра. Первый параметр метода — строка сообщения; второй — целое число, определяющее набор кнопок диалога и тип отображаемой на поле диалога иконки; третий — строка заголовка диалога; четвертый — ссылка на персонаж MS Agent. Используя последний параметр, окно диалога автоматически позиционируется на экране в зависимости от положения персонажа. Кроме того, при перетаскивании персонажа пользователем диалог объекта Balloon перемещается вместе с ним.

С помощью этого же метода, как показано в листинге 1.21, можно заменить и стандартную функцию встроенного языка Вопрос (рис. 1.7).

Рис. 1.7. Вывод модального окна с вопросом BalloonDialog совместно с Microsoft Agent

Листинг 1.21. Вывод вопроса с помощью BalloonDialog

```
// Вывод модального вопроса
Ответ = Balloon.MsgBalloon("Вы действительно хотите закрыть обработку?",
36, "Внимание вопрос!", Characters);
Если Ответ = 6 Тогда
  Сообщить("Вы выбрали ответ ""Да""");
Иначе
  Сообщить("Вы выбрали ответ ""Нет""");
КонецЕсли;
```

По сравнению с предыдущим кодом в методе `MsgBalloon` появился второй параметр. Как уже было сказано, второй параметр означает целое число, определяющее набор кнопок диалога и тип отображаемой на поле диалога иконки. Точнее этот параметр определяется суммой двух чисел: кодом набора кнопок и кодом типа иконки диалога.

Например, число 36 (4+32) обозначает, что диалог будет содержать кнопки **Yes** и **No** (код 4) и отображаться иконка с вопросом (код 32).

Все возможные наборы кнопок, которые применяются в `BalloonDialog`, приведены ниже.

- `0` — отображается только кнопка **OK**.
- `1` — отображаются кнопки **OK** и **Cancel**.
- `2` — отображаются кнопки **Abort**, **Retry** и **Ignore**.
- `3` — отображаются кнопки **Yes**, **No** и **Cancel**.
- `4` — отображаются кнопки **Yes** и **No**.
- `5` — отображаются кнопки **Retry** и **Cancel**.

Кроме этого, в `BalloonDialog` допускается пять типов иконок диалога.

- `16` — отображается иконка **Ошибка**.
- `32` — отображается иконка **Вопрос**.
- `48` — отображается иконка **Предупреждение**.
- `64` — отображается иконка **Информация**.
- `80` — отображается произвольная иконка, определяемая свойствами `IconPicture` или `URLIconPicture`.

Как уже было сказано, метод `MsgBalloon` возвращает номер, соответствующий нажатой кнопке.

- `1` — если пользователь нажал кнопку **OK**.
- `2` — если пользователь нажал кнопку **Cancel**.
- `3` — если пользователь нажал кнопку **Abort**.
- `4` — если пользователь нажал кнопку **Retry**.
- `5` — если пользователь нажал кнопку **Ignore**.
- `6` — если пользователь нажал кнопку **Yes**.
- `7` — если пользователь нажал кнопку **No**.

Так как система "1С:Предприятие" в основном используется в России и с русским интерфейсом, то крайне желательно, чтобы все кнопки выводились на русском языке. Чтобы изменить названия кнопок, существует свойство `ButtonsCaptions`. Вообще говоря, с помощью этого свойства можно переименовывать кнопки абсолютно произвольным образом (рис. 1.8).

Рис. 1.8. Замена названий кнопок в сообщениях `BalloonDialog`

Рассмотрим пример вывода диалога с произвольными кнопками (листинг 1.22).

Листинг 1.22. Вывод вопроса с помощью `BalloonDialog` с произвольными кнопками

```
// Задаем произвольные названия для кнопок
Balloon.ButtonsCaptions="&ОК;&Сок;П&рервать;&Повторить;&Игнорировать;&Чай
;&Кофе";
// Выводим модальный вопрос
Ответ = Balloon.MsgBalloon("Предлагаю отдохнуть и перекусить!", 35, "Вни-
мание вопрос!", Characters);
Если Ответ = 6 Тогда
  Сообщить("Чай закончился!");
ИначеЕсли Ответ = 7 Тогда
  Сообщить("Кофе не закупили!");
ИначеЕсли Ответ = 2 Тогда
  Сообщить("Сок выпил кот!");
КонецЕсли;

// Вернем названия кнопок в более привычный вид, причем русский
Balloon.ButtonsCaptions="&ОК;О&тмена;П&рервать;&Повторить;&Игнорировать;&
Да;&Нет";
```

Символ `&` в заголовке кнопки ставится перед "горячим" или подчеркнутым символом для отработки комбинации <Alt>+<подчеркнутый символ>.

Для заметы стандартной функции "1С:Предприятия" ВвестиСтроку в BalloonDialog применяется метод InputBalloon (рис. 1.9).

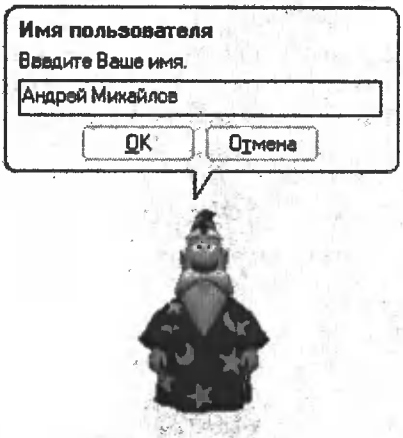


Рис. 1.9. Поле ввода текста в сообщениях BalloonDialog

Данный метод предназначен для формирования модального диалога при вводе текста и возвращает введенную пользователем строку (листинг 1.23).

Листинг 1.23. Вывод диалога BalloonDialog для ввода текста

```
// Диалог ввода текста
Имя = Balloon.InputBalloon("Введите Ваше имя.", "Имя пользователя",
"Андрей Михайлов", Characters);
// Сообщение, выводящее введенный текст
Balloon.MsgBalloon(Имя,, "Здравствуйте!", Characters);
```

Метод InputBalloon имеет четыре параметра. Первый — текст сообщения, второй — заголовок сообщения, третий — значение, подставленное в поле ввода по умолчанию, и четвертый — ссылка на персонаж.

Помимо довольно простых диалогов BalloonDialog позволяет конструировать довольно сложные диалоговые окна, включающие в себя сообщения, комментарии, кнопки, радиокнопки, поля ввода. Специально для этого существует объект FormBalloon.

Приведем пример, в котором пользователю предлагается диалог выбора одного из нескольких справочников; после выбора справочника должна открываться его форма списка (рис. 1.10).

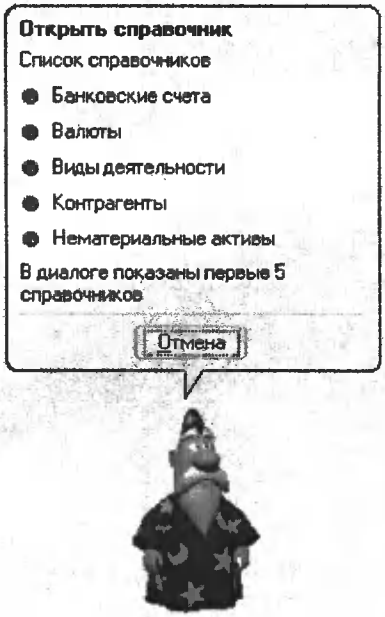


Рис. 1.10. Вывод сложных диалогов `BalloonDialog` с кнопками, радиокнопками и сообщениями

Для того чтобы код работал в произвольной конфигурации, немного усложним задачу и сделаем так, чтобы имена справочников и форм брались из метаданных конфигурации. А так как работа с метаданными и формами существенно различается в версиях 7.7 и 8.0, рассмотрим примеры работы в обеих версиях (листинги 1.24 и 1.25).

Листинг 1.24. Выбор справочника из радиокнопок в `BalloonDialog` для версии 7.7

```
// Добавление кнопки
Balloon.FormBalloon.Buttons.Add("&Отмена");

// Добавление радиокнопок
Для Ном = 1 По Мин(Метаданные.Справочник(), 5) Цикл
  Balloon.FormBalloon.OptionButtons.Add(Метаданные.Справочник(Ном). Представление());
КонецЦикла;

Balloon.FormBalloon.TextBox = 0; // Поле ввода не показываем
Balloon.FormBalloon.TBoxLines = 3; // Трехстрочный текст
```

```
// Текст по умолчанию
Balloon.FormBalloon.TBoxText = "Некоторый текст для ввода";

Balloon.FormBalloon.Title = "Открыть справочник"; // Заголовок диалога
Balloon.FormBalloon.Message = "Список справочников"; // Сообщение диалога
// Комментарий
Balloon.FormBalloon.Comment = "В диалоге показаны первые 5 справочников";
Balloon.FormBalloon.SepLine = 1; // Признак вывода разделителя

// Показать сложный диалог
Balloon.ShowFormBalloon(Characters);

// Если была выбрана радиокнопка, а не кнопка Отмена
Если Balloon.FormBalloon.ButtonPressed = 0 Тогда
  // Получаем номер выбранной позиции
  НомерПозиции = Balloon.FormBalloon.OptionPressed;
  // Из метаданных получаем имя справочника
  ИмяСправочника = Метаданные.Справочник(НомерПозиции).Идентификатор;
  ОткрытьФорму("Справочник." + ИмяСправочника); // Открываем форму
КонецЕсли;
```

Листинг 1.25. Выбор справочника из радиокнопок в BalloonDialog для версии 8.0

```
// Добавление кнопки
Balloon.FormBalloon.Buttons.Add("&Отмена");

// Добавление радиокнопок
Количество = 1;
Для каждого Справочник Из Метаданные.Справочники Цикл
  Balloon.FormBalloon.OptionButtons.Add(Справочник.Представление());
  Количество = Количество + 1;
  Если Количество = 6 Тогда
    Прервать;
  КонецЕсли;
КонецЦикла;

Balloon.FormBalloon.TextBox = Ложь; // Поле ввода не показываем
Balloon.FormBalloon.TBoxLines = 3; // Трехстрочный текст
```

```
// Текст по умолчанию
Balloon.FormBalloon.TBoxText = "Некоторый текст для ввода";

Balloon.FormBalloon.Title = "Открыть справочник"; // Заголовок диалога
Balloon.FormBalloon.Message = "Список справочников"; // Сообщение диалога
Balloon.FormBalloon.Comment = "В диалоге показаны первые 5 справочников";
// Комментарий
Balloon.FormBalloon.SepLine = Истина; // Признак вывода разделителя

// Показать сложный диалог
Balloon.ShowFormBalloon(Characters);

// Если была выбрана радиокнопка, а не кнопка Отмена
Если Balloon.FormBalloon.ButtonPressed = 0 Тогда
  // Получаем номер выбранной позиции
  НомерПозиции = Balloon.FormBalloon.OptionPressed;
  // Из метаданных получаем имя справочника
  ИмяСправочника = Метаданные.Справочники[НомерПозиции-1].Имя;
  // Получаем ссылку на форму
  Форма = Справочники[ИмяСправочника].ПолучитьФормуСписка();
  Форма.Открыть(); // Открываем форму
КонецЕсли;
```

Оба приведенных листинга различаются только работой с объектом метаданных. Все остальные методы почти идентичны.

Метод `Buttons.Add` объекта `FormBalloon` добавляет на форму кнопку. Метод `OptionButtons.Add` добавляет радиокнопку.

Метод `ShowFormBalloon` открывает модально сконструированный диалог и ждет пока пользователь сделает свой выбор. Данный метод имеет единственный параметр — ссылку на персонаж MS Agent.

После возвращения управления программой в "1С:Предприятие" становится доступно свойство `ButtonPressed`, которое может принимать два значения:

- `0` — диалоговое окно закрылось без нажатия на кнопку;
- `1` — диалоговое окно пользователь закрыл нажатием на какую-либо кнопку.

Для определения номера позиции выбранной пользователем радиокнопки используется свойство `OptionPressed`.

Работа с метаданными преднамеренно не описывается, т. к. это не является темой данной главы.

Вы так же можете изменить шрифт текстов, выводимых в диалоге. Для этого у объекта `Balloon` есть свойство `Font`:

- `Balloon.Font.Bold` — признак полужирного выделения текста;
- `Balloon.Font.Charset` — номер набора символов;
- `Balloon.Font.Italic` — признак выделения текста курсивом;
- `Balloon.Font.Name` — имя шрифта;
- `Balloon.Font.Size` — размер шрифта;
- `Balloon.Font.Strikethrough` — признак зачеркнутого текста;
- `Balloon.Font.Underline` — признак подчеркивания текста.

Создание HTML-редактора

Для создания полноценного HTML-редактора с возможностью визуального редактирования больше всего подходит ActiveX-компонент Microsoft Internet Explorer.

В системе "1С:Предприятие 7.7" использование данного компонента ActiveX, как и любого другого, является затруднительным, поэтому все, о чем здесь пойдет речь, будет касаться версии 8.0.

На рис. 1.11 изображена форма HTML-редактора, реализованная средствами встроенного языка системы "1С:Предприятие 8.0".

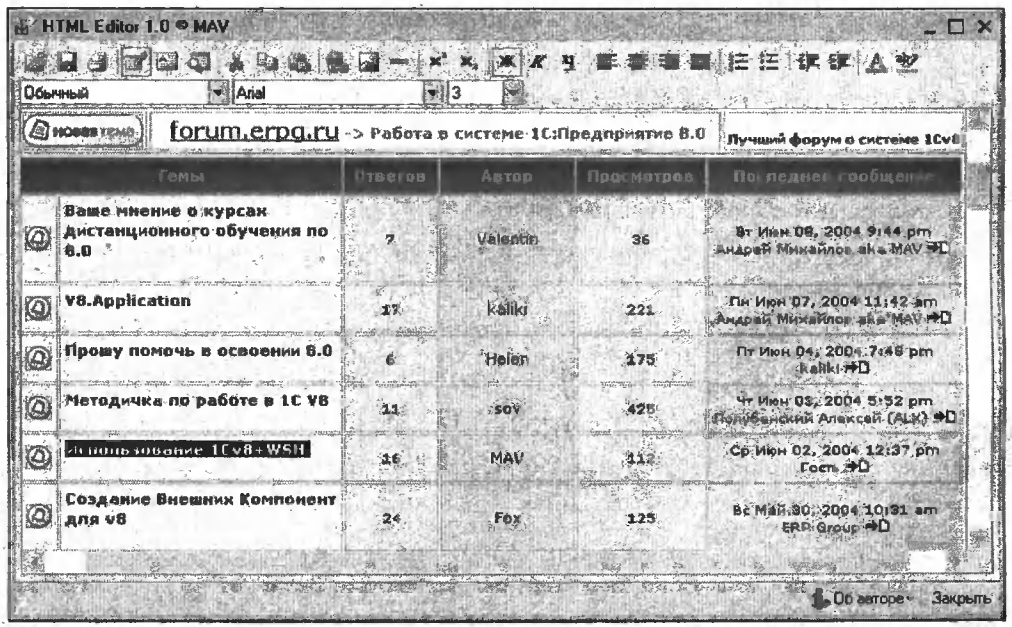


Рис. 1.11. Визуальный HTML-редактор

Основным элементом данной формы является ПолеHTMLДокумента, который имеет один очень важный метод — Перейти, и одно не менее важное свойство — Документ. Метод Перейти переводит элемент управления ПолеHTMLДокумента на страницу с заданным адресом, который определяется в единственном параметре метода. Свойство Документ возвращает значение типа COMОбъект, которое предоставляет доступ к HTML-документу.

Рассмотрим основные методы объекта Документ.

- Open() — открывает новый поток для методов Write и WriteLn;
- Close() — закрывает текущий документ;
- Write(Text) — записывает HTML-текст в текущий открытый поток;
- WriteLn(Text) — аналогичен методу Write, но добавляет в конец возврат каретки;
- ExecCommand(Command, UserInterface, Value) — выполняет команду над всем выбранным фрагментом, при этом переменные метода определяются как:
  - Command — идентификатор команды, подлежащей выполнению;
  - UserInterface — признак того, нужно ли при необходимости предоставлять графический интерфейс (необязательный параметр);
  - Value — значение команды (необязательный параметр);
- QueryCommandEnabled(Command) — возвращает двоичное значение, показывающее, доступна ли заданная команда;
- QueryCommandValue(Command) — возвращает текущее значение заданной команды;
- ElementFromPoint(X, Y) — возвращает элемент, которому принадлежит точка, заданная координатами X и Y;
- QueryCommandSupported(Command) — возвращает признак, доступна ли заданная команда;
- QueryCommandState(Command) — возвращает текущее состояние команды;
- QueryCommandIndeterm(Command) — возвращает признак, находится ли команда в неопределенном состоянии или нет;
- CreateElement(Tag) — создает объект элемента заданного тега;

Рассмотрим пример формирования простейшей HTML-страницы (листинг 1.26).

Листинг 1.26. Пример формирования HTML-страницы

```
Документ = ЭлементыФормы.ПолеHTMLДокумента.Документ;
Документ.Open();
Документ.WriteLn("<H1>Пример формирования HTML страницы</H1>");
Документ.Write("<P>Пишите <A href=""mailto:mav@erpg.ru"">письма</A>");
Документ.Close();
```

Данный пример демонстрирует динамическое формирование HTML-документа, но не редактирование его. Для реализации возможности полноценного редактирования документа предназначен метод `ExecCommand(Command, UserInterface, Value)`. Здесь, как уже было сказано ранее, `Command` — это имя выполняемой команды, `UserInterface` — необязательный логический параметр (принимает значения `Истина` или `Ложь`), который отвечает за вывод пользовательского интерфейса команды (если таковой имеется). Если данный параметр опущен, то его значение будет — `Ложь`. `Value` — необязательный параметр, указывающий дополнительные значения для команды.

Важно отметить, что метод `ExecCommand` может работать с выделенным текстом.

С помощью данного метода можно: преобразовывать обычный текст в полужирный (выделенный текст будет помещен в теги `<STRONG></STRONG>`); наклонный; задавать цвет фона; цвет текста; вставлять гиперссылки, рисунки и многое другое (листинг 1.27).

Листинг 1.27. Пример форматирования выделенного фрагмента текста

```
// Раскрасить выделенный фрагмент в красный цвет
Документ.execCommand("ForeColor", Ложь, "FF0000");
// Выделить текст полужирным шрифтом
Документ.execCommand("Bold", Ложь);
// Вставить рисунок
Документ.execCommand("InsertImage", Ложь, "C:\Img\mav.jpg");
// Отформатировать текст по правому краю
Документ.execCommand("JustifyRight", Ложь);
// Установить для выделенного фрагмента текста шрифт Arial
Документ.execCommand("FontName", Ложь, "Arial");
// Отформатировать текст как заголовок второго уровня
Документ.execCommand("FormatBlock", Ложь, "<H2>");
```

Приведем полный список доступных команд.

- `BackColor` — устанавливает или получает цвет фона текущего выделения (`Value` должно содержать имя цвета или его шестнадцатеричный RGB эквивалент, например, `FFCC00`);
- `Bold` — переключает начертание текста текущего выделения между полужирным и нормальным;
- `Copy` — копирует выделение в буфер обмена;
- `CreateBookmark` — получает имя якоря или создает его для текущего выделения (`Value` — строка, содержащая имя якоря);
- `CreateLink` — получает URL-ссылки или создает новую ссылку (параметр `Value` должен содержать URL);
- `Cut` — вырезает текущее выделение в буфер обмена;
- `Delete` — очищает текущее выделение (удаляет все его содержимое);
- `Find` — находит текст, заданный в параметре `Value` в текущем выделении;
- `FontName` — устанавливает шрифт для текущего выделения, причем `Value` содержит описание этого шрифта (как в теге `<FONT>`);
- `FontSize` — устанавливает размер шрифта (`Value` — число от 1 до 7 включительно);
- `ForeColor` — устанавливает цвет текста, при этом `Value` должно содержать имя цвета или его шестнадцатеричный RGB-эквивалент (например, `FFCC00`);
- `FormatBlock` — устанавливает или получает форматирование текущего блока (`Value` может содержать теги-описатели);
- `Indent` — увеличивает отступ выделенного текста на одну единицу приращения;
- `InsertButton` — перезаписывает идентификатор кнопки вместо текущего выделения (`Value` — строка, содержащая идентификатор кнопки);
- `InsertFieldset` — вставляет поля ввода;
- `InsertHorizontalRule` — вставляет горизонтальную полосу;
- `InsertIFrame` — вставляет встроенный фрейм (`IFRAME`);
- `InsertImage` — вставляет изображение;
- `InsertInputButton` — вставляет кнопку;
- `InsertInputCheckbox` — вставляет флажок (`CheckBox`);
- `InsertInputFileUpload` — вставляет элемент выбора файла;
- `InsertInputHidden` — вставляет скрытое поле (`hidden`);
- `InsertInputImage` — вставляет изображение;

- `InsertInputPassword` — вставляет поля ввода пароля;
- `InsertInputRadio` — вставляет радиокнопку (`Radio`);
- `InsertInputReset` — вставляет кнопку `Reset`;
- `InsertInputSubmit` — вставляет кнопку `Submit`;
- `InsertInputText` — вставляет поля ввода текста;
- `InsertParagraph` — вставляет новый раздел (абзац);
- `InsertOrderedList` — переключает стиль текущего выделения между нумерованным списком и простым текстом;
- `InsertUnorderedList` — переключает стиль текущего выделения между маркированным списком и простым текстом;
- `InsertSelectDropdown` — записывает элемент Drop-down вместо текущего выделения (`Value` должно содержать идентификатор элемента);
- `InsertTextArea` — вставляет элемент `TextArea`;
- `Italic` — переключает начертание текста текущего выделения между наклонным и обычным;
- `JustifyCenter` — устанавливает выравнивание по центру для всего блока, в котором расположено текущее выделение;
- `JustifyLeft` — устанавливает выравнивание по левому краю для всего блока, в котором расположено текущее выделение;
- `JustifyRight` — устанавливает выравнивание по правому краю для всего блока, в котором расположено текущее выделение;
- `Outdent` — уменьшает отступ для всего блока, в котором расположено выделение, на одну единицу;
- `OverWrite` — переключается между режимами вставки текста и замены текста при вводе (значением `Value` может быть `Истина` — замена, `Ложь` — вставка);
- `Paste` — вставляет текст из буфера обмена вместо текущего выделения;
- `Refresh` — обновляет текущий документ;
- `RemoveFormat` — удаляет из текущего фрагмента все теги форматирования;
- `SelectAll` — выделяет все содержимое документа;
- `UnBookmark` — удаляет все закладки из текущего выделения;
- `Underline` — переключает начертание текста текущего выделения между подчеркнутым и обычным;
- `Unlink` — удаляет все гиперссылки из текущего выделенного фрагмента;
- `Unselect` — снимает выделение.

С помощью объекта Документ можно переключать представления редактируемого HTML-документа между режимами визуального редактирования, текстового и режимом просмотра.

Для переключения между режимами необходимо знать не только в какой режим нужно перевести HTML-документ, но и из какого. В листинге 1.28 приведен пример процедуры, которая производит данное переключение.

Листинг 1.28. Процедура переключения между режимами

```
Процедура ПереключитьРежим(Документ, ПредыдущийРежим, НовыйРежим)
  Если НовыйРежим = "Текст" Тогда
    Документ.Body.InnerText = Документ.Body.InnerHTML;
  ИначеЕсли ПредыдущийРежим = "Текст" Тогда
    Документ.Body.InnerHTML = Документ.Body.InnerText;
  КонецЕсли;

  Если НовыйРежим = "Просмотр" Тогда
    Документ.Body.ContentEditable = "false";
  Иначе
    Документ.Body.ContentEditable = "true";
  КонецЕсли;
КонецПроцедуры
```

Данная процедура имеет три параметра:

- Документ — ссылка на объект Документ;
- ПредыдущийРежим — строка, описывающая ранее использованный режим документа, причем этот параметр может принимать значения Текст и Просмотр;
- НовыйРежим — строка, описывающая новый режим документа.

При использовании процедуры, описанной в листинге 1.29, для переключения в режим редактирования HTML-кода существует довольно большой недостаток — весь код отображается одним цветом, что очень затрудняет его чтение и понимание, в то время как все современные редакторы выделяют разными цветами различные группы тегов. Чтобы раскрасить код в окне редактора, необходимо запустить специальный скрипт, который будет выполняться средствами компоненты Internet Explorer.

Приведем пример процедуры переключения между режимами с возможностью раскраски тегов (рис. 1.12) в режиме редактирования HTML кода (листинг 1.29).

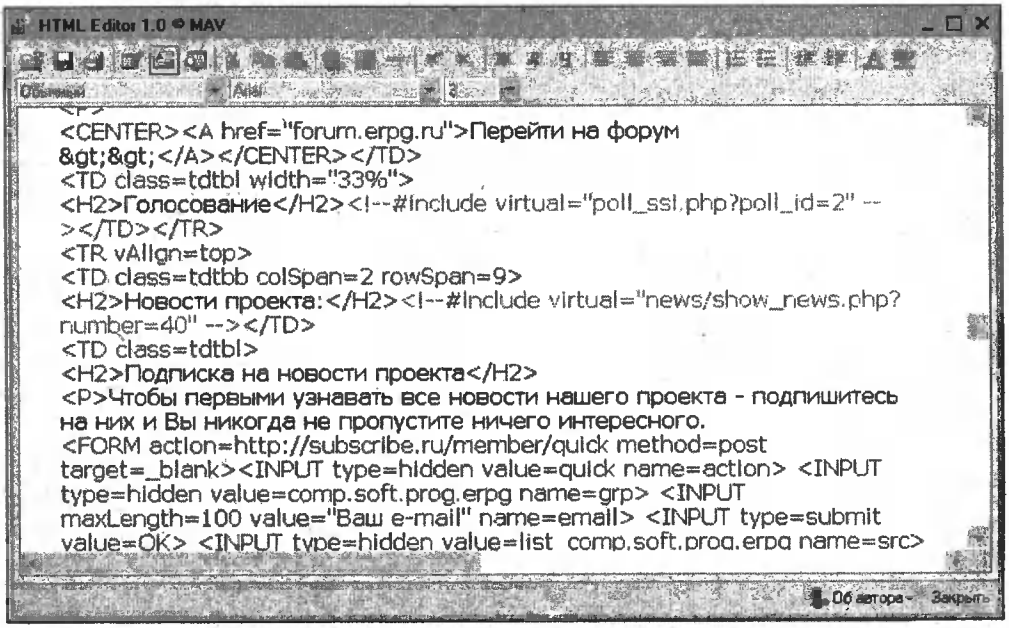


Рис. 1.12. Цветовая раскраска тегов в HTML-редакторе

Листинг 1.29. Процедура переключения между режимами с цветовой раскраской тегов

```
Процедура ПереключитьРежим(Документ, ПредыдущийРежим, НовыйРежим)
  Если НовыйРежим = "Текст" Тогда
    sExpression = "
    |document.body.innerText = document.body.innerHTML;
    |document.body.innerHTML = colourCode(document.body.innerHTML);
    |function colourCode(code)
    |{
    |    htmlTag = /(&lt;([\s\S]*?)&gt;)/gi
    |    tableTag = /(&lt;(table|tbody|th|tr|td|\/table|\/tbody|\/th|\
         /tr|\/td)([\s\S]*?)&gt;)/gi
    |    commentTag = /(&lt;!--([\s\S]*?)&gt;)/gi
    |    imageTag = /(&lt;img([\s\S]*?)&gt;)/gi
    |    linkTag = /(&lt;(a|\/a)([\s\S]*?)&gt;)/gi
    |    scriptTag = /(&lt;(script|\/script)([\s\S]*?)&gt;)/gi
    |    code = code.replace(htmlTag,""<font color=#FF0000>$1</font>"")
    |    code = code.replace(tableTag,""<font color=#008080>$1</font>"")
```

```
    |    code = code.replace(commentTag,
         ""<font color=#808080>$1</font>"")
    |    code = code.replace(imageTag,""<font color=#800080>$1</font>"")
    |    code = code.replace(linkTag,""<font color=#008000>$1</font>"")
    |    code = code.replace(scriptTag,""<font color=#800000>$1</font>"")
    |    return code;
    |}";
    Документ.parentWindow.execScript(sExpression);

  ИначеЕсли ПредыдущийРежим = "Текст" Тогда
    Документ.Body.InnerHTML = Документ.Body.InnerText;
  КонецЕсли;

  Если НовыйРежим = "Просмотр" Тогда
    Документ.Body.ContentEditable = "false";
  Иначе
    Документ.Body.ContentEditable = "true";
  КонецЕсли;
КонецПроцедуры
```

Регулярные выражения

Регулярные выражения — это очень мощный механизм для обработки строк. С его помощью можно легко найти нужные части текста, проверить, удовлетворяет ли строка определенной маске, и т. п. Такие выражения встроены во многие языки программирования, такие как Perl, JavaScript, Visual Basic. К сожалению, встроенный язык системы "1С:Предприятие" не позволяет напрямую работать с регулярными выражениями, поэтому для решения подобных задач используются COM-объекты.

Регулярные выражения являются частью технологии Microsoft Windows Script Technologies и входят в VBScript. Для использования движка регулярных выражений вам необходимо иметь библиотеку vbscript.dll. Данная библиотека включена в Internet Explorer 4 и выше.

С помощью регулярных выражений можно производить поиск, замену подстрок, используя шаблоны. Шаблоны состоят из обычных символов и так называемых метасимволов (metacharacters) — управляющих символов. Список метасимволов достаточно обширен. Наиболее часто используемые метасимволы приведены в табл. 1.1.

Таблица 1.1. Основные метасимволы регулярных выражений

Символ	Назначение
*	Соответствует выражению, находящемуся до знака "*", взятому ноль или большее количество раз. Например, шаблон "[0-9]*" определяет строку, содержащую ноль или большее количество цифр
\	Предназначен для определения символа, являющегося метасимволом. Например, шаблон "." соответствует любому символу, а шаблон "\." будет соответствовать точке
^	Определяет начало входной строки
$	Определяет конец входной строки
+	Соответствует выражению, находящемуся до знака "+", взятому один или более раз. Например, шаблон "[0-9]+" определяет строку, содержащую одну или более цифр
. (точка)	Определяет любой символ, кроме символа перевода строки
|	Разделяет два выражения. Например, шаблону "a|b" будут соответствовать строки "a" и "b"
[a-z]	Определяет диапазон символов. Например, шаблон "[0-9]" определяет цифру
[^...]	Определяет любой символ, не соответствующий заданному набору. Например, шаблон "[^0-9]" определяет любой символ, кроме цифры
\w	Слово. То же, что и [a-zA-Z_0-9]
\W	Все, кроме слов. То же, что и [^a-zA-Z_0-9]
\s	Любое пустое место
\S	Любое непустое место
\d	Десятичная цифра. То же, что и [0-9]
\D	Не цифра. То же, что и [^0-9]
{}	В фигурных скобках указывается количество символов, подходящих по описанное ранее правило
()	С помощью круглых скобок можно объединять метасимволы в группы

Для лучшего понимания синтаксиса регулярных выражений рассмотрим несколько примеров.

- "(два|три) богатыря" — соответствует как строке, которая может принимать значение как "два богатыря", так и "три богатыря".
- ".+@.+\..+" — соответствует e-mail-адресу.

- "a(.)\1c" — соответствует одному символу "a", затем любым двум одинаковым символам, потом символу "c". Например, шаблон совпадет с "axxc", но не с "axyc".
- "(\d\d)\.\1\.\d\d\d\d" — соответствует дате, у которой число и месяц одинаковы, то есть дата "09.09.2001" — подойдет, а дата "09.12.2001" — нет.
- "\d{2}-\d{5}" — данный шаблон проверяет правильность номера идентификатора, состоящего из двух цифр, дефиса, и еще пяти цифр.
- "<(.*)>.*<\/\1>"— шаблон соответствует тегу HTML.

Для использования в системе механизма регулярных выражений необходимо создать COM-объект с идентификатором VBScript.RegExp.

Объект RegExp имеет следующие свойства:

- Global — признак поиска соответствия во всей строке, причем если свойство принимает значение Истина, то поиск ведется во всей строке, если Ложь — то только до первого совпадения;
- IgnoreCase — признак игнорирования регистра символов при поиске (свойство может принимать значение Истина и Ложь);
- Pattern — строковый шаблон для поиска;
- MultiLine — признак, является ли текст многострочным или нет (свойство может принимать значение Истина и Ложь).

Рассмотрим три метода объекта RegExp.

- Test(ИсходнаяСтрока) — метод Test позволяет протестировать строку ИсходнаяСтрока на соответствие ее шаблону (Pattern). Возвращает значение Истина, если строка, переданная в качестве параметра, соответствует определенному шаблону, Ложь — иначе.
- Execute(ИсходнаяСтрока) — метод выполняет поиск всех совпадений регулярного выражения в строке ИсходнаяСтрока и возвращает коллекцию MatchCollection, которая содержит в себе всю информацию о совпадениях. Метод Execute может быть использован, например, для извлечения всех e-mail-адресов из строки ИсходнаяСтрока.
- Replace(ИсходнаяСтрока, НоваяСтрока) — метод Replace дает возможность заменить все совпадения регулярного выражения строкой НоваяСтрока.

В листинге 1.30 приведен пример использования метода Execute и разбора коллекции MatchCollection.

Листинг 1.30. Поиск совпадений регулярного выражения в строке

```
// Описание регулярного выражения (шаблона)
РегулярноеВыражение = "<BODY (.*)</BODY>";
// Определение строки
ТестоваяСтрока = "<HTML><BODY bgcolor=#FFAABB><P>Это пример</P></BODY>
</HTML>";

// Создание объекта работы с регулярными выражениями
RegExp = Новый COMОбъект("VBScript.RegExp");
RegExp.MultiLine = Ложь;
RegExp.Global = Истина;
RegExp.IgnoreCase = Истина;
RegExp.Pattern = РегулярноеВыражение;

// Поиск совпадений регулярного выражения в строке
Matches = RegExp.Execute(ТестоваяСтрока);

// Обход элементов коллекции MatchCollection
Для каждого Match Из Matches Цикл
  Сообщить("Номер первого совпавшего символа: " + Match.FirstIndex);
  Сообщить("Длина совпавшего образца: " + Match.Length);
  Сообщить("Первый совпавший образец: " + Match.SubMatches(0));
  Сообщить("Полный образец: " + Match.Value);
КонецЦикла;
```

В результате выполнения такого кода в окне сообщений появятся следующие строки:

- первая — число `6` (свойство `FirstIndex`) определяет номер первого совпавшего символа (т. е. символа "b");
- вторая строка — `46` (свойство `Length`) определяет длину совпавшего образца;
- третья строка — `"bgcolor=#FFAABB><P>Это пример</P>"` является первым совпавшим образцом, т. е. текст, попавший в скобки `"(.*)"` в шаблоне (для второго совпадения нужно выполнить метод `SubMatches(1)` и т. д.);
- последняя строка — `"<BODY bgcolor=#FFAABB><P>Это пример</P></BODY>"` (свойство `Value`) представляет полный образец.

Для того чтобы получить цвет фона, определенного в теге `<BODY>` в переменной `ТестоваяСтрока` достаточно изменить регулярное выражение на

строку "(#[0-9A-F]{6})". При этом свойство SubMatches(0) вернет как раз необходимую строку "#FFAABB", содержащую цвет в шестнадцатеричной системе счисления.

Для проверки совпадения строки с регулярным выражением используется метод Test. В листинге 1.31 приведен пример такой проверки.

Листинг 1.31. Проверка совпадений регулярного выражения в строке

```
// Описание регулярного выражения (шаблона)
РегулярноеВыражение = "(#[0-9A-F]{6})";
// Определение строки
ТестоваяСтрока = "<HTML><BODY bgcolor=#FFAABB><P>Это пример</P></BODY> </HTML>";

// Создание объекта работы с регулярными выражениями
RegExp = Новый COMОбъект("VBScript.RegExp");
RegExp.MultiLine = Ложь;
RegExp.Global = Истина;
RegExp.IgnoreCase = Истина;
RegExp.Pattern = РегулярноеВыражение;

// Проверка совпадений регулярного выражения в строке
Если RegExp.Test(ТестоваяСтрока) Тогда
  Сообщить("Найдены совпадения!");
Иначе
  Сообщить("Совпадений нет!");
КонецЕсли;
```

Данный пример приведен для системы "1С:Предприятие 8.0". В версии 7.7 метод Test будет возвращать -1 при значении Истина и 0 при значении Ложь.

Для демонстрации работы метода Replace рассмотрим пример, приведенный в листинге 1.32, который заменит текст, находящийся в тегах <P>, на другой.

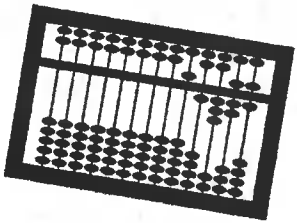
Листинг 1.32. Замена совпадения регулярного выражения в строке

```
// Определение строки
ТестоваяСтрока = "<HTML><BODY bgcolor=#FFAABB><P>Это пример</P></BODY> </HTML>";
// Определение регулярного выражения
```

```
RegExp.Pattern = "<P>(.*?)</P>";
// Замена текста в переменной ТестоваяСтрока на другую
НовыйТекст = RegExp.Replace(ТестоваяСтрока, "<P>Новый текст в теге
P!</P>")
// Вывод нового текста
Сообщить("Текст после замены:" + Символы.ПС + НовыйТекст);
```

В результате работы метода `Replace` переменная `НовыйТекст` будет содержать строку `"<HTML><BODY bgcolor=#FFAABB><P>Новый текст в теге P!</P></BODY></HTML> "`.

В данном случае в тестируемой строке будет только одна пара тегов `<P></P>`. Но вы можете сами добавить еще одну пару и проверить работоспособность кода.



Глава 2

Администрирование Windows средствами WMI

Понятие WMI

Одной из базовых технологий компании Microsoft, предназначенной для централизованного управления и слежения за работой различных частей компьютерной сети, под управлением Windows, является *Windows Management Instrumentation* (WMI).

Объектную модель WMI можно использовать практически в любых языках и средах программирования, в том числе и в системе "1С:Предприятие". Кроме этого, в операционных системах Windows Xp или Server 2003 реализован механизм доступа к WMI из командной строки и обычных пакетных файлов (технология WMI Command line, WMIC). Применение WMI позволяет автоматически выполнять административные задачи практически любой степени сложности, включая отслеживание и обработку событий, связанных с теми или иными изменениями в информационной системе (например, появление определенной записи в журнале событий на локальном или удаленном компьютере, заполнение жесткого диска сервера до определенного предела и т. п.).

Напомним, что в Windows NT для администрирования операционной системы приходилось пользоваться несколькими утилитами и инструментами. Это обусловлено тем, что данные о компонентах системы хранятся в различных источниках (база пользователей SAM, журнал событий Event Log, системный реестр и т. д.), доступ к которым осуществлялся с помощью разных утилит (диспетчер пользователей — User Manager, просмотрщик журнала событий — Event Log Viewer, редактор реестра — Regedit) и программных интерфейсов (Network API — для работы с данными о пользователях, Event Log API — для просмотра сведений о произошедших событиях, Registry API — для чтения или редактирования системного реестра). Понятно, что это неудобно, т. к., во-первых, приходилось осваивать множество не связанных друг с другом технологий и инструментов, а во-вторых, усложнялось

взаимодействие разных компонентов компьютерной системы друг с другом (например, настройки в локальном реестре рабочей станции могли конфликтовать с политиками безопасности, которые заданы на сервере).

С появлением Windows 2000 эта ситуация была в основном исправлена. Появилась встроенная в операционную систему консоль управления MMC (Microsoft Management Console), с помощью которой можно из одной точки управлять большинством логических и физических компонентов компьютерной сети, построенной на основе Windows. В свою очередь, это стало возможным именно благодаря применению технологии WMI. Данная технология, во-первых, делает доступ к информации о самых различных компонентах информационной системы (журнал событий, системный реестр, подсистема производительности, драйверы устройств и т. д.) независимым от типа этих компонентов, а во-вторых, осуществляет управление любой подсистемой стандартным, не зависящим от реализации этой подсистемы, методом.

Итак, технология WMI — это глобальная концепция настройки, управления и слежения за работой различных частей корпоративной компьютерной сети. В частности, используя WMI, можно с помощью специальных утилит или сценариев Windows Script Host (WSH) решать ряд важных задач.

- *Управление различными версиями операционной системы Windows.* С помощью сценариев WMI можно обращаться к системным счетчикам производительности, анализировать журналы событий (Event Logs), работать с файловой системой, установленными принтерами, управлять запущенными процессами и сервисами, просматривать и изменять настройки реестра, создавать и удалять совместно используемые ресурсы и т. д. При этом все операции можно выполнять одинаковым образом, как на локальной, так и на удаленной машине.
- *Управление ресурсами и службами сети.* Сценарии WMI позволяют настраивать сетевые службы (DNS, DHCP и т. п.) и управлять сетевыми устройствами, поддерживающими технологию SNMP (Simple Network Management Protocol).
- *Мониторинг состояния системы в реальном времени.* Можно создавать сценарии — обработчики событий WMI, которые позволяют отслеживать и нужным образом обрабатывать события, связанные с теми или иными изменениями в информационной системе. Например, появление определенной записи в журнале событий на локальном или удаленном компьютере, заполнение жесткого диска сервера до определенного предела, изменение определенного ключа в системном реестре и т. п.
- *Управление серверными приложениями Windows.* С помощью WMI можно управлять различными приложениями Microsoft: Application Center, Operations Manager, Systems Management Server, Internet Information Server, Exchange Server, SQL Server.

Доступ к объектам WMI

Для программирования WMI разработан набор классов, которые осуществляют управление объектами WMI. Ввиду большого их количества, ограничимся рассмотрением только основных, с которыми обязательно придется работать.

- `SWbemLocator` — обеспечивает доступ к объекту `SWbemServices`, который осуществляет подключение к WMI. Предоставляет дополнительные возможности проверки подлинности при установлении соединения.
- `SWbemServices` — обеспечивает доступ к экземплярам управляемых объектов, выполняет запросы к классам WMI, выполняет другие связанные с WMI операции.
- `SWbemObjectSet` — является коллекцией объектов `SWbemObject`. Создать объект `SWbemObjectSet` и получить к нему доступ можно при помощи нескольких методов, принадлежащих объектам `SWbemServices` и `SWbemObject`.
- `SWbemObject` — представляет собой определение одного класса WMI в форме экземпляра объекта.

Подробное описание всех объектов, их свойств и методов можно найти на сайте Microsoft Developer Network Library (MSDN) — **http://msdn.microsoft.com/library/**.

Для доступа к WMI через библиотеку сценариев WMI из системы "1С:Предприятие" нужно сделать три шага, которые являются общими для большинства сценариев WMI. Первое — необходимо подключиться к службе Windows Management Service, второе — получить экземпляры управляемых объектов WMI, третье — вызвать метод или получить доступ к свойствам управляемого объекта. После знакомства с интерфейсами, которые используются для выполнения этих трех шагов, гораздо увереннее чувствуешь себя на пути к вершине искусства написания сценариев.

Самое неприятное в этих основных трех шагах то, что код их написания довольно коренным образом отличается в разных версиях "1С:Предприятия" (7.7 и 8.0). Причем многие конструкции в версии 7.7 штатными средствами просто не выполнить, поэтому приходится прибегать к внешним вспомогательным инструментам.

Для доступа к объектам WMI из "1С:Предприятия" версии 8.0 вполне хватает стандартных языковых конструкций внутреннего языка, поэтому сначала будут описаны алгоритмы подключения именно для этой версии.

Рассмотрим пример подключения к классу `Win32_LogicalDisk` (листинг 2.1).

Листинг 2.1. Создание доступа к объектам WMI для версии 8.0

```
// Создать указатель на объект SWbemLocator
Попытка
  Locator = Новый COMОбъект("WbemScripting.SWbemLocator");
Исключение
  Сообщить(ОписаниеОшибки());
  Возврат;
КонецПопытки;

// Шаг 1. Подключение к WMI на локальном компьютере
ServicesSet = Locator.ConnectServer(".");

// Шаг 2. Извлечение экземпляров класса Win32_LogicalDisk
ObjectSet = ServicesSet.InstancesOf("Win32_LogicalDisk");

// Шаг 3. Просмотр SWbemObjectSet, содержащего экземпляры
Для каждого Item Из ObjectSet Цикл
  Сообщить("Имя: " + Item.Caption);
КонецЦикла;
```

Данный код покажет название всех имеющихся логических дисков на локальном компьютере, но это пока нам не важно. Можно также просмотреть все процессоры не только на локальном компьютере, но и на удаленном. Для этого необходимо в предыдущем коде только изменить первый шаг (листинг 2.2).

Листинг 2.2. Подключение к удаленному серверу для версии 8.0

```
// Шаг 1. Подключение к WMI на удаленном компьютере
ServicesSet = Locator.ConnectServer("MainServer", "root\cimv2", "Admin",
"password");
```

Рассмотрим каждый из параметров метода `ConnectServer` в отдельности.

- Первый — имя удаленного компьютера (для локального указывается точка ".").
- Второй — имя пути для WMI. По умолчанию `root\cimv2` можно не указывать;
- Третий — имя пользователя (для авторизации).
- Четвертый — пароль пользователя (для авторизации).

Иногда возникает необходимость подключиться к компьютеру создав COM-объект с помощью расширенной функции ПолучитьCOMОбъект (аналог GetObject в Visual Basic и Visual Basic Script). Пример такого подключения показан в листинге 2.3.

Листинг 2.3. Подключение к серверу с помощью моникера для версии 8.0

```
ИмяКомпьютера = ".";

// Шаг 1. Подключение к WMI на локальном компьютере
ServicesSet =
ПолучитьCOMОбъект("winmgmts:{impersonationLevel=impersonate}!\\
" + ИмяКомпьютера + "\root\cimv2");

// Шаг 2. Извлечение экземпляров класса Win32_LogicalDisk
ObjectSet = ServicesSet.ExecQuery("Select * from Win32_LogicalDisk");

// Шаг 3. Просмотр SWbemObjectSet, содержащего экземпляры
Для каждого Item Из ObjectSet Цикл
  Сообщить("Имя: " + Item.Caption);
КонецЦикла;
```

В данном примере для подключения к WMI на целевом компьютере и доступа ко всем экземплярам класса Win32_LogicalDisk используется WMI-моникер (moniker) winmgmts. *Моникер* — это специальный COM-объект, который отыскивает в сети или создает и инициализирует экземпляр объекта и возвращает клиенту указатель на него. Для повышения безопасности WMI-моникер использует встроенные средства аутентификации, а также позволяет устанавливать дополнительные маршруты к объектам.

Сценарий возвращает каждый экземпляр Win32_LogicalDisk как SWbemObject из коллекции SWbemObjectSet. SWbemObjectSet и SWbemObject — это лишь два из нескольких интерфейсов, которые предоставляет библиотека сценариев WMI. Так как SWbemObjectSet является коллекцией, то доступ к ее элементам можно организовать при помощи конструкции Для каждого..Из..Цикл..КонецЦикла. Внутри цикла организован доступ к свойству Caption, которое определено в классе Win32_LogicalDisk.

Самое интересное в этом примере то, что на втором шаге пишется текст запроса, который очень похож на конструкции структуризированного языка запросов (SQL). Благодаря этому мы можем выбрать не все экземпляры выборки, а только необходимые, т. е. соответствующие некоторым условиям отбора.

Модифицируем последний пример так, чтобы он выводил информацию только о жестких дисках (HDD) и не показывал CD-ROM, дисководы FDD и т. п. (листинг 2.4).

Листинг 2.4. Пример извлечения экземпляров класса с условием для версии 8.0

```
// Шаг 2. Извлечение экземпляров класса Win32_LogicalDisk
ObjectSet = ServicesSet.ExecQuery("Select * from Win32_LogicalDisk Where
DriveType=3");

// Шаг 3. Просмотр SWbemObjectSet, содержащего экземпляры
Для каждого Item Из ObjectSet Цикл
  Сообщить("Имя: " + Item.Caption);
КонецЦикла;
```

В данном примере тип диска (`DriveType`), равный трем, соответствует жестким дискам. Это бывает довольно удобно для ограничения выборки элементов и повышения производительности системы.

Хотя варианты подключения с помощью моникера и объекта `SWbemLocator` функционально одинаковы, они демонстрируют действие различных механизмов, заложенных в библиотеке WMI. Новичкам может показаться, что синтаксис моникера излишне запутан, поэтому вначале они могут поработать с `SWbemLocator`. С точки зрения функциональности метод `ConnectServer`, который предоставляет `SWbemLocator`, позволяет распределять полномочия, необходимые для проведения аутентификации WMI-соединения. Синтаксис моникера предоставляет возможность выполнения нескольких действий в одной строке кода.

Теперь рассмотрим, как все то же самое выполняется в системе "1С:Предприятие" версии 7.7.

Сразу необходимо отметить, что в версии 7.7 невозможно штатными средствами:

- перебрать элементы коллекции, так как в 7.7 нет цикла, аналогичному конструкции `Для каждого...Из...Цикл...КонецЦикла`;
- получить доступ к экземплярам классов, если используется WMI-моникер, так как в 7.7 нет аналога функции `ПолучитьCOMОбъект`.

Это довольно серьезные ограничения, связанные с работой COM-объектов, но вполне решаемые с помощью вспомогательных внешних компонент или дополнительных COM-объектов, которые можно написать на любом другом языке программирования. Так как эта глава книги (как, впрочем, и многие другие) посвящена COM-технологиям, то и для обхода ограничений версии 7.7 будет использоваться COM-объект.

Для обхода данных ограничений понадобятся всего три функции (листинг 2.5):

- функция получения очередного элемента коллекции;
- функция получения количества элементов в коллекции;
- функция, заменяющая функцию ПолучитьCOMОбъект в версии 8.0.

Листинг 2.5. Пример кода COM-объекта на языке Visual Basic

```
'Функция возвращает элемент коллекции ObjSet по номеру Number
Public Function EnumerateCollection(ObjSet, Number)
    Dim Current As Integer
    Current = 0
    For Each Obj In ObjSet
        If Current = Number Then
            Set EnumerateCollection = Obj
            Exit Function
        End If
        Current = Current + 1
    Next
End Function

'Функция возвращает количество элементов коллекции ObjSet
Public Function GetCountCollection(ObjSet)
    Dim Current As Integer
    Current = 0
    For Each Obj In ObjSet
        Current = Current + 1
    Next
    GetCountCollection = Current
End Function

'Функция возвращает COM-объект по моникеру или имени файла
Public Function GetCOMObject(Str)
    Set GetCOMObject = GetObject(Str)
End Function
```

Данный пример является частью кода COM-объекта, созданного в среде Visual Basic. После компиляции мы получим библиотеку COMServices.dll, которая представляет COM-объект — COMServices.COM.

Местоположение файла COMServices.dll не важно, главное зарегистрировать его в системе командой regsvr32 COMServices.dll.

Теперь рассмотрим, как описанные ранее три шага подключения к WMI будут выглядеть в версии 7.7 с использованием вспомогательного COM-объекта `COMServices` (листинг 2.6).

Листинг 2.6. Создание доступа к объектам WMI для версии 8.0

```
// Создать указатель на объект COMServices
Попытка
  COMServices = СоздатьОбъект("COMServices.COM");
Исключение
  Предупреждение("Ошибка загрузки COM-объекта COMServices.");
  Возврат;
КонецПопытки;

// Создать указатель на объект SWbemLocator
Попытка
  Locator = СоздатьОбъект("WbemScripting.SWbemLocator");
Исключение
  Сообщить(ОписаниеОшибки());
  Возврат;
КонецПопытки;

// Шаг 1. Подключение к WMI на локальном компьютере
ServicesSet = Locator.ConnectServer(".");

// Шаг 2. Извлечение экземпляров класса Win32_LogicalDisk
ObjectSet = ServicesSet.InstancesOf("Win32_LogicalDisk");

// Шаг 3. Просмотр SWbemObjectSet, содержащего экземпляры
Для Ном = 0 По ObjectSet.Count-1 Цикл
  Item = COMServices.EnumerateCollection(ObjectSet, Ном);
  Сообщить("Имя: " + Item.Caption);
КонецЦикла;
```

В данном примере используется конструкция цикла `Для...По...Цикл...КонецЦикла` от нуля до количества элементов (свойство `Count`) коллекции минус один. Внутри цикла вызывается метод `EnumerateCollection` объекта `COMServices`,

в который передается ссылка на коллекцию ObjectSet и номер извлекаемого элемента коллекции. Во всем остальном код ничем не отличается от аналогичного кода в версии 8.0. Важно отметить, что не каждый класс имеет свойство Count. Если оно отсутствует, то для определения количества элементов в коллекции применяется метод GetCountCollection объекта COMServices.

Рассмотрим второй вариант подключения к объектам WMI с помощью мониkеров в версии 7.7 (листинг 2.7).

Листинг 2.7. Подключение к серверу с помощью мониkера для версии 7.7

```
// Сетевое имя компьютера, к которому производится подключение
ИмяСервера = ".";

// Шаг 1. Подключение к WMI на локальном компьютере
ServicesSet = COMServices.GetCOMObject
("winmgmts:{impersonationLevel=impersonate}!\\" + ИмяСервера +
"\root\cimv2");

// Шаг 2. Извлечение экземпляров класса Win32_LogicalDisk
ObjectSet = ServicesSet.ExecQuery("Select * from Win32_LogicalDisk");

// Шаг 3. Просмотр SWbemObjectSet, содержащего экземпляры
Для Ном = 0 По ObjectSet.Count-1 Цикл
  Item = COMServices.EnumerateCollection(ObjectSet, Ном);
  Сообщить("Имя: " + Item.Caption);
КонецЦикла;
```

Данный пример модифицировался лишь в том, что на первом шаге добавился вызов метода GetCOMObject для получения объекта SWbemServices.

В следующих частях этой главы будут показаны примеры практического применения технологии WMI. Большинство примеров будет написано только для версии 8.0 системы "1С:Предприятие". При необходимости перевести их в формат версии 7.7 не составит труда.

Сбор данных об аппаратном составе

В листинге 2.1 приведен пример вывода названий всех имеющихся логических дисков локального компьютера. Помимо названий класс Win32_LogicalDisk имеет еще очень большое количество свойств, таких как описание, файло-

вая система, метка диска и т. п. В листинге 2.8 приведен пример вывода основных свойств каждого элемента класса Win32_LogicalDisk.

Листинг 2.8. Вывод основных свойств класса Win32_LogicalDisk

```
// Создать указатель на объект SWbemLocator
Попытка
  Locator = Новый COMОбъект("WbemScripting.SWbemLocator");
Исключение
  Сообщить(ОписаниеОшибки());
  Возврат;
КонецПопытки;

// Шаг 1. Подключение к WMI на локальном компьютере
ServicesSet = Locator.ConnectServer(".");

// Шаг 2. Извлечение экземпляров класса Win32_LogicalDisk
ObjectSet = ServicesSet.InstancesOf("Win32_LogicalDisk");

// Шаг 3. Просмотр SWbemObjectSet, содержащего экземпляры
Для каждого Item Из ObjectSet Цикл
  Сообщить("Имя: " + Item.Caption);
  Сообщить("Описание: " + Item.Description);
  Сообщить("Файловая система: " + Item.FileSystem);
  Сообщить("Свободное место: " + Item.FreeSpace);
  Сообщить("Метка диска: " + Item.VolumeName);
  Сообщить("==========================================");
КонецЦикла;
```

Для вывода всех имеющихся свойств класса Win32_LogicalDisk необходимо модифицировать только третий шаг (листинг 2.9).

Листинг 2.9. Вывод всех свойств класса Win32_LogicalDisk

```
// Шаг 3. Просмотр SWbemObjectSet, содержащего экземпляры
Для каждого Item Из ObjectSet Цикл
  Сообщить("Диск: " + Item.Caption);
  // Выборка по всем свойствам
  Для каждого Свойство Из Item.Properties_ Цикл
```

```
    Сообщить(Свойство.Name + " = " + Свойство.Value);
  КонецЦикла;
  Сообщить("========================");
КонецЦикла;
```

На практике чаще всего обращаются к конкретному свойству по его имени, однако нужно учитывать, что каждый класс имеет свой набор свойств.

Помимо класса `Win32_LogicalDisk` есть еще ряд классов, позволяющих получать информацию об аппаратном составе локального или удаленного компьютера. Наиболее интересные из них следующие:

- `Win32_Processor` — информация о процессорах;
- `Win32_BaseBoard` — информация о материнской плате;
- `Win32_DiskDrive` — информация о дисковых накопителях (HDD);
- `Win32_PhysicalMemory` — информация о физической памяти (RAM);
- `Win32_SoundDevice` — информация о звуковых картах;
- `Win32_NetworkAdapter` — информация о сетевых адаптерах;
- `Win32_CDROMDrive` — информация о приводах CD-ROM;
- `Win32_FloppyDrive` — информация о дисководах (Floppy);
- `Win32_DesktopMonitor` — информация о мониторах;
- `Win32_VideoController` — информация о видеосистеме;
- `Win32_Printer` — информация о принтерах;
- `Win32_Keyboard` — информация о клавиатуре;
- `Win32_PointingDevice` — информация о точечных манипуляторах (мыши, трекболы и т. п.);
- `Win32_USBHub` — информация о портах USB.

Работа с программным обеспечением

Для работы с программным обеспечением предназначен класс `Win32_Product`. Единственное ограничение данного класса состоит в том, что он работает только с тем программным обеспечением, которое было установлено с помощью Microsoft Windows Installer (MSI).

Основные свойства класса `Win32_Product`:

- `Name` — название продукта;
- `Description` — описание;
- `IdentifyingNumber` — идентификационный номер продукта (например серийный номер или номер аппаратного устройства);

- `InstallDate` — дата инсталляции;
- `InstallLocation` — путь к продукту;
- `Vendor` — производитель;
- `Version` — версия.

Основные методы `Win32_Product`:

- `Install(PackageLocation, Options, AllUsers)` — установка продукта, причем параметры метода определяются как:
  - `PackageLocation` — путь к файлу установки;
  - `Options` — параметры командной строки;
  - `AllUsers` — признак установки продукта для всех пользователей;
- `Reinstall(ReinstallMode)` — переустановить продукт с параметром:
  - `ReinstallMode` — режим переустановки продукта (например `1` — скопировать только недостающие файлы, `6` — обновить все файлы).
- `Uninstall()` — удалить целиком продукт.

Внимание

Все методы возвращает `0`, если они завершены успешно. Иначе возвращается число `2147549445`.

Получение списка установленного программного обеспечения

Пример получения всего списка установленного на данном компьютере программного обеспечения, с помощью класса `Win32_Product`, представлен в листинге 2.10.

Листинг 2.10. Получение списка установленного программного обеспечения

```
ServicesSet = Locator.ConnectServer(ИмяКомпьютера, "root\cimv2",
Пользователь, Пароль);
Items = ServicesSet.ExecQuery("Select * from Win32_Product");
Для каждого Item Из Items Цикл
  Сообщить("Имя: " + Item.Name + " (Вер." + Item.Version + ")");
КонецЦикла;
```

Если есть необходимость получить список программного обеспечения по какому-либо условию, можно в запросе использовать выражение `Where`. Приведем пример получения списка всех продуктов компании Microsoft, установленных в операционной системе (листинг 2.11).

Листинг 2.11. Получение списка продуктов Microsoft

```
ServicesSet = Locator.ConnectServer(ИмяКомпьютера, "root\cimv2",
Пользователь, Пароль);
Items = ServicesSet.ExecQuery("Select * from Win32_Product Where Vendor
LIKE '%Microsoft%'");
Для каждого Item Из Items Цикл
  Сообщить("Имя: " + Item.Name + " (Вер." + Item.Version + ")");
КонецЦикла;
```

В данном примере используется фильтр по маске — "%Microsoft%". Это означает, что в запрос попадут все объекты, у которых свойство `Vendor` содержит строку "Microsoft".

Удаление программного обеспечения

Для удаления какого-либо программного продукта используется метод `Uninstall` класса `Win32_Product`. Приведем пример удаления программы Adobe Acrobat 6.0 Professional (листинг 2.12).

Листинг 2.12. Удаление продукта Adobe Acrobat 6.0 Professional

```
ИмяПрограммы = "Adobe Acrobat 6.0 Professional";
Если Вопрос("Вы уверены, что хотите удалить " + ИмяПрограммы + "?",
РежимДиалогаВопрос.ДаНет) = КодВозвратаДиалога.Да Тогда
  ServicesSet = Locator.ConnectServer(ИмяКомпьютера, "root\cimv2",
  Пользователь, Пароль);
  Items = ServicesSet.ExecQuery("Select * from Win32_Product Where Name =
'" + ИмяПрограммы + "'");
  Для каждого Item Из Items Цикл
    Item.Uninstall();
  КонецЦикла;
КонецЕсли;
```

Работа со службами и процессами

Для работы со службами (сервисами) в WMI используется класс `Win32_Service`. Класс имеет довольно большое количество свойств. Рассмотрим только основные:

- `AcceptPause` — признак возможности приостановить службу;
- `AcceptStop` — признак возможности остановить службу;

- `Name` — название службы;
- `Description` — описание;
- `DisplayName` — имя службы, показываемое пользователю;
- `PathName` — путь к файлу;
- `ProcessId` — идентификатор процесса;
- `Started` — признак, показывающий, запущена служба или нет;
- `State` — строка, описывающая состояние службы (`Stopped`, `Start Pending`, `Stop Pending`, `Running`, `Continue Pending`, `Pause Pending`, `Paused`, `Unknown`).

Рассмотрим основные методы класса `Win32_Service`:

- `StartService()` — запускает службу;
- `StopService()` — останавливает службу;
- `PauseService()` — приостанавливает работу службы;
- `ResumeService()` — восстанавливает работу после приостановления;
- `Delete()` — удалить службу;
- `Create(...)` — создать службу (метод имеет множество параметров, которые не будут рассматриваться в рамках данной книги).

Для работы с процессами в WMI используется класс `Win32_Process`. Рассмотрим основные свойства класса:

- `Description` — описание процесса;
- `ProcessId` — идентификатор процесса;
- `Name` — имя процесса;
- `Priority` — приоритет (от 0 до 37);
- `VirtualSize` — объем памяти, необходимый процессу.

Основные методы класса `Win32_Process`:

- `SetPriority(Priority)` — установить приоритет процесса, при этом параметр `Priority` может принимать значения:
  - `64` — Idle (Ожидание, наиболее низкий приоритет);
  - `16384` — Below Normal (Ниже нормального);
  - `32` — Normal (Обычный);
  - `32768` — Above Normal (Выше нормального);
  - `128` — High Priority (Высокий приоритет);
  - `256` — Realtime (Приоритет реального времени, наивысший приоритет).
- `Terminate()` — завершить процесс и выгрузить его из памяти.

Получение списка сервисов

Для получения списка установленных сервисов используется класс `Win32_Service` (листинг 2.13).

Листинг 2.13. Получение списка сервисов

```
ServicesSet =
ПолучитьCOMОбъект("winmgmts:{impersonationLevel=impersonate}!\\
" + ИмяКомпьютера + "\root\cimv2");
Items = ServicesSet.ExecQuery("Select * from Win32_Service");
Для каждого Item Из Items Цикл
  Сообщить("Сервис: " + Item.DisplayName + " (" + Item.State + ")");
КонецЦикла;
```

Для определения списка сервисов, которые возможно остановить, используется также класс `Win32_Service`. Однако запрос ограничивается условием того, что свойство `AcceptStop` равно `True` (Истина). Пример определения такого списка показан в листинге 2.14.

Листинг 2.14. Получение списка сервисов с ограничением по условию

```
ServicesSet =
ПолучитьCOMОбъект("winmgmts:{impersonationLevel=impersonate}!\\" +
ИмяКомпьютера + "\root\cimv2");
Items = ServicesSet.ExecQuery("Select * from Win32_Service Where
AcceptStop = True");
Для каждого Item Из Items Цикл
  Сообщить("Имя процесса: " + Item.DisplayName);
КонецЦикла;
```

Установка приоритета процесса

Рассмотрим пример (листинг 2.15) установки среднего приоритета для запущенного процесса Notepad.exe (блокнот).

Листинг 2.15. Установка приоритета процесса

```
ABOVE_NORMAL = 32768;
ServicesSet =
ПолучитьCOMОбъект("winmgmts:{impersonationLevel=impersonate}!\\" +
ИмяКомпьютера + "\root\cimv2");
Items = ServicesSet.ExecQuery("Select * from Win32_Process Where Name =
```

```
'Notepad.exe'");
Для каждого Item Из Items Цикл
  Item.SetPriority(ABOVE_NORMAL);
КонецЦикла;
```

Завершение процесса

Для завершения процесса используется метод `Terminate` без параметров (листинг 2.16).

Листинг 2.16. Завершение процесса

```
ServicesSet = ПолучитьCOMОбъект
("winmgmts:{impersonationLevel=impersonate}!\\" +
ИмяКомпьютера + "\root\cimv2");
Items = ServicesSet.ExecQuery("Select * from Win32_Process Where Name =
'Notepad.exe'");
Для каждого Item Из Items Цикл
  Item.Terminate();
КонецЦикла;
```

Запуск и приостановка выполнения сервиса

Для запуска сервиса используется метод `StartService` (листинг 2.17). Для приостановки — метод `StopService`. Оба метода используются без параметров.

Листинг 2.17. Приостановление сервиса

```
ServicesSet = ПолучитьCOMОбъект
("winmgmts:{impersonationLevel=impersonate}!\\" +
ИмяКомпьютера + "\root\cimv2");
Items = ServicesSet.ExecQuery("Select * from Win32_Service Where Name
='Alerter'"); // Оповещатель
Для каждого Item Из Items Цикл
  Item.StartService(); // Запустить процесс
КонецЦикла;
```

В запросе метода `ExecQuery` используется фильтрация списка сервисов по имени `Alerter` (`Оповещатель`). Таким образом, в выборку попадает всего один сервис — `Оповещатель`, который и запустится после выполнения метода `StartService`.

Работа с операционной системой

Для работы с операционной системой в WMI существует несколько классов, основным из которых является — `Win32_OperatingSystem`. Рассмотрим основные свойства данного класса:

- `Name` — название операционной системы;
- `BootDevice` — имя физического диска, с которого загружается операционная сиситема;
- `SystemDrive` — символ логического диска, на котором установлена система;
- `WindowsDirectory` — папка, в которой установлена система;
- `Description` — описание к объекту класса;
- `FreePhysicalMemory` — размер свободной физической памяти (в килобайтах);
- `FreeVirtualMemory` — размер свободной виртуальной памяти (в килобайтах);
- `Manufacturer` — производитель операционной системы. Обычно данное свойство имеет значение "Microsoft Corporation";
- `NumberOfProcesses` — количество запущенных процессов;
- `Organization` — название организации, на которую зарегистрирована данная копия системы;
- `RegisteredUser` — имя пользователя, на которого зарегистрирована данная копия системы;
- `SerialNumber` — регистрационный номер;
- `OSLanguage` — языковая версия системы. Например `0009` — английская, `0419` — русская и т. д.;
- `OSType` — тип операционной системы. Например, `17` — WIN98, `18` — WINNT, `30` — SunOS и т. д.;
- `Primary` — признак основной операционной системы;
- `Version` — версия системы;
- `BuildNumber` — номер сборки операционной системы;

Основные методы класса `Win32_OperatingSystem`:

- `Reboot()` — перезагрузить систему;
- `Shutdown()` — выключить компьютер;
- `Win32Shutdown(Flag)` — управление функциями выключения компьютера (дляWin32-операционных систем). Параметр `Flag` может принимать следующие значения:
  - `0` — завершить работу пользователя (Log Off);

- 4 — принудительное завершение работы пользователя (Forced Log Off);
- 1 — выключение компьютера (Shutdown);
- 5 — принудительное выключение компьютера (Forced Shutdown);
- 2 — перезагрузка (Reboot);
- 6 — принудительная перезагрузка (Forced Reboot);
- 8 — выключение питания (Power Off);
- 12 — принудительное выключение питания (Forced Power Off).

Определение информации об ОС

Рассмотрим пример вывода основных свойств операционной системы. В программном коде, представленном в листинге 2.18, показан пример вывода названия, пути установки и номера версий установленных операционных систем (их может быть несколько).

Листинг 2.18. Определение информации об ОС

```
ServicesSet = Locator.ConnectServer(ИмяКомпьютера, "root\cimv2",
Пользователь, Пароль);
Items = ServicesSet.ExecQuery("Select * from Win32_OperatingSystem");
Для каждого Item Из Items Цикл
  Сообщить("Название: " + Item.Caption);
  Сообщить("Путь: " + Item.WindowsDirectory);
  Сообщить("Версия: " + Item.Version);
КонецЦикла;
```

Выключение компьютера

Рассмотрим пример выключения компьютера с помощью метода Win32Shutdown (листинг 2.19).

Листинг 2.19. Выключение компьютера

```
ServicesSet=ПолучитьCOMОбъект("winmgmts:{impersonationLevel=impersonate,
(Shutdown)}!\\" + ИмяКомпьютера + "\root\cimv2");
Items = ServicesSet.ExecQuery("Select * from Win32_OperatingSystem");
Для каждого Item Из Items Цикл
  Item.Win32Shutdown(5); // Выключение
КонецЦикла;
```

Определение разрешения экрана

Для определения разрешения экрана используются свойства `ScreenWidth` (ширина) и `ScreenHeight` (высота) класса `Win32_DesktopMonitor` (листинг 2.20).

Листинг 2.20. Определение разрешения экрана

```
ServicesSet = Locator.ConnectServer(ИмяКомпьютера, "root\cimv2",
Пользователь, Пароль);
Items = ServicesSet.ExecQuery("Select * from Win32_DesktopMonitor");
Для каждого Item Из Items Цикл
  Сообщить("Разрешение: " + Item.ScreenWidth + " x " + Item.ScreenHeight);
КонецЦикла;
```

Список элементов автозагрузки

Для определения списка программ, которые загружаются при запуске операционной системы Windows, предназначен класс `Win32_StartupCommand` (листинг 2.21).

Листинг 2.21. Получение списка элементов автозагрузки

```
ServicesSet = Locator.ConnectServer(ИмяКомпьютера, "root\cimv2",
Пользователь, Пароль);
Items = ServicesSet.ExecQuery("Select * from Win32_StartupCommand");
Для каждого Item Из Items Цикл
  Сообщить("Команда: " + Item.Command);
  Сообщить("Описание: " + Item.Description);
  Сообщить("Путь: " + Item.Location);
  Сообщить("Имя: " + Item.Name);
  Сообщить("Пользователь: " + Item.User);
КонецЦикла;
```

Активизация копии Windows XP/Server 2003

Для активации операционной системы Windows XP или Windows Server 2003 используется метод `ActivateOnline()` класса `Win32_WindowsProductActivation` (листинг 2.22).

Листинг 2.22. Активизация копии Windows XP/Server 2003

```
ServicesSet = Locator.ConnectServer(ИмяКомпьютера, "root\cimv2",
Пользователь, Пароль);
Items = ServicesSet.ExecQuery("Select * from
Win32_WindowsProductActivation");
Для каждого Item Из Items Цикл
  Item.ActivateOnline();
КонецЦикла;
```

Просмотр учетных записей и доменов

Для работы с учетными записями и доменами обычно используются следующие классы:

- `Win32_ComputerSystem` — содержит системную информацию компьютера;
- `Win32_Group` — содержит данные о пользовательских группах;
- `Win32_Account` — содержит данные о группах и пользователях системы.

Рассмотрим основные свойства класса `Win32_ComputerSystem`:

- `Description` — описание объекта;
- `Domain` — имя домена, в котором зарегистрирован компьютер;
- `DomainRole` — роль компьютера;
- `InfraredSupported` — признак поддержки работы с инфракрасным портом;
- `Manufacturer` — производитель компьютера;
- `Model` — название компьютера;
- `NumberOfProcessors` — количество процессоров;
- `UserName` — имя пользователя, работающего в настоящее время с системой;
- `Workgroup` — имя рабочей группы.

Класс `Win32_ComputerSystem` имеет важный метод `Rename(строка)`, который может изменять сетевое имя компьютера. Метод имеет один параметр — `строка`, содержащий новое имя компьютера.

Так как классы `Win32_Group` и `Win32_Account` довольно простые, поэтому подробно они рассматриваться не будут.

Определение домена компьютера

Для определения домена, в котором находится компьютер, необходимо прочитать свойство `Domain` класса `Win32_ComputerSystem` (листинг 2.23).

Листинг 2.23. Определение домена компьютера

```
ServicesSet = Locator.ConnectServer(ИмяКомпьютера, "root\cimv2",
Пользователь, Пароль);
Computers = ServicesSet.ExecQuery("Select * from Win32_ComputerSystem");
Для каждого Computer Из Computers Цикл
  Сообщить("Имя: " + Computer.Name);
  Сообщить("Домен: " + Computer.Domain);
КонецЦикла;
```

Для определения домена так же можно использовать свойство `DNSDomain` класса `Win32_NetworkAdapterConfiguration`.

Определение роли компьютера

Для определения роли компьютера, т. е. является ли компьютер сервером или рабочей станцией, используется свойство `DomainRole` класса `Win32_ComputerSystem`. Данное свойство может возвращать одно из следующих значений:

- `0` — самостоятельная рабочая станция (Standalone Workstation);
- `1` — член рабочей станции (Member Workstation);
- `2` — самостоятельный сервер (Standalone Server);
- `3` — рядовой сервер, т. е. сервер, не имеющий статуса контроллера в конкретном домене (Member Server);
- `4` — резервный контроллер домена, т. е. компьютер в домене Windows NT, который содержит копию политики безопасности домена и его базы данных, а также служит резервом на случай, если главный контроллер домена недоступен (Backup Domain Controller);
- `5` — главный контроллер домена, т. е. компьютер в домене, который хранит главную копию базы данных учетных записей домена, проводит аутентификацию пользователей, а также может работать как файл-сервер, принт-сервер и сервер приложений (Primary Domain Controller).

Пример определения сетевой роли компьютера показан в листинге 2.24.

Листинг 2.24. Определение роли компьютера

```
ServicesSet = Locator.ConnectServer(ИмяКомпьютера, "root\cimv2",
Пользователь, Пароль);
Computers = ServicesSet.ExecQuery("Select * from Win32_ComputerSystem");
Для каждого Computer Из Computers Цикл
```

```
  Массив = Новый Массив();
  Массив.Добавить("Standalone Workstation");
  Массив.Добавить("Member Workstationn");
  Массив.Добавить("Standalone Server");
  Массив.Добавить("Member Server");
  Массив.Добавить("Backup Domain Controller");
  Массив.Добавить("Primary Domain Controller");

  Сообщить("Роль компьютера: " + Массив[Computer.DomainRole]);
КонецЦикла;
```

Определение имени текущего пользователя

Для определения имени пользователя, который в настоящее время подключен к компьютеру, используется свойство `UserName` класса `Win32_ComputerSystem` (листинг 2.25).

Листинг 2.25. Определение имени текущего пользователя

```
ServicesSet = Locator.ConnectServer(ИмяКомпьютера, "root\cimv2",
Пользователь, Пароль);
Computers = ServicesSet.ExecQuery("Select * from Win32_ComputerSystem");
Для каждого Computer Из Computers Цикл
  Сообщить("Пользователь: " + Computer.UserName);
КонецЦикла;
```

Определение локальных групп компьютера

Для определения только локальных групп компьютера используется класс `Win32_Group` с условием, что свойство `LocalAccount` равно `True` (Истина). Пример определения локальных групп приведен в листинге 2.26.

Листинг 2.26. Определение локальных групп компьютера

```
ServicesSet = Locator.ConnectServer(ИмяКомпьютера, "root\cimv2",
Пользователь, Пароль);
Items = ServicesSet.ExecQuery("Select * from Win32_Group
Where LocalAccount = True");
Для каждого Item Из Items Цикл
  Сообщить("Локальная запись: " + Item.LocalAccount);
```

```
  Сообщить("Имя группы: " + Item.Name);
  Сообщить("Идентификатор: " + Item.SID);
  Сообщить("Тип: " + Item.SIDType);
  Сообщить("Статус: " + Item.Status);
КонецЦикла;
```

Определение списка пользователей и групп

Для получения полного списка пользователей и групп используется класс `Win32_Account`. Данный класс имеет свойство `SIDType`, которое может принимать следующие значения:

- `1` — если объект представляет свойства пользователя;
- `2` — группы пользователей;
- `3` — домена;
- `4` — псевдонима;
- `5` — общие группы;
- `6` — удаленной учетной записи;
- `7` — ошибочной учетной записи;
- `8` — неизвестной учетной записи;
- `9` — компьютера.

Ниже приведен пример вывода только пользователей и групп (листинг 2.27).

Листинг 2.27. Определение списка пользователей и групп

```
ServicesSet = Locator.ConnectServer(ИмяКомпьютера, "root\cimv2",
Пользователь, Пароль);
Items = ServicesSet.InstancesOf("Win32_Account");

Для каждого Item Из Items Цикл
  Если Item.SIDType = 1 Тогда
    Сообщить("Пользователь: " + Item.Caption);
  ИначеЕсли (Item.SIDType = 2) или (Item.SIDType = 4)
        или (Item.SIDType = 5) Тогда
    Сообщить("Группа: " + Item.Caption);
  КонецЕсли;
КонецЦикла;
```

Важно отметить, что объекты данного класса имеют разные наборы свойств в зависимости от значения свойства `SIDType`.

Работа с системной датой и временем

Для работы с датой и временем в WMI используется класс `Win32_LocalTime`. Подробно останавливаться на нем не будем, т. к. он довольно простой.

Определение даты и времени

Приведем пример определения даты и времени, установленные на конкретном компьютере с помощью класса `Win32_LocalTime` (листинг 2.28).

Листинг 2.28. Определение даты и времени

```
ServicesSet = Locator.ConnectServer(ИмяКомпьютера, "root\cimv2",
Пользователь, Пароль);
Items = ServicesSet.ExecQuery("Select * from Win32_LocalTime");
Для каждого Item Из Items Цикл
  Сообщить("День: " + Item.Day);
  Сообщить("Месяц: " + Item.Month);
  Сообщить("Год: " + Item.Year);
  Сообщить("Час: " + Item.Hour);
  Сообщить("Минута: " + Item.Minute);
  Сообщить("Секунда: " + Item.Second);
КонецЦикла;
```

Определение временной зоны

Для определения временной зоны, установленной на компьютере, используется класс `Win32_TimeZone` (листинг 2.29). В данном классе имеется два свойства: для отображения имени временной зоны, соответствующей летнему времени (`DaylightName`), и стандартное имя (`StandardName`), которое соответствует временной зоне зимнего времени.

Класс `Win32_TimeZone` имеет следующие свойства для определения данных летнего времени:

- `DaylightName` — имя временной зоны;
- `DaylightDayOfWeek` — день недели (`1` — воскресенье, `2` — понедельник и т. д., `7` — суббота);
- `DaylightHour` — час дня;
- `DaylightMinute` — минута;
- `DaylightSecond` — секунда;

- DaylightDay — день;
- DaylightMonth — месяц;
- DaylightYear — год.

Для временной зоны зимнего времени существует такой же набор свойств, только префикс Daylight заменен на Standard.

Листинг 2.29. Определение названий временных зон и их времени

```
ServicesSet = Locator.ConnectServer(ИмяКомпьютера, "root\cimv2",
Пользователь, Пароль);
Items = ServicesSet.ExecQuery("Select * from Win32_TimeZone");
Для каждого Item Из Items Цикл
  Сообщить("Временная зона (светлое время суток): " + Item.DaylightName);
  Сообщить("Время: " + Item.DaylightHour + ":" + Item.DaylightMinute +
":" + Item.DaylightSecond);
  Сообщить("Временная зона (стандартное имя): " + Item.StandardName);
  Сообщить("Время: " + Item.StandardHour + ":" + Item.StandardMinute +
":" + Item.StandardSecond);
КонецЦикла;
```

Работа с журналами событий

Для работы с журналами событий в WMI существуют два класса:

- Win32_NTEventLogFile — работа с файлами журналов событий;
- Win32_NTLogEvent — просмотр записей журналов.

Класс Win32_NTEventLogFile имеет следующие свойства:

- Compressed — признак сжатия файла журнала событий;
- CompressionMethod — метод сжатия файла;
- Description — описание объекта;
- Drive — имя логического диска, на котором располагается файл;
- Encrypted — признак криптования файла;
- EncryptionMethod — метод криптования файла;
- FileName — имя файла;
- FileSize — размер файла в байтах;
- NumberOfRecords — количество записей в журнале.

Класс Win32_NTEventLogFile имеет следующие методы:

- Copy(FileName) — копирует файл журнала в имя, указанное в параметре FileName;

- `Rename(FileName)` — переименовывает файл журнала. Новое имя файла задается в параметре `FileName`;
- `Delete()` — удаляет файл журнала;
- `Compress()` — сжимает (архивирует) файл;
- `Uncompress()` — разархивирует файл журнала;
- `ClearEventLog()` — очистка записей журнала.

Свойства класса `Win32_NTLogEvent` рассматриваться не будет, т. к. их довольно немного и все предназначены только для просмотра журнала событий. Пример просмотра журнала событий показан в листинге 2.32.

Получение информации о журнале событий

Приведем пример просмотра количества записей журнала "System", максимально допустимого размера и имени файла (листинг 2.30).

Листинг 2.30. Получение информации о журнале событий

```
ServicesSet = Locator.ConnectServer(ИмяКомпьютера, "root\cimv2",
Пользователь, Пароль);
Items = ServicesSet.ExecQuery("Select * from Win32_NTEventLogFile where
LogFileName='System'");
Для каждого Item Из Items Цикл
  Сообщить("Количество записей в журнале: " + Item.NumberOfRecords);
  Сообщить("Максимальный размер файла: " + Item.MaxFileSize);
  Сообщить("Имя файла журнала: " + Item.Name);
КонецЦикла;
```

В запросе данного кода происходит фильтрация объектов по условию — метод `LogFileName` (имя журнала) равен строке `'System'`.

Копирование и очистка журнала

Иногда бывает необходимо сделать копию журнала событий, после чего очистить его. Для этих целей предназначены методы `Copy` и `ClearEventLog` класса `Win32_NTEventLogFile` (листинг 2.31).

Листинг 2.31. Копирование и очистка журнала событий

```
ServicesSet = Locator.ConnectServer(ИмяКомпьютера, "root\cimv2",
Пользователь, Пароль);
Items = ServicesSet.ExecQuery("Select * from Win32_NTEventLogFile where
```

```
LogFileName='Application'");
Для каждого Item Из Items Цикл
  Item.Copy("c:\application.evt"); // Копирование журнала
  Item.ClearEventLog(); // Очистка журнала от записей
КонецЦикла;
```

Просмотр записей журнала

Для просмотра записей журнала используется класс `Win32_NTLogEvent`. Приведем пример просмотра записей журнала "System" (листинг 2.32).

Листинг 2.32. Просмотр записей журнала

```
ServicesSet = Locator.ConnectServer(ИмяКомпьютера, "root\cimv2",
Пользователь, Пароль);
Items = ServicesSet.ExecQuery("Select * from Win32_NTLogEvent where
Logfile='System'");
Для каждого Item Из Items Цикл
  Сообщить("Category: " + Item.Category);
  Сообщить("ComputerName: " + Item.ComputerName);
  Сообщить("Message: " + Item.Message);
  Сообщить("RecordNumber: " + Item.RecordNumber);
  Сообщить("TimeWritten: " + Item.TimeWritten);
  Сообщить("User: " + Item.User);
  Сообщить("————————————————————");
КонецЦикла;
```

Работа с файлами и папками

Для работы с файлами обычно применяется класс `CIM_DataFile`. Основные свойства класса:

- `Name` — полное имя файла;
- `Path` — путь к файлу;
- `Archive` — признак архивного файла;
- `System` — признак системного файла;
- `Compressed` — признак сжатого файла;
- `Drive` — символ (идентификатор) логического диска, на котором записан файл;

- `Extension` — расширение файла;
- `FileSize` — размер файла;
- `EightDotThreeFileName` — DOS-совместимое имя файла (например, c:\progra~1).

Основные методы класса:

- `Compress()` — сжимает файл;
- `Copy(FileName)` — копирует файл (новое имя файла определяется параметром `FileName`);
- `Delete()` — удаляет файл;
- `Rename(FileName)` — переименовывает файл (новое имя файла определяется параметром `FileName`).

В листингах 2.33 и 2.34 приведены примеры работы со свойствами и методами данного класса.

Переименование файлов

Приведем пример переименования файла ReadMe.txt, который находится в папке c:\Program Files\1Cv77 (листинг 2.33).

Листинг 2.33. Переименование файлов

```
ServicesSet = ПолучитьCOMОбъект("winmgmts:!\\" + ИмяКомпьютера +
"\root\cimv2");
Items = ServicesSet.ExecQuery("Select * from Cim_Datafile where Name =
'c:\\Program Files\\1Cv77\\ReadMe.txt'");
Для каждого Item Из Items Цикл
  Результат = Item.Rename("c:\\Program Files\\1Cv77\\ReadMe.bak");
КонецЦикла;
```

Поиск всех файлов по расширению

Для того чтобы найти все файлы пользователя по расширению, используется запрос к классу `CIM_DataFile` с условием по расширению (свойство `Extension`). Пример поиска всех файлов с расширением MP3 приведен в листинге 2.34.

Листинг 2.34. Поиск всех файлов по расширению

```
ServicesSet = Locator.ConnectServer(ИмяКомпьютера, "root\cimv2",
Пользователь, Пароль);
Items = ServicesSet.ExecQuery("Select * from CIM_DataFile where Extension
= 'mp3'");
```

```
Для каждого Item Из Items Цикл
  Сообщить("Имя файла: " + Item.Name);
КонецЦикла;
```

Открытие общего доступа к папке

Для управления доступом к папкам в WMI используется класс `Win32_Share`. Для создания доступа к папке предназначен метод `Create`, который имеет следующие параметры:

- `Path` — путь к папке;
- `Name` — имя общего ресурса;
- `Type` — тип общего ресурса (`0` — папка или диск, `1` — принтер, `2` — какое-либо устройство);
- `MaximumAllowed` — максимальное количество подключений;
- `Description` — описание.

Для удаления доступа предназначен метод `Delete` без параметров.

Пример создания общего доступа к папке C:\Program Files приведен в листинге 2.35).

Листинг 2.35. Открытие общего доступа к папке

```
FILE_SHARE = 0;
MAXIMUM_CONNECTIONS = 25;
ServicesSet =
ПолучитьCOMОбъект("winmgmts:{impersonationLevel=impersonate}!\\" +
ИмяКомпьютера + "\root\cimv2");
NewShare = ServicesSet.Get("Win32_Share");
Попытка
  Результат = NewShare.Create("C:\Program Files", "Share4You",
FILE_SHARE, MAXIMUM_CONNECTIONS, "Общий доступ.");
Исключение
  Сообщить("Ошибка. Невозможно открыть доступ.");
КонецПопытки;
```

Работа с принтерами

Для работы с принтерами используется класс `Win32_Printer`.

Рассмотрим основные свойства данного класса:

- `Name` — имя принтера;
- `Comment` — комментарий к принтеру;

- `Default` — признак принтера по умолчанию;
- `Description` — описание;
- `DriverName` — название драйвера принтера;
- `Local` — признак локального принтера;
- `Network` — признак сетевого принтера;
- `Shared` — признак наличия общего доступа к принтеру;
- `PortName` — имя порта принтера;
- `PrinterStatus` — текущий статус принтера (`1` — прочий, `2` — неизвестный, `3` — ожидает, `4` — печатает, `5` — подготавливается, `6` — остановлен, `7` — выключен).

Класс `Win32_Printer` имеет следующие методы:

- `AddPrinterConnection(Name)` — добавить новый сетевой принтер с сетевым именем `Name`;
- `SetDefaultPrinter()` — установить принтер по умолчанию;
- `CancelAllJobs()` — отменить все задания;
- `Pause()` — приостановить печать;
- `Resume()` — возобновить печать;
- `PrintTestPage()` — распечатать тестовую страницу;
- `RenamePrinter(NewPrinterName)` — переименовать принтер (новое имя принтера задается в параметре `NewPrinterName`).

Добавление нового сетевого принтера

Приведем пример добавления сетевого принтера с сетевым именем "\\OfficeServer\MainOfficePrinter" (листинг 2.36).

Листинг 2.36. Добавление нового сетевого принтера

```
ServicesSet =
ПолучитьCOMОбъект("winmgmts:{impersonationLevel=impersonate}!\\" +
ИмяКомпьютера + "\root\cimv2");
NewPrinter = ServicesSet.Get("Win32_Printer");
Попытка
  Результат = NewPrinter.AddPrinterConnection
("\\OfficeServer\MainOfficePrinter");
Исключение
  Сообщить("Ошибка. Невозможно добавить принтер.");
КонецПопытки;
```

Установка принтера по умолчанию

Пример установки принтера по умолчанию приведен в листинге 2.37.

Листинг 2.37. Установка принтера по умолчанию

```
ИмяПринтера = "MainOfficePrinter";
ServicesSet = ПолучитьCOMОбъект("winmgmts:{impersonationLevel=
impersonate}!\\" + ИмяКомпьютера + "\root\cimv2");
Printers = ServicesSet.Get("Win32_Printer");
Items = ServicesSet.ExecQuery("Select * from Win32_Printer Where Name =
'" + ИмяПринтера + "'");
Для каждого Item Из Items Цикл
  Item.SetDefaultPrinter();
КонецЦикла;
```

В данном примере, в запросе, используется конструкция `Where` для фильтрации объектов коллекции по имени (свойство `Name`).

Работа с заданиями планировщика

Для работы с заданиями планировщика используется класс `Win32_ScheduledJob`. Основные свойства данного класса определены на примере просмотра заданий (листинг 2.38).

Класс `Win32_ScheduledJob` имеет два метода.

- `Create` — добавить задание. Метод имеет следующие параметры:
  - `Command` — путь к программе, которая будет запускаться;
  - `StartTime` — время запуска в UTC-формате (Universal Time Coordinated — универсальное синхронизированное время), при этом общее представление формата можно записать в виде — "YYYYMMDDHHMMSS.MMMMMM(+-)OOO", где "YYYYMMDD" можно заменить на "********" (например, дата "********123000.000000-420" означает запуск в 12:30 после полудня;
  - `RunRepeatedly` — признак многократного выполнения;
  - `DaysOfWeek` — день недели (`1` — понедельник, `2` — вторник, `4` — среда, `8` — четверг, `16` — пятница, `32` — суббота, `64` — воскресенье);
  - `DaysOfMonth` — число месяца;
  - `InteractWithDesktop` — признак взаимодействия с рабочим столом;
  - `JobId` — идентификационный номер задания.
- `Delete()` — удалить задание.

Просмотр заданий в планировщике

Рассмотрим пример просмотра всех заданий планировщика (листинг 2.38).

Листинг 2.38. Просмотр заданий в планировщике

```
ServicesSet = ПолучитьCOMОбъект("winmgmts:{impersonationLevel=
impersonate}!\\" + ИмяКомпьютера + "\root\cimv2");
Items = ServicesSet.ExecQuery("Select * from Win32_ScheduledJob");
Для каждого Item Из Items Цикл
  Сообщить("Заголовок: " + Item.Caption);
  Сообщить("Команда: " + Item.Command);
  Сообщить("День месяца: " + Item.DaysOfMonth);
  Сообщить("День недели: " + Item.DaysOfWeek);
  Сообщить("Описание: " + Item.Description);
  Сообщить("Продолжительность: " + Item.ElapsedTime);
  Сообщить("Дата инсталляции: " + Item.InstallDate);
  Сообщить("Имя: " + Item.Name);
  Сообщить("Оповещение: " + Item.Notify);
  Сообщить("Приоритет: " + Item.Priority);
  Сообщить("Время старта: " + Item.StartTime);
  Сообщить("Статус: " + Item.Status);
  Сообщить("=====================================");
КонецЦикла;
```

Создание заданий

Для создания задания используется метод `Create`. Приведем пример ежемесячного запуска программы "Блокнот" (notepad.exe) каждого второго числа месяца в 23:30 (листинг 2.39).

Листинг 2.39. Создание заданий в планировщике

```
Перем Номер;
ServicesSet = ПолучитьCOMОбъект("winmgmts:{impersonationLevel=
impersonate}!\\" + ИмяКомпьютера + "\root\cimv2");
ObjectSet = ПолучитьCOMОбъект("WinMgmts:Win32_ScheduledJob");
ObjectSet.Create("notepad.exe", "********123000.000000-
420",True,1,2,True,Номер);
Сообщить("Заданию присвоен номер: " + Номер);
```

Работа с системным реестром

Для работы с системным реестром предназначен класс `StdRegProv`, принадлежащий пространству имен root\default. Данный класс совсем не имеет свойств, все действия и значения ключей доступны только через методы.

- `CreateKey(DefKey, SubKeyName)` — создание нового ключа. Параметры метода могут принимать значения:
  - `DefKey` — идентификатор корневого имени раздела, определяется значениями:
    - `2147483648` — HKEY_CLASSES_ROOT;
    - `2147483649` — HKEY_CURRENT_USER;
    - `2147483650` — HKEY_LOCAL_MACHINE;
    - `2147483651` — HKEY_USERS;
    - `2147483653` — HKEY_CURRENT_CONFIG;
    - `2147483654` — HKEY_DYN_DATA;
  - `SubKeyName` — имя создаваемого ключа (раздела).
- `DeleteKey(DefKey, SubKeyName)` — удаление раздела. Параметры метода аналогичны методу `CreateKey`.
- `DeleteValue(DefKey, SubKeyName, ValueName)` — удаление параметра. Параметры `DefKey` и `SubKeyName` аналогичны параметрам метода `CreateKey`, параметр `ValueName` задает имя удаляемого параметра.
- `GetDWORDValue(DefKey, SubKeyName, ValueName, Value)` — получение значения параметра типа `DWORD`. Параметры `DefKey`, `SubKeyName` и `ValueName` аналогичны параметрам метода `DeleteValue`. В параметр `Value` возвращается значение ключа `ValueName`.
- `GetStringValue(DefKey, SubKeyName, ValueName, Value)` — получение строкового параметра. Все параметры аналогичны параметрам метода `GetDWORDValue`.
- `GetExpandedStringValue(DefKey, SubKeyName, ValueName, Value)` — получение значения расширяемого строкового параметра. Все параметры аналогичны параметрам метода `GetStringValue`.
- `SetDWORDValue(DefKey, SubKeyName, ValueName, Value)` — создание нового параметра типа `DWORD`. Здесь параметр `Value` задает значение ключа, остальные аналогичны параметрам метода `GetStringValue`.
- `SetStringValue(DefKey, SubKeyName, ValueName, Value)` — создание строкового параметра. Все параметры аналогичны параметрам метода `SetDWORDValue`.

- `SetExpandedStringValue(DefKey, SubKeyName, ValueName, Value)` — создание параметра расширяемого строкового типа. Все параметры аналогичны параметрам метода `SetDWORDValue`.

Чтение значений ключей реестра

Для чтения значений ключей реестра используются методы `GetStringValue`, `GetExpandedStringValue` и `GetStringValue`, в зависимости от типа значения параметра.

Рассмотрим пример чтения значения параметра с именем `"HistoryBufferSize"`, находящимся в разделе HKEY_CURRENT_USER\ Console (листинг 2.40).

Листинг 2.40. Чтение значений ключей реестра

```
Перем Значение;

HKEY_CURRENT_USER = 2147483649;
ПутьККлючу = "Console";
ИмяПараметра = "HistoryBufferSize";

Reg = ПолучитьCOMОбъект("winmgmts:{impersonationLevel=impersonate}!\\" +
ИмяКомпьютера + "\root\default:StdRegProv");
Reg.GetDWORDValue(HKEY_CURRENT_USER, ПутьККлючу, ИмяПараметра, Значение);
Сообщить("Значение: " + Значение);
```

Создание нового раздела

Приведем пример создания нового раздела с помощью класса `StdRegProv` (листинг 2.41).

Листинг 2.41. Создание нового ключа раздела

```
HKEY_LOCAL_MACHINE = 2147483650;
Reg = ПолучитьCOMОбъект("winmgmts:{impersonationLevel=impersonate}!\\" +
ИмяКомпьютера + "\root\default:StdRegProv");
ПутьККлючу = "SOFTWARE\NewKey";
Reg.CreateKey(HKEY_LOCAL_MACHINE, ПутьККлючу);
```

В приведенном примере создается раздел HKEY_LOCAL_MACHINE\ SOFTWARE\ NewKey.

Создание нового параметра

Рассмотрим пример создания нового параметра реестра (листинг 2.42). В примере создается параметр "Форум для вопросов" со значением "http://forum.erpg.ru" в разделе HKEY_LOCAL_MACHINE\ SOFTWARE\ NewKey.

Листинг 2.42. Создание нового параметра реестра

```
HKEY_LOCAL_MACHINE = 2147483650;
ПутьККлючу = "SOFTWARE\NewKey";
Reg = ПолучитьCOMОбъект("winmgmts:{impersonationLevel=impersonate}!\\" +
ИмяКомпьютера + "\root\default:StdRegProv");
ИмяПараметра = "Форум для вопросов";
Значение = "http://forum.erpg.ru";
Reg.SetExpandedStringValue(HKEY_LOCAL_MACHINE, ПутьККлючу, ИмяПараметра,
Значение);
```

Работа с сетью

Для работы с сетью в WMI предназначены два основных класса: Win32_NetworkAdapterConfiguration и Win32_NetworkAdapter. Перечислять все свойства и методы не имеет смысла, т. к. их огромное количество. Назначение многих очень специфичные и будут понятны только сетевым администраторам.

Рассмотрим только некоторые из них.

Отключение сетевого соединения

Если вы используете протокол динамической конфигурации хоста (DHCP, Dynamic Host Configuration Protocol), то отключить сетевое соединение можно методом ReleaseDHCPLease класса Win32_NetworkAdapterConfiguration (листинг 2.43). Иначе, с помощью WMI, это выполнить невозможно. Для возобновления соединения используется метод RenewDHCPLease.

Листинг 2.43. Отключение сетевого соединения

```
ServicesSet = ПолучитьCOMОбъект("winmgmts:{impersonationLevel=
impersonate}!\\" + ИмяКомпьютера + "\root\cimv2");
Items = ServicesSet.ExecQuery("Select * From
Win32_NetworkAdapterConfiguration Where IPEnabled = True");
Для каждого Item Из Items Цикл
  Item.ReleaseDHCPLease();
КонецЦикла;
```

Определение MAC- и IP-адресов

Для определения IP-адреса, который назначен данному сетевому компьютеру, необходимо сначала определить его MAC-адрес, используя класс `Win32_NetworkAdapter`. А затем, используя класс `Win32_NetworkAdapterConfiguration`, найти IP-адрес, соответствующий данному MAC-адресу. Пример реализации такого механизма поиска приведен в листинге 2.44.

Листинг 2.44. Определение MAC- и IP-адресов

```
MACAddress = "";
ServicesSet = ПолучитьCOMОбъект("winmgmts:{impersonationLevel=
impersonate}!\\" + ИмяКомпьютера + "\root\cimv2");
Items = ServicesSet.ExecQuery("Select * From Win32_NetworkAdapter");
Для каждого Item Из Items Цикл
  MACAddress = Item.MACAddress;
КонецЦикла;

Если MACAddress <> "" Тогда
  Сообщить("MAC адрес: " + MACAddress);

  Items = ServicesSet.ExecQuery("Select * From
Win32_NetworkAdapterConfiguration");
  Для каждого Item Из Items Цикл
    Если Item.MACAddress = MACAddress Тогда
      Для каждого IPAddress Из Item.IPAddress Цикл
        Сообщить("IP адрес: " + IPAddress);
      КонецЦикла;
    КонецЕсли;
  КонецЦикла;
КонецЕсли;
```

Использование команды Ping

Для проверки доступности адресата путем передачи ему специального сигнала можно использовать программу ping.exe. Средствами WMI это можно сделать с помощью проверки значения свойства `StatusCode` класса `Win32_PingStatus` (листинг 2.45).

В качестве значения свойства `Address`, в запросе, может использоваться как имя компьютера (Hostname), так и IP-адрес.

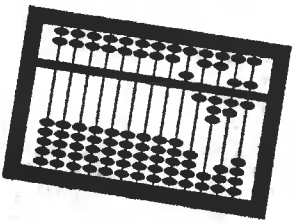
Листинг 2.45. Использование команды Ping

```
//IPAddress = "172.20.129.102"; // IP-адрес
IPAddress = "MAVCOMP"; // Hostname

ServicesSet = ПолучитьCOMОбъект("winmgmts:\\" + ИмяКомпьютера +
"\root\cimv2");
Items = ServicesSet.ExecQuery("Select * From Win32_PingStatus
where Address = '" + IPAddress + "'");

Для каждого Item Из Items Цикл
  Если (Item.StatusCode = NULL) или (Item.StatusCode <> 0) Тогда
    Сообщить("Компьютер не отвечает.");
  Иначе
    Сообщить("Компьютер ответил.");
  КонецЕсли;

КонецЦикла;
```



Глава 3

Использование скриптов WSH

Что такое WSH

Windows Script Host (*WSH*) — это стандартный компонент операционной системы Windows, который позволяет без предварительной компиляции непосредственно в операционной системе запускать сценарии, написанные на любых языках, поддерживающих технологию COM. Собственная объектная модель WSH позволяет из сценариев работать с файловой системой, системным реестром, специальными папками и ярлыками Windows, ресурсами локальной сети, а также запускать процессы и контролировать ход их выполнения. Кроме этого, в самих сценариях WSH можно использовать любые серверы автоматизации (COM-объекты), зарегистрированные в системе.

Примечание

WSH встроена в операционные системы Microsoft Windows 98/ME/NT/2000/XP/Server 2003. Для Windows 95 Windows Script Host можно загрузить с сайта Microsoft — **http://msdn.microsoft.com/scripting.**

Объектная модель WSH

Для того чтобы воспользоваться всеми возможностями, которые представляет эта технология, нужно разобраться в ее структуре. Windows Script Host состоит из целого ряда объектов.

- `Wscript` — это главный объект WSH, который содержит информацию о сервере сценариев WSH и позволяет создавать COM-объекты, выдавать сообщения в операционную систему, вводить данные с клавиатуры и т. п. Так как данный объект позволяет выполнять все те базовые действия, которые без особых трудностей можно выполнить в среде "1С:Предприятие", то рассматривать этот объект в контексте данной книги будет не уместно.

- `WshArguments`, `WshNamed` и `WshUnnamed` — объекты, обеспечивающие доступ к параметрам командной строки запущенного сценария Windows. Данные объекты используются исключительно во время запуска скриптов из операционной системы, поэтому их рассмотрение в рамках данной книги бессмысленно.
- `WshShell` — объект, который позволяет запускать независимые процессы, создавать ярлыки, работать с переменными среды, системным реестром и специальными папками Windows.
- `WshSpecialFolders` — объект, обеспечивающий доступ к специальным папкам Windows.
- `WshShortout` — объект, который обеспечивает работу с ярлыками Windows.
- `WshUrlShortout` — объект, предназначенный для работы с ярлыками сетевых ресурсов.
- `WshEnvironment` — объект, предназначенный для работы с переменными среды (для просмотра, изменения и удаления переменных среды).
- `WshNetwork` — объект, использующийся при работе с локальной сетью. Он содержит сетевую информацию для локального компьютера, а также позволяет подключать сетевые диски и принтеры.
- `WshScriptExec` — объект, позволяющий запускать консольные приложения в качестве дочерних процессов. Обеспечивает контроль этих приложений и доступ к их стандартным входным и выходным потокам.
- `WshController` — объект, управляющий запуском сценариев на удаленных машинах.
- `WshRemote` — объект, позволяющий управлять сценарием, запущенным на удаленной машине.
- `WshRemoteError` — объект, используемый для получения информации об ошибке, возникшей в результате выполнения сценария, запущенного на удаленной машине.
- `FileSystemObject` — дополнительный объект, обеспечивающий доступ к файловой системе компьютера.

Запуск произвольного VB-скрипта (VBS)

В настоящее время очень распространены VB-скрипты, которые пришли на замену пакетным файлам (bat). Очень часто возникает необходимость в запуске того или иного скрипта из системы "1С:Предприятие". Однако времени, а зачастую и возможности переводить код из одного языка программирования в другой, нет. Как раз для такой ситуации и используется

COM-объект — MSScriptControl, который позволяет запустить произвольный скрипт из контекста "1С:Предприятия". Объект MSScriptControl не входит в объектную модель WSH, но часто применяется в "скриптовых" языках.

Рассмотрим пример запуска VB-скрипта и вывода его результата в окно сообщений системы "1С:Предприятие" (листинг 3.1).

Листинг 3.1. Запуск произвольного VB-скрипта (VBS) для версии 8.0

```
// Произвольный скрипт
ТекстСкрипта = "Function VBSFunction()
|  GetDefaultPrinter = vbNullString
|  Set objWMIService = GetObject(""winmgmts:"" & ""{impersonationLevel=
impersonate}!\\.\root\cimv2"")
|  Set colInstalledPrinters = objWMIService.ExecQuery(""Select * from
Win32_Printer"")
|  For Each objPrinter in colInstalledPrinters
|    If objPrinter.Attributes and 4 Then
|      VBSFunction=objPrinter.Name
|      Exit For
|    End If
|  Next
|End Function";

// Создание объекта
scrptCtrl = Новый COMОбъект("MSScriptControl.ScriptControl");
scrptCtrl.Language="vbscript"; // Указание языка
// Добавление исполняемого кода
scrptCtrl.AddCode(ЭлементыФормы.ПолеТекстовогоДокумента.ПолучитьТекст());
// Запуск функции по имени и вывод результата
Сообщить(scrptCtrl.run("VBSFunction"));
```

На самом скрипте останавливаться не будем, т. к. это тема другой главы. Здесь только следует отметить, что он возвращает имя принтера, используемого в системе по умолчанию. Вообще в переменную ТекстСкрипта можно записать любую функцию на VB-скрипте, а потом выполнить эту функцию, выполнив метод run объекта MSScriptControl.ScriptControl с указанием имени запускаемой функции.

Примечание

Рассмотренный выше код для версии 7.7 будет аналогичным, за исключением создания COM-объекта.

Выполнение операций с файловой системой

Для работы с файловой системой из сценариев WSH предназначен объект `FileSystemObject`. С помощью этого объекта можно выполнить следующие основные действия:

- создавать каталоги;
- создавать или открывать текстовые файлы;
- копировать или перемещать файлы и каталоги;
- удалять файлы и каталоги;
- создавать объекты `Drive`, `Folder` и `File` для доступа к конкретному диску, каталогу или файлу соответственно.

С помощью свойств объектов `Drive`, `Folder` и `File` можно получить детальную информацию о тех элементах файловой системы, с которыми они ассоциированы. Объекты `Folder` и `File` также предоставляют методы манипулирования файлами и каталогами (создание, удаление, копирование, перемещение); эти методы в основном копируют соответствующие методы объекта `FileSystemObject`.

Кроме того, имеются три объекта-коллекции: `Drives`, `Folders` и `Files`. Коллекция `Drives` содержит объекты `Drive` для всех имеющихся в системе дисков, `Folders` — объекты `Folder` для всех подкаталогов заданного каталога, `Files` — объекты `File` для всех файлов, находящихся внутри определенного каталога.

Наконец, из сценария можно читать информацию текстовых файлов и записывать в них данные. Методы для этого предоставляет объект `TextStream`.

С помощью объекта `FileSystemObject` можно очень просто написать некое подобие файлового менеджера "Проводник". Рассмотрим основные операции с файловой системой.

Получение сведений о дисках

Для получения сведений об определенном диске предназначен объект `Drive`. Данный объект позволяет получить информацию о типе файловой системы, метки тома, общем объеме и количестве свободного места и т. д.

Сам объект `Drive` создается с помощью метода `GetDrive` объекта `FileSystemObject`. Приведем пример вывода всех доступных свойств диска (листинг 3.2).

Листинг 3.2. Вывод информации о диске

```
// Создаем массив типов дисков
ТипыДисков = Новый Массив();
ТипыДисков.Добавить("Неизвестно");
ТипыДисков.Добавить("Съемный");
ТипыДисков.Добавить("Постоянный");
ТипыДисков.Добавить("Сетевой");
ТипыДисков.Добавить("CD-ROM");
ТипыДисков.Добавить("RAM-диск");

// Создаем объект FileSystemObject
FSO = Новый COMОбъект("Scripting.FileSystemObject");
// Получаем ссылку на объект
Drive = FSO.GetDrive("C");

// Выводим свойства диска
Сообщить("Информация о диске C:");
Сообщить("Метка диска: " + Drive.VolumeName);
Сообщить("Размер диска: " + Drive.TotalSize/1024 + " Кбайт");
Сообщить("Свободно: " + Drive.FreeSpace/1024 + " Кбайт");
Сообщить("Буква диска: " + Drive.DriveLetter);
Сообщить("Тип: " + ТипыДисков[Drive.DriveType]);
Сообщить("Файловая система: " + Drive.FileSystem);
Сообщить("Готов к чтению: " + Drive.IsReady);
Сообщить("Путь: " + Drive.Path);
Сообщить("Начальная папка: " + Drive.RootFolder);
Сообщить("Серийный номер: " + Drive.SerialNumber);
Сообщить("Сетевое имя: " + Drive.ShareName);

// Изменяем метку диска
Drive.VolumeName = "www.erpg.ru";
```

Из всех используемых в коде свойств только `VolumeName` доступно как для чтения, так и для записи, т. е. с помощью этого свойства можно изменять метку диска.

Получение сведений о папках и файлах

Для получения сведений о папках и файлах используются объекты `Folder` и `File`. Данные объекты позволяют определить дату создания или последнего обращения к папке или файлу, их размер, атрибуты и т. д. Сами объекты создаются с помощью методов `GetFolder` и `GetFile` объекта `FileSystemObject`.

Рассмотрим пример вывода всех доступных свойств папки (листинг 3.3).

Листинг 3.3. Получение сведений о папке

```
// Функция возвращает строку с атрибутами файла File.
// Вызывается из основного кода при получении сведений о папке или файле
Функция ВернутьАтрибутыФайла(File)
  Attr = ПереводИз10(File.Attributes, 2);
  // Функция ПереводИз10 переводит число из 10 системы счисления
  // в произвольную, в данном случае - двоичную.

  Если Attr = 0 Тогда
    Возврат "Обычный";
  КонецЕсли;

  Стр = "";
  Если (Сред(Attr,СтрДлина(Attr)-4,1) = "1") и (СтрДлина(Attr) > 4)
       Тогда Стр = Стр + "Папка " КонецЕсли;
  Если (Сред(Attr,СтрДлина(Attr),1) = "1") и (СтрДлина(Attr) > 0)
       Тогда Стр = Стр + "Только чтение " КонецЕсли;
  Если (Сред(Attr,СтрДлина(Attr)-1,1) = "1") и (СтрДлина(Attr) > 1)
       Тогда Стр = Стр + "Скрытый " КонецЕсли;
  Если (Сред(Attr,СтрДлина(Attr)-2,1) = "1") и (СтрДлина(Attr) > 2)
       Тогда Стр = Стр + "Системный " КонецЕсли;
  Если (Сред(Attr,СтрДлина(Attr)-3,1) = "1") и (СтрДлина(Attr) > 3)
       Тогда Стр = Стр + "Метка " КонецЕсли;
  Если (Сред(Attr,СтрДлина(Attr)-5,1) = "1") и (СтрДлина(Attr) > 5)
       Тогда Стр = Стр + "Архивный " КонецЕсли;
  Если (Сред(Attr,СтрДлина(Attr)-10,1) = "1") и (СтрДлина(Attr) > 10)
       Тогда Стр = Стр + "Синоним " КонецЕсли;
  Если (Сред(Attr,СтрДлина(Attr)-11,1) = "1") и (СтрДлина(Attr) > 11)
       Тогда Стр = Стр + "Сжатый " КонецЕсли;
```

```
  Возврат Стр;
КонецФункции

// Основная часть кода.
// Создаем объект FileSystemObject
FSO = Новый COMОбъект("Scripting.FileSystemObject");

// Получаем ссылку на объект
Folder = FSO.GetFolder("c:\Program Files\");

// Выводим свойства папки
Сообщить("Информация о папке c:\Program Files\");
Сообщить("Атрибуты: " + ВернутьАттрибутыФайла(Folder));
Сообщить("Дата создания: " + Folder.DateCreated);
Сообщить("Дата последнего доступа: " + Folder.DateLastAccessed);
Сообщить("Дата последнего изменения: " + Folder.DateLastModified);
Сообщить("Признак корневой папки: " + Folder.IsRootFolder);
Сообщить("Имя: " + Folder.Name); //!!
Сообщить("Родительская папка: " + Folder.ParentFolder);
Сообщить("Путь: " + Folder.Path);
Сообщить("Короткое имя: " + Folder.ShortName);
Сообщить("Короткий путь: " + Folder.ShortPath);
Сообщить("Размер: " + Folder.Size);
Сообщить("Доступ к дочерним папкам: " + Folder.SubFolders);
Сообщить("Type: " + Folder.Type);
```

Объект `File` имеет подобный набор свойств.

Примечание

Свойство `Name` доступно для записи, т. е. с помощью его можно переименовать файл или папку.

Отдельного внимания заслуживает свойство `SubFolders`, которое предоставляет доступ к дочерним папкам. Данное свойство возвращает коллекцию `Folders`, которая содержит объекты `Folder`. Элементы коллекции системы "1С:Предприятие" в версии 8.0 можно получить с помощью конструкции `Для каждого...из...Цикл...КонецЦикла`. Для версии 7.7 это можно сделать только с помощью внешней компоненты или COM-объекта (пример такого COM-объекта был рассмотрен в *главе 2*).

Проверка существования определенного диска, каталога или файла

Для проверки существования определенного диска, каталога или файла предназначены методы `DriveExists`, `FolderExists` и `FileExists` объекта `FileSystemObject`.

Данные методы возвращают `Истина` (в версии 7.7 метод возвращает -1), если диск, папка или файл, переданный в качестве параметров найден, иначе — `Ложь` (в версии 7.7 метод возвращает 0).

Обычно данные методы применяются перед какими-либо действиями над дисками, файлами или папками, чтобы удостовериться, что они реально существуют и дальнейшая работа с ними не вызовет ошибок.

Приведем простой пример проверки наличия файла (листинг 3.4).

Листинг 3.4. Проверка на наличие файла

```
ИмяФайла = "c:\Program Files\1cv8\bin\1cv8.exe";
FSO = Новый COMОбъект("Scripting.FileSystemObject");
Если FSO.FileExists(ИмяФайла) Тогда
  Сообщить("Файл существует!");
Иначе
  Сообщить("Файл не найден!");
КонецЕсли;
```

Копирование файлов и папок

Для копирования файлов и папок предназначены методы `CopyFile` и `CopyFolder` объекта `FileSystemObject`, а также методы `File.Copy` и `Folder.Copy`.

Рассмотрим простой пример копирования файла и папки (листинг 3.5).

Листинг 3.5. Копирование файлов и папок

```
FSO = Новый COMОбъект("Scripting.FileSystemObject");

// Копирование файла
FSO.CopyFile("c:\Program Files\1Cv77\ReadMe.txt", "c:\");
// Копирование папки
FSO.CopyFolder("c:\1c", "c:\2c");
```

Метод `CopyFile` имеет важную особенность — в качестве первого параметра можно задавать маску копируемых файлов. Например, чтобы скопировать все файлы с расширением doc, необходимо вызвать метод следующим образом: `CopyFile("c:\*.doc", "d:\")`.

Перемещение файлов и папок

Для перемещения файлов и папок предназначены методы `MoveFile` и `MoveFolder` объекта `FileSystemObject`, или методы `File.Move` и `Folder.Move`.

Рассмотрим пример перемещения файлов (листинг 3.6).

Листинг 3.6. Перемещение файлов

```
FSO = Новый COMОбъект("Scripting.FileSystemObject");

// Перемещение файлов
FSO.MoveFile("c:\Program Files\1cv8\*.htm", "c:\");
```

В приведенном коде перемещаются все файлы с расширением htm.

Удаление файлов и папок

Для удаления файлов и папок применяются методы `DeleteFile` и `DeleteFolder` объекта `FileSystemObject`, или методы `File.Delete` и `Folder.Delete`.

Рассмотрим пример удаления файлов (листинг 3.7).

Листинг 3.7. Удаление файлов

```
FSO = Новый COMОбъект("Scripting.FileSystemObject");

// Удаление файлов
FSO.DeleteFile("c:\*.htm");
```

Приведенный выше код осуществляет удаление всех файлов с расширением htm.

Создание папок

Для создания папок предназначены методы `FileSystemObject.CreateFolder` и `Folders.Add`. Перед созданием вложенной папки необходимо удостове-

риться, что существует папка верхнего уровня, иначе появится ошибка. То есть данные методы не предназначены для создания группы вложенных папок.

Ниже приведен пример последовательного создания группы вложенных папок (листинг 3.8).

Листинг 3.8. Создание папок

```
FSO = Новый COMОбъект("Scripting.FileSystemObject");

Корень = "c:\test\";
FSO.CreateFolder(Корень);

// Создаем группу папок
Для Ном1 = 1 По 5 Цикл
  FSO.CreateFolder(Корень + "folder" + Ном1);
  Для Ном2 = 1 По 5 Цикл
    FSO.CreateFolder(Корень + "folder" + Ном1 + "\subfolder" + Ном1 + "_"
+ Ном2);
  КонецЦикла;
КонецЦикла;
```

Работа с текстовыми файлами

С помощью WSH можно создавать текстовые файлы. Для этого предусмотрены методы `FileSystemObject.CreateTextFile` и `Folders.CreateTextFile`. Данные методы возвращают ссылку на объект `TextStream` и имеют три параметра:

- `Filename` — имя файла;
- `Overwrite` — признак замены существующего файла (параметр является не обязательным);
- `Unicode` — признак создания файла в формате Unicode, иначе создается в формате ANSI (параметр является не обязательным).

С помощью методов `WriteLine` и `WriteBlankLines` полученного объекта `TextStream` можно добавлять строки в файл.

Рассмотрим пример создания файла в формате ANSI и записи в него нескольких строк (листинг 3.9).

Листинг 3.9. Создание текстового файла

```
FSO = Новый COMОбъект("Scripting.FileSystemObject");

// Получаем объект TextStream
File = FSO.CreateTextFile("C:\Пример создания файла.txt", Истина, Ложь);
File.WriteLine("Дата создания: " + РабочаяДата);
File.WriteBlankLines(2); // Добавляем 2 пустые строки
File.WriteLine("Добро пожаловать на сайт автора: www.erpg.ru");
File.WriteLine("Copyright (C) Андрей Михайлов aka MAV 2004");
File.Close();
```

В WSH нельзя создавать файлы, отличные от текстовых.

Для открытия текстового файла, с целью чтения, записи или добавления новых записей, предназначены методы `FileSystemObject.OpenTextFile` и `File.OpenTextStream`. Данные методы имеют ряд параметров.

- `Filename` — имя файла.
- `Iomode` — режим открытия файла. Параметр может принимать следующие значения:
  - `1` — только для чтения;
  - `2` — для записи;
  - `8` — открытие файла и добавления строк в конец.
- `Create` — признак создания нового файла.
- `Format` — формат открываемого файла. Параметр может принимать следующие значения:
  - `-2` — используется формат, принятый по умолчанию в системе;
  - `-1` — Unicode;
  - `0` — ASCII.

Для чтения всего текста из файла используется метод `ReadAll()`. Для построчного чтения — `ReadLine()`. Рассмотрим пример открытия и чтения файла (листинг 3.10).

Листинг 3.10. Открытие и чтение текстового файла

```
ИмяФайла = "C:\Пример создания файла.txt";
FSO = Новый COMОбъект("Scripting.FileSystemObject");

// Проверяем, существует ли необходимый файл на диске
```

```
Если FSO.FileExists(ИмяФайла) Тогда
  File = FSO.OpenTextFile(ИмяФайла, 1, Ложь, 0);
  Сообщить(File.ReadAll());
Иначе
  Сообщить("Файл не найден!");
КонецЕсли;
```

Получение списка всех доступных дисков, папок и файлов

Для получения списка всех доступных дисков у объекта `FileSystemObject` существует коллекция `Drives`, содержащаяся в свойстве `FileSystemObject.Drives`. Для папок, в этом случае, имеется коллекция `Folders`, содержащаяся в свойстве `Folder.SubFolders`. Для файлов — коллекция `Files`, содержащаяся в свойстве `Folder.Files`.

Рассмотрим примеры вывода списка всех дисков (листинг 3.11) и списка всех файлов и папок (листинг 3.12).

Листинг 3.11. Получение списка всех доступных дисков

```
FSO = Новый COMОбъект("Scripting.FileSystemObject");

// Выборка объектов из коллекции Drives
Для каждого Диск Из FSO.Drives Цикл
  Стр = Диск.DriveLetter;
  Если Диск.DriveType = 3 Тогда
    Стр = Стр + " - " + Диск.ShareName;
  ИначеЕсли Диск.IsReady Тогда
    Стр = Стр + " - " + Диск.VolumeName;
  Иначе
    Стр = Стр + " - [Диск не найден]";
  КонецЕсли;
  Сообщить(Стр);
КонецЦикла;
```

Листинг 3.12. Получение списка всех доступных папок и файлов

```
FSO = Новый COMОбъект("Scripting.FileSystemObject");
Folder = FSO.GetFolder("C:\");

// Выборка всех папок
```

```
Для каждого Папка Из Folder.SubFolders Цикл
  Сообщить("[" + ВРег(Папка.Name) + "]");
КонецЦикла;

// Выборка всех файлов
Для каждого Файл Из Folder.Files Цикл
  Сообщить(НРег(Файл.Name));
КонецЦикла;
```

В листинге 3.12 приведен пример вывода списка только папок и файлов одного уровня, находящихся в корневом каталоге диска C:. Для построения дерева папок и файлов можно использовать данный пример, который должен находиться в функции, вызываемой рекурсивно.

Чтение свойств MP3-файлов

Как уже было показано ранее, с помощью объекта `FileSystemObject` можно обращаться к файлам.

Рассмотрим следующую задачу. Допустим необходимо прочитать теги (IDv1) MP3-файла. Тегами обычно называют свойства MP3-файла, включающими в себя информацию об авторе, названии, размере файла и т. п. Для решения данной задачи необходимо выполнить следующие действия:

1. Создать объект `FileSystemObject` для доступа к файлам.
2. Проверить наличие необходимого файла на диске.
3. Получить ссылку на объект `File` с помощью метода `GetFile`.
4. Прочитать размер файла с помощью метода `File.Size` и проверить, чтобы он был не менее 128 байт.
5. Открыть файл для чтения. Метод `File.OpenAsTextStream(1)` возвращает объект `Stream`.
6. Получить строку, содержащую последние 128 байт, в которых и хранятся теги MP3-файла.
7. Проверить наличие в начале полученной строки символов "TAG".
8. Выделить из строки нужные теги согласно спецификации MP3-файлов.

Рассмотрим пример чтения тегов MP3-файла (листинг 3.13).

Листинг 3.13. Чтение тегов MP3-файла

```
// Переменная ИмяФайла должна содержать строку, содержащую путь к MP3-файлу
Если (ИмяФайла = Неопределено) или СокрЛП(ИмяФайла = "") Тогда
  Сообщить("Не указано имя файла!");
```

```
  Возврат;
КонецЕсли;

// Создание объекта FileSystemObject
Попытка
  FSO = Новый COMОбъект("Scripting.FileSystemObject");
Исключение
  Сообщить("Ошибка создания объекта!");
  Возврат;
КонецПопытки;

// Проверка на наличие файла
Если НЕ FSO.FileExists(ИмяФайла) Тогда
  Сообщить("Файл не найден!");
  Возврат;
КонецЕсли;

// Получаем размер файла и проверяем, чтоб он был не менее
минимально допустимого
File = FSO.GetFile(ИмяФайла);
FileLength = File.Size;

Если FileLength <= 128 Тогда
  Сообщить("Файл малого размера!");
  Возврат;
КонецЕсли;

// Открываем файл для чтения (параметр = 1)
Попытка
  Stream = File.OpenAsTextStream(1);
Исключение
  Сообщить("Невозможно открыть файл!");
  Возврат;
КонецПопытки;

// Считываем строку с информацией (ID3v1)
Stream.Skip(FileLength - 128);
Попытка
  Data = Stream.Read(128);
```

```
Исключение
  Сообщить("Невозможно прочитать файл!");
  Возврат;
КонецПопытки;

// Закрываем поток и удаляем объекты
Stream.Close();
File = Неопределено;
FSO = Неопределено;

// Проверяем полученную строку на корректность
Если Лев(Data, 3) <> "TAG" Тогда
  Сообщить("Неверный формат файла!");
  Возврат;
КонецЕсли;

// Перечисляем все жанры, встречающиеся в спецификации к MP3-файлам
Жанры = Новый Массив();
Жанры.Добавить("Blues");
Жанры.Добавить("Classic Rock");
// и т.д...
Жанры.Добавить("JPop");
Жанры.Добавить("Synth Pop");

// Выделяем из строки нужные поля
Название = СокрЛП(Сред(Data, 4, 30));
Артист = СокрЛП(Сред(Data, 34, 30));
Альбом = СокрЛП(Сред(Data, 64, 30));
Год = СокрЛП(Сред(Data, 94, 4));
Комментарий = СокрЛП(Сред(Data, 98, 30));
Трек = КодСимвола(Сред(Data, 127, 1));
НомерЖанра = КодСимвола(Сред(Data, 128, 1));

Если НомерЖанра < Жанры.Количество() Тогда
  Жанр = Жанры[НомерЖанра];
Иначе
  Жанр = "";
КонецЕсли;
```

```
// Выводим теги
Сообщить("Название: " + Название);
Сообщить("Артист: " + Артист);
Сообщить("Альбом: " + Альбом);
Сообщить("Год: " + Год);
Сообщить("Комментарий: " + Комментарий);
Сообщить("Трек: " + Трек);
Сообщить("Жанр: " + Жанр);
```

Файлы MP3 хранятся в формате Unicode. Однако, к сожалению, версия 7.7 системы "1С:Предприятие" не позволяет работать с такими форматами. В связи с этим, напрямую в версии 7.7 теги MP3-файлов прочитать не удается. Для чтения тегов в данном случае нужно использовать внешние компоненты или объект `MSScriptControl.ScriptControl`, вынося весь код в VB-скрипт.

Специальные папки

Бывают ситуации, когда необходимо записать какой-либо файл в специальную папку (на рабочий стол, в избранное, включить в автозагрузку и т. п.). В английской версии Windows существует папка Startup (Автозагрузка), и если вы, например, укажите путь C:\WINDOWS\Главное меню\Программы\Автозагрузка, то эта запись сработает только в русской версии Windows и нигде больше. Чтобы избежать этих проблем, был создан список специальных папок. Чтобы получить доступ к какой-либо специальной папке, можно воспользоваться методом `SpecialFolders` объекта `WshShell`. Данный метод имеет один параметр — строку, описывающую имя специальной папки. Рассмотрим возможные параметры метода `SpecialFolders`:

- `AllUsersDesktop` — рабочий стол (для всех пользователей);
- `AllUsersStartMenu` — меню **Старт** (для всех пользователей);
- `AllUsersPrograms` — меню **Программы** (для всех пользователей);
- `AllUsersStartup` — **Автозагрузка** (для всех пользователей);
- `Desktop` — рабочий стол;
- `Favorites` — **Избранное**;
- `Fonts` — шрифты;
- `MyDocuments` — **Мои документы**;
- `NetHood` — папка NetHood;
- `PrintHood` — папка PrintHood;

- `Programs` — меню **Программы**;
- `Recent` — последние запуски;
- `SendTo` — меню **Отправить**;
- `StartMenu` — меню **Старт**;
- `Startup` — **Автозагрузка**;
- `Templates`— шаблоны.

Рассмотрим пример вывода некоторых специальных папок в системе "1С:Предприятие" версии 7.7 (листинг 3.14).

Листинг 3.14. Вывод специальных папок в версии 7.7

```
WSHShell = СоздатьОбъект("WScript.Shell");

Сообщить("Автозагрузка: " + WshShell.SpecialFolders("Startup"));
Сообщить("Программы: " + WshShell.SpecialFolders("Programs"));
Сообщить("Мои документы: " + WshShell.SpecialFolders("MyDocuments"));
Сообщить("Шрифты: " + WshShell.SpecialFolders("Fonts"));
Сообщить("Меню 'Старт': " + WshShell.SpecialFolders("StartMenu"));
```

В версии 7.7 все довольно просто и прозрачно. А вот в 8.0, по непонятным причинам, данный код вызывает ошибку. Чтобы обойти эту ошибку, можно использовать объект `MSScriptControl.ScriptControl`, в котором и получать путь, указывающий местоположение специальных папок, после чего возвращать их в контекст системы "1С:Предприятие". Рассмотрим пример получения специальных папок через VB-скрипт (листинг 3.15).

Листинг 3.15. Вывод специальных папок в версии 8.0

```
// Функция возвращает путь к системной папке, указанной в параметре <Имя>
Функция ПолучитьСистемнуюПапку(Имя)
  Попытка
    Ctrl = Новый COMОбъект("MSScriptControl.ScriptControl");
    // Указываем язык скрипта (VBS)
    Ctrl.Language = "vbscript";
    // Добавляем код на VBS
    Ctrl.AddCode("
    |Function SpecialFolders(Name)
    |Set Shell = CreateObject(""Wscript.Shell"")
    |SpecialFolders = Shell.SpecialFolders(Name)
```

```
    |End Function");
    // Запускаем функцию SpecialFolders с параметром <Имя>
    ИмяПапки = Ctrl.Run("SpecialFolders", Имя);

  Исключение
    // В случае неудачного выполнения возвращаем — Неопределено
    ИмяПапки = Неопределено;
  КонецПопытки;

  Возврат ИмяПапки;
КонецФункции

Сообщить("Рабочий стол: " + ПолучитьСистемнуюПапку("Desktop"));
Сообщить("Мои документы: " + ПолучитьСистемнуюПапку("MyDocuments"));
Сообщить("Избранное: " + ПолучитьСистемнуюПапку("Recent"));
```

Работа с ярлыками Windows

С помощью WSH можно создавать ярлыки для файлов и адресов сайтов в Интернете (URL). Для этого используется объект `WshShell`.

Создание ярлыков для файлов

Для создания ярлыка используется метод `CreateShortcut` объекта `WshShell`, который имеет единственный параметр — полное имя файла для ярлыка.

Приведем пример создания ярлыка для запуска системы SQL-версии "1С:Предприятие 7.7" (листинг 3.16).

Листинг 3.16. Создание ярлыка для файла

```
Shell = Новый COMОбъект("WScript.Shell");
DesktopPath = ПолучитьСистемнуюПапку("Desktop");
Link = Shell.CreateShortcut(DesktopPath + "\test.lnk");
Link.Arguments = "1 2 3";
Link.Description = "Тестовый пример";
Link.HotKey = "CTRL+ALT+SHIFT+X";
Link.IconLocation = "1cv7s.exe,1";
Link.TargetPath = "c:\Program Files\1Cv77\BIN\1cv7s.exe";
Link.WindowStyle = 3;
Link.WorkingDirectory = "c:\Program Files\1Cv77\BIN\";
Link.Save();
```

В данном примере метод `CreateShortcut` возвращает ссылку на объект `Link`. Для описания параметров ярлыка сначала заполняются основные свойства объект `Link`. После этого он сохраняется с помощью метода `Save()`.

Объект `Link` имеет следующие свойства:

- `Arguments` — описание параметров запуска файла;
- `Description` — текстовое описание к ярлыку;
- `HotKey` — комбинация клавиш для быстрого запуска ярлыка;
- `IconLocation` — файл с иконкой;
- `TargetPath` — полное имя запускаемого файла;
- `WindowStyle` — способ запуска программы, причем свойство может принимать следующие значения:
  - `3` — при запуске окно программы будет развернуто на весь экран;
  - `4` — устанавливает стандартный размер окна;
  - `7` — окно будет свернуто в значок на панели задач;
- `WorkingDirectory` — рабочая папка.

Копирование, перемещение и удаление ярлыков

Для копирования, перемещения и удаления ярлыков применяются такие же методы, что и для работы с файлами. А именно `CopyFile`, `MoveFile` и `DeleteFile` (листинг 3.17).

Листинг 3.17. Копирование, перемещение и удаление ярлыка

```
Shell = Новый COMОбъект("WScript.Shell");
FSO = Новый COMОбъект("Scripting.FileSystemObject");
DesktopPath = ПолучитьСистемнуюПапку("Desktop") + "\test.lnk";
NewPath = "C:\\";

// Копирование ярлыка
FSO.CopyFile(DesktopPath, NewPath);

// Перемещение ярлыка
MyDocumentsPath = ПолучитьСистемнуюПапку("MyDocuments") + "\test.lnk";
FSO.MoveFile(NewPath, MyDocumentsPath);

// Удаление ярлыка
FSO.DeleteFile(MyDocumentsPath);
```

Создание ярлыков для URL-адресов

Помимо ярлыков для файлов, WSH позволяет создавать ярлыки для запуска Web-страниц. В этом случае применяется все тот же метод `CreateShortcut`, но в качестве аргумента необходимо использовать файл с расширением URL и в свойстве `TargetPath` указывать адрес интернет-страницы.

Рассмотрим пример создания на рабочем столе ярлыка для запуска форума, посвященного системе "1С:Предприятие" (листинг 3.18).

Листинг 3.18. Создание ярлыка для URL-адреса

```
Shell = Новый COMОбъект("WScript.Shell");
DesktopPath = ПолучитьСистемнуюПапку("Desktop");

URL = Shell.CreateShortcut(DesktopPath + "\ERP Group.url");
URL.TargetPath = "http://www.erpg.ru";
URL.Save();

URL = Shell.CreateShortcut(DesktopPath + "\Ваш форум.url");
URL.TargetPath = "http://forum.erpg.ru";
URL.Save();
```

Работа с системным реестром Windows

Понятие реестра

Реестр — это база данных операционной системы, содержащая данные о текущей конфигурации программных и аппаратных средств вычислительной системы. Физически вся информация реестра разбита на два файла: SYSTEM.DAT и USER.DAT, находящиеся в каталоге Windows.

Чему действительно необходимо уделить внимание, так это структуре реестра. Он содержит шесть корневых разделов (ветвей), на которых ниже остановимся подробнее. Каждый из них включает подразделы, отображаемые в левой части окна в виде значка папки. Конечным элементом дерева реестра являются ключи или параметры, делящиеся на три типа:

- строковые (например, "C:\Windows");
- двоичные (например, 10 82 A0 8F), причем максимальная длина такого ключа 16 Кбайт;

- типа DWORD, которые занимают по 4 байта и отображаются в шестнадцатеричном и в десятичном виде (например, 0x00000020 (32), причем в скобках указано десятичное значение ключа).

Рассмотрим корневые разделы раздела, точнее, какая информация в них содержится.

- *HKEY_CLASSES_ROOT*. В этом разделе содержится информация о зарегистрированных в Windows типах файлов, что позволяет открывать их по двойному щелчку мыши, а также информация об OLE-серверах (подробнее о технологии OLE см. в *главе 4*).
- *HKEY_CURRENT_USER*. Здесь содержатся настройки оболочки пользователя (например, "Рабочего стола", меню **Пуск** и т. п.), вошедшего в среду Windows. Они дублируют содержимое подраздела HKEY_USER\name, где name — имя пользователя, вошедшего в среду Windows. Если на компьютере работает один пользователь и используется обычный вход в среду операционной системы, то значения раздела берутся из подраздела HKEY_USERS\.DEFAULT.
- *HKEY_LOCAL_MACHINE*. Этот раздел содержит информацию, относящуюся к компьютеру: драйверы, установленное программное обеспечение и его настройки.
- *HKEY_USERS*. Содержит настройки оболочки Windows для всех пользователей. Как было сказано выше, именно из этого раздела информация копируется в раздел HKEY_CURRENT_USER. Все изменения в HKCU (сокращенное название раздела HKEY_CURRENT_USER) автоматически переносятся в HKU.
- *HKEY_CURRENT_CONFIG*. В этом разделе содержится информация о конфигурации устройств Plug-and-Play и сведения о конфигурации компьютера с переменным составом аппаратных средств.
- *HKEY_DYN_DATA*. Здесь хранятся динамические данные о состоянии различных устройств, установленных на компьютере. Именно сведения этой ветви отображаются в окне **Система | Устройства | Свойства**, вызываемого из Панели управления. Данные этого раздела изменяются самой операционной системой, так что редактировать что-либо вручную не желательно.

Запись, чтение и удаление ветвей реестра

С помощью Windows Script Host возможно в "невидимом" режиме добавлять какую-либо информацию в реестр. Также возможно считывать информацию, создавать и удалять разделы и параметры. Пользователь может даже не подозревать, что на его компьютере выполняются какие-то работы. Очевидно, что многим администраторам такая возможность придется по душе.

Для работы с реестром в WSH предусмотрены методы RegWrite, RegRead и RegDelete объекта WshShell.

Метод RegWrite предназначен для создания ветвей реестра и записи в них параметров. Метод содержит три аргумента.

- Name — полное имя ветви или параметра. Если в конце имени стоит обратный слэш (\), то аргумент является ветвью, иначе — параметром;
- Value — значение параметра или значение по умолчанию в случае создания ветви;
- Type — тип значения (REG_SZ — строковый параметр, REG_EXPAND_SZ — расширяемый строковый параметр, REG_DWORD — числовой параметр, REG_BINARY — двоичный параметр).

Метод RegRead предназначен для чтения значения параметра и имеет один параметр — полное имя ветви или параметра реестра. В случае указания имени ветви (с обратным слэшем на конце) метод возвращает значения параметра по умолчанию.

Метод RegDelete предназначен для удаления ветви или параметра реестра и имеет один параметр — полное имя ветви или параметра.

Рассмотрим пример работы с системным реестром (листинг 3.19).

Листинг 3.19. Работа с системным реестром Windows

```
Shell = Новый COMОбъект("WScript.Shell");
Key =  "HKEY_CURRENT_USER\";

// Создание ветки HCU\TestKey\
Shell.RegWrite(Key + "TestKey\", "testkeydefault");
// Создание строкового параметра string1
Shell.RegWrite(Key + "TestKey\string1", "testkeystring");
// Создание параметра типа DWORD
Shell.RegWrite(Key + "TestKey\int", 123, "REG_DWORD");

// Чтение созданных параметров
Сообщить(Shell.RegRead(Key + "TestKey\"));
Сообщить(Shell.RegRead(Key + "TestKey\string1"));
Сообщить(Shell.RegRead(Key + "TestKey\int"));

// Удаление ветки HCU\TestKey\
Shell.RegDelete(Key + "TestKey\");
```

Работа с сетевыми ресурсами

Для работы с сетевыми ресурсами в WSH предназначен объект `WshNetwork`. Данный объект позволяет решать такие задачи, как подключения к сети, инсталляцию принтеров, назначение буквенных обозначений сетевым дискам, получение информации об имени пользователя или компьютера и так далее.

Рассмотрим основные свойства объекта `WshNetwork`:

- `ComputerName` — возвращает имя компьютера;
- `UserDomain` — возвращает имя домена, в котором зарегистрирован пользователь;
- `UserName` — возвращает имя пользователя.

Рассмотрим основные методы объекта `WshNetwork`:

- `MapNetworkDrive` — присваивает назначенную пользователем букву сетевому диску;
- `EnumNetworkDrives` — возвращает список подключенных сетевых дисков;
- `RemoveNetworkDrive` — отключает сетевой диск;
- `AddWindowsPrinterConnection` — позволяет установить новый принтер в Windows, т. е. указать путь к принтеру и инициировать установку драйверов;
- `AddPrinterConnection` — подключает сетевой принтер;
- `RemovePrinterConnection` — отключает сетевой принтер;
- `SetDefaultPrinter` — задает принтер по умолчанию;
- `EnumPrinterConnections` — возвращает список подключенных принтеров.

Получение информации о сетевой идентификации

Для начала разберемся, как работают методы `ComputerName`, `UserDomain` и `UserName` объекта `WshNetwork`.

Рассмотрим небольшой пример, в котором выводятся сведения об имени пользователя, имени компьютера и домене (листинг 3.20).

Листинг 3.20. Вывод информации об имени пользователя, имени компьютера и домене

```
Network = Новый COMОбъект("WScript.Network");

Сообщить("Имя пользователя: " + Network.UserName);
Сообщить("Имя компьютера: " + Network.ComputerName);
Сообщить("Имя домена: " + Network.UserDomain);
```

В данном примере использовались свойства объекта WshNetwork: UserName, ComputerName и UserDomain. С их помощью была получена информация о подключившемся пользователе. Эта информация может понадобиться, например, если на компьютере работает несколько пользователей и им нужен доступ к разным сетевым дискам. Таким образом, можно обеспечить автоматическое подключение этих дисков в зависимости от того, какой пользователь сейчас работает.

Работа с сетевыми дисками

Перед тем как подключать какой-либо сетевой диск, необходимо сначала определить — подключен ли уже нужный диск или нет. Рассмотрим пример вывода информации о том, какие сетевые диски в данный момент времени уже подключены (листинг 3.21).

Листинг 3.21. Вывод списка подключенных сетевых дисков

```
Network = Новый COMОбъект("WScript.Network");
// Получаем коллекцию сетевых дисков
Drives = Network.EnumNetworkDrives();
Для каждого Диск Из Drives Цикл
  Сообщить(Диск);
КонецЦикла;
```

Для подключения и отключения сетевых дисков существуют методы MapNetworkDrive и RemoveNetworkDrive соответственно. Рассмотрим пример подключения сетевого диска (листинг 3.22).

Листинг 3.22. Подключение сетевого диска

```
Net = Новый COMОбъект("WScript.Network");
Попытка
  Net.MapNetworkDrive("M:", "\\Server\DATA\CRM");
Исключение
  Сообщить("Невозможно произвести подключение!");
КонецПопытки;
```

В данном примере в методе MapNetworkDrive используются только первые два параметра: буква, на которую будет подключаться диск, и, собственно, путь к сетевому диску. Третий параметр (Истина или Ложь) указывает, осуществлять или нет подключение этого диска при следующем входе пользо-

вателя. Четвертый и пятый параметры определяют имя пользователя и его пароль, с которыми может подключаться диск. Лучше организовать доступ к диску нужным пользователям на сервере средствами самой ОС. Тогда посторонние люди не смогут узнать пароль для работы с этим ресурсом.

При отключении сетевого диска используется метод `RemoveNetworkDrive`, который имеет три параметра. Обязателен только первый параметр — буква отключаемого диска. Второй параметр — отключает диск (если `Истина`) даже если он в этот момент используется. Третий параметр сохраняет настройки для следующих входов пользователя.

Работа с сетевыми принтерами

В `WshNetwork` реализовано несколько методов сетевого доступа к принтерам. Чтобы явно установить связь принтера с портом, следует использовать метод `AddPrinterConnection`. Для соединения с принтерами в среде Windows (и автоматической установки драйверов на машинах с операционными системами Windows 2000/XP/NT и Server 2003) нужно применить метод `AddWindowsPrinterConnection`, который представлен в листингах 3.23 и 3.24. Метод `SetDefaultPrinter` назначает компьютеру принтер, связь с которым устанавливается по умолчанию.

Листинг 3.23. Подключение сетевого принтера в Windows NT/2000/XP/Server 2003

```
Network = Новый COMОбъект("WScript.Network");
Попытка
  Network.AddWindowsPrinterConnection("\\ServerName\PrinterName");
  Network.SetDefaultPrinter("\\ServerName\PrinterName");
Исключение
  Сообщить("Невозможно произвести подключение!");
КонецПопытки;
```

Листинг 3.24. Подключение сетевого принтера в Windows 9x/ME

```
Network = Новый COMОбъект("WScript.Network");
Попытка
  Network.AddWindowsPrinterConnection("\\ServerName\
PrinterName", " Lexmark Optra S 1650");
  Network.SetDefaultPrinter("\\ServerName\PrinterName");
Исключение
  Сообщить("Невозможно произвести подключение!");
КонецПопытки;
```

Данные примеры подключают сетевой принтер с именем PrinterName, находящийся на сервере ServerName и устанавливают его в качестве принтера, используемого по умолчанию.

Для отключения сетевого принтера используется метод RemovePrinterConnection, который имеет три параметра:

- первый параметр — буква отключаемого диска;
- второй параметр — (если Истина) отключает принтер, даже если он в этот момент времени используется;
- третий параметр — признак сохранения настройки для следующих входов пользователя.

Обязательным является только первый параметр.

Как и в случае с сетевыми дисками, объект WshNetwork позволяет определить список подключенных сетевых принтеров. Для этого используется метод EnumPrinterConnections, который возвращает коллекцию принтеров. Данную коллекцию, как и любую другую, в системе "1С:Предприятие" 8.0 можно перебрать с помощью конструкции Для...каждого из...Цикл...КонецЦикла. В версии 7.7 системы "1С:Предприятие" — с помощью объекта COMServices.

Управление программами

Под управлением программ будем понимать: запуск программ; активизацию их окон; имитацию нажатия, в границах этих окон, различных клавиш клавиатуры. Для решения перечисленных задач нужны всего три функции.

- SendKeys(String) — имитируется нажатие клавиши или последовательности клавиш на клавиатуре, указанных в единственном параметре.
- AppActivate(Title) — активизирует приложение по заголовку окна или по идентификатору процесса.
- Run(Command, WindowStyle, WaitOnReturn) — запускает приложение по командной строке. Параметры функции имеют следующее назначение:
  - первый параметр (Command) — определяет запускаемое приложение;
  - второй параметр (WindowStyle) — определяет стиль окна и может принимать значения:
    - ◊ 0 — запускать программу в скрытом окне;

Примечание

Если вы попытаетесь запустить приложение с таким значением второй переменной, то окно программы будет невидимым и кнопка на панели задач не будет создана. Удостовериться, что приложение действительно запустилось, можно в диспетчере задач, который вызывается по комбинации клавиш <Ctrl>+<Alt>+<Del>.

- ◊ 1 — запустить программу в оригинальном размере и положении окна;
- ◊ 2 — запустить программу минимизированной;
- ◊ 3 — запустить программу максимизированной (на полный экран);
- ◊ 4 — запустить программу с последним использованным размером и положением окна на экране;
- ◊ 5 — запустить программу с текущими размером и позицией окна на экране;
- ◊ 6 — запустить программу минимизированной (при этом активным будет следующее окно в последовательности окон активных приложений Windows);

- третий параметр (значения — Истина или Ложь) указывает на необходимость дождаться завершения выполнения приложения.

В качестве параметра метода SendKeys можно указывать как алфавитно-цифровые символы, так и символы специальных клавиш, например, <Enter>, <Tab>, <F1>...<F12>, <Alt>, <Shift>, <Ctrl> и т. п. В табл. 3.1 приведены специальные обозначения для этих клавиш.

Таблица 3.1. *Обозначения специальных клавиш*

Клавиша	Обозначение
<BACKSPACE>	{BACKSPACE}, {BS}, или {BKSP}
<BREAK>	{BREAK}
<CAPS LOCK>	{CAPSLOCK}
<DEL> или <DELETE>	{DELETE} или {DEL}
<DOWN ARROW>	{DOWN}
<END>	{END}
<ENTER>	{ENTER} или ~
<ESC>	{ESC}
<HELP>	{HELP}
<HOME>	{HOME}
<INS или INSERT>	{INSERT} или {INS}
<LEFT ARROW>	{LEFT}
<NUM LOCK>	{NUMLOCK}
<PAGE DOWN>	{PGDN}
<PAGE UP>	{PGUP}

Таблица 3.1 (окончание)

Клавиша	Обозначение
<PRINT SCREEN>	{PRTSC}
<RIGHT ARROW>	{RIGHT}
<SCROLL LOCK>	{SCROLLLOCK}
<TAB>	{TAB}
<UP ARROW>	{UP}
<F1>	{F1}
<F2>	{F2}
<F3>	{F3}
<F4>	{F4}
<F5>	{F5}
<F6>	{F6}
<F7>	{F7}
<F8 >	{F8}
<F9>	{F9}
<F10>	{F10}
<F11>	{F11}
<F12>	{F12}

Для указания клавиш <Alt>, <Shift>, <Ctrl> существуют специальные коды:

- <Shift> — "+";
- <Ctrl> — "^";
- <Alt> — "%".

Например, если вы хотите передать нажатие нескольких клавиш с нажатой клавишей <Shift>, надо выполнять команду `WshShell.SendKeys("+(ABC)")`, если только одной клавиши (первую, указанную в последовательности клавиш) — команду `WshShell.SendKeys("+ABC")`.

Примечание

Необходимо заметить, что такие символы, как "+", "^" и "%", зарезервированы под обозначение специальных клавиш, так что просто так передать эти (и некоторые другие) символы не удастся. Для этого надо заключить их в фигурные скобки, например, "{+}", "{%}" и т. д. Вот полный список символов, которые необходимо заключать в фигурные скобки: +, ^, %, ~, (,), {, }, [,].

Если вы хотите передать несколько нажатий подряд одной клавиши, то вовсе не обязательно все их набирать в скрипте. Можно просто указать количество повторений. Так, строка `WshShell.SendKeys("{A 100}")` эквивалентна сотне нажатий клавиши "A". Правда, есть ограничение на количество возможных повторений нажатия клавиш. Это значение лежит где-то между 8150 и 8180 раз. При превышении этого значения будет выдана ошибка.

Управление калькулятором

В листинге 3.25 приведен код, демонстрирующий, как запустить калькулятор и рассчитать значение произведения чисел 2 и 3.

Листинг 3.25. Управление калькулятором

```
WshShell = Новый COMОбъект("WScript.Shell");

// Запускаем калькулятор
WshShell.Run("calc");

// Ждем пока калькулятор загрузится (для русской версии ОС)
Пока Не WshShell.AppActivate("Калькулятор") Цикл
КонецЦикла;

// Эмулируем нажатие клавиш
WshShell.SendKeys("1{+}");
WshShell.SendKeys("2");
WshShell.SendKeys("~");
WshShell.SendKeys("*3");
WshShell.SendKeys("~");
```

Приведенный пример запускает калькулятор, ждет пока он загрузится и после активации его окна эмулирует последовательное нажатие клавиш. В результате выполнения в окне калькулятора появится цифра 9.

Управление Microsoft Word

Рассмотрим еще один интересный пример, в котором запускается Microsoft Word и в нем печатается некоторый текст (листинг 3.26).

Листинг 3.26. Управление Microsoft Word

```
WshShell = Новый COMОбъект("WScript.Shell");

Если WshShell.AppActivate("Word") Тогда
  WshShell.SendKeys("Welcome to http://www.erpg.ru");
  WshShell.SendKeys("~"); // Enter
  WshShell.SendKeys("%"); // Alt
  // Спускаемся на 4 пункта вниз
  WshShell.SendKeys("{DOWN}{DOWN}{DOWN}{DOWN}");
  WshShell.SendKeys("~"); // Enter
  WshShell.SendKeys("c:\test.doc");
  WshShell.SendKeys("~"); // Enter
Иначе
  Сообщить("Microsoft Word не запущен!");
КонецЕсли;
```

Реализация примера подразумевает, что у вас уже запущен MS Word с пустой страницей. В результате выполнения в документе напечатается текст и будет сохранен под именем c:\test.doc, но только в том случае, если пункт меню **Сохранить** находится четвертым сверху.

Запуск встроенного калькулятора

В системе "1С:Предприятие" (в версиях 7.7 и 8.0) существует встроенный калькулятор, который вызывается при нажатии комбинации клавиш <Ctrl>+<F2> или через пункт меню **Сервис | Калькулятор**. С помощью метода `SendKeys` объекта `WshShell` можно программно вызвать встроенный калькулятор (листинг 3.27).

Листинг 3.27. Запуск встроенного калькулятора

```
WshShell = Новый COMОбъект("WScript.Shell");
WshShell.SendKeys("^{F2}"); // Ctrl+F2
```

Закрытие окна сообщений в системе "1С:Предприятие"

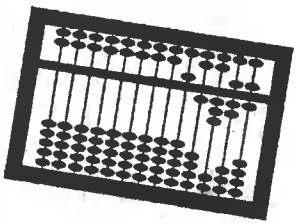
В системе "1С:Предприятие" (в версиях 7.7 и 8.0) при вызове оператора `Сообщить` появляется окно сообщений. Часто бывает ситуация, когда необ-

ходимо программно закрыть это окно. Оно закрывается при нажатии комбинации клавиш <Ctrl>+<Shift>+<Z>. Приведем пример закрытия окна сообщений (листинг 3.28).

Листинг 3.28. Закрытие окна сообщений

```
// С помощью вывода текста показываем окно сообщений
Для Ном = 1 По 100 Цикл
        Сообщить("Текст " + Ном);
КонецЦикла;

WshShell = Новый COMОбъект("WScript.Shell");
// Закрываем окно сообщений
WshShell.SendKeys("^+(z)"); // <Ctrl>+<Shift>+<Z>
WshShell.SendKeys("^+(я)"); // для русской раскладки
```



Глава 4

Использование OLE Automation

Понятие OLE Automation

OLE — это аббревиатура названия технологии Object Linking and Embedding (связывание и внедрение объектов). Она указывает на способность работать с составными объектами, созданными в других приложениях (например, рисунками, документами и т. п.). Основные термины, с которыми оперирует данная технология, — это OLE-объект, сервер приложения и контейнер приложения.

OLE-объектом называют объект, созданный в другом приложении и сохранивший связь с этим приложением. Точечный рисунок, созданный в редакторе Paint, электронные таблицы в формате Excel или диаграмма из MS Graph — все они могут быть OLE-объектами, если будут вставлены в документ соответствующим образом. Если не вставлять их как OLE-объект, то связь с оригинальным приложением отсутствует.

Контейнером приложения OLE называют приложение, в котором создается составной документ, содержащий OLE-объект, позволяя обрабатывать его в исходном приложении (например, таком как Paint или Excel), которое использовалось для создания этого объекта.

Сервером приложения OLE (OLE Server Application) называют приложение, создающее объекты, которые можно поместить в документ-контейнер. Программы — "1С:Предприятие", Microsoft Word и Excel являются приложениями, которые могут выступать и как OLE-сервер, и как OLE-контейнер. Другими словами, эти приложения могут создавать новые OLE-объекты, а также хранить OLE-объекты, созданные в других приложениях.

Первоначально механизм OLE был задуман как технология интеграции программных продуктов, входящих в комплект Microsoft Office, в другие приложения. Первое воплощение OLE (OLE 1) представляло собой механизм создания и работы с составными документами (compound documents).

С точки зрения пользователя, составной документ выглядит единым набором информации, но фактически содержит элементы, созданные двумя или несколькими разными приложениями. С помощью OLE 1 пользователь мог, например, объединить электронную таблицу, созданную Microsoft Excel, с текстовым документом Microsoft Word. Идея состояла в том, чтобы документо-ориентированная (document-centric) модель работы с компьютером позволила бы пользователю больше думать об информации и меньше о приложениях, ее обрабатывающих. Как следует из слов "связывание и внедрение", составные документы можно создать, либо связав их, либо полностью внедрив один документ в другой.

OLE 1, как и большинство первых версий программных продуктов, была несовершенна. Архитекторам следующей версии предстояло улучшить первоначальный проект. Вскоре они поняли, что составные документы это лишь частный случай более общей проблемы — как разные программные компоненты должны предоставлять друг другу сервисы? Для решения этой проблемы архитекторы OLE создали группу технологий, область применения которых гораздо шире составных документов. Основу *OLE 2* составляет важнейшая из этих технологий — "Модель многокомпонентных объектов" (Component Object Model или COM). Новая версия OLE не только обеспечивает поддержку составных документов лучше, чем первая, но и несомненно идет куда дальше простого объединения документов, созданных в разных приложениях. OLE 2 позволяет по-новому взглянуть на взаимодействие любых типов программ.

Электронные таблицы, текстовые процессоры и другие программы предоставляют множество полезных возможностей. Почему бы не обеспечить доступ к ним и другому программному обеспечению? Чтобы это стало возможным, приложения должны предоставлять свои сервисы не только человеку, но и программам, т. е. они должны быть программируемыми. Обеспечение программируемости и является целью "Автоматизации" (*Automation*, первоначально называлась OLE-автоматизацией).

Приложение можно сделать программируемым, обеспечив доступ к его сервисам, через обычный COM-интерфейс. Однако так поступают редко. Вместо этого доступ к сервисам приложений осуществляется через диспинтерфейсы (dispinterface). Они очень похожи на интерфейсы (имеют методы, клиенты осуществляют к ним доступ через указатели интерфейса и т. д.). Однако диспинтерфейсы имеют и существенные отличия. В частности, их методы гораздо проще вызывать клиентам, написанным на простых языках типа встроенного языка системы "1С:Предприятие".

Использование системы "1С:Предприятие" в качестве OLE Automation сервера

Методы работы с OLE-объектами существенно отличаются в разных версиях системы "1С:Предприятие". Поэтому их рассмотрение будет происходить по отдельности, для каждой версии системы.

Использование OLE Automation сервера в версии 7.7

Для запуска системы "1С:Предприятие", в качестве *OLE Automation сервера*, из внешнего приложения, выполняется следующая последовательность действий:

1. Создается объект с OLE идентификатором:
   - `V1CEnterprise.Application` — версия независимый ключ;
   - `V77.Application` — версия зависимый ключ;
   - `V77S.Application` — версия зависимый ключ, SQL-версия;
   - `V77L.Application` — версия зависимый ключ, локальная версия;
   - `V77M.Application` — версия зависимый ключ, сетевая версия.
2. Выполняется инициализация системы "1С:Предприятие" методом `Initialize()`.
3. Вызываются атрибуты и методы системы "1С:Предприятие", как OLE Automation сервера.

"1С:Предприятие", в качестве OLE Automation сервера, имеет 4 метода:

1. `Initialize()` — выполняет инициализацию системы "1С:Предприятие".
2. `CreateObject()` — создает объект агрегатного типа данных системы "1С:Предприятие" и возвращает ссылку на него.
3. `EvalExpr()` — вычисляет выражение системы.
4. `ExecuteBatch()` — выполняет последовательность операторов.

Подключение к базе данных

Для открытия и инициализации базы данных системы "1С:Предприятие" (версии 7.7) предназначен метод `Initialize()`, который имеет следующие параметры:

- `Имя объекта.RMTrade` — имя переменной и ключевое слово `RMTrade`;
- `КомандняяСтрока` — командная строка, в которой можно прописать путь к базе данных, имя пользователя и пароль;
- `ПоказыватьЗаставку` — либо пустая строка, либо — `"NO_SPLASH_SHOW"`, чтобы не показывать заставку при загрузке.

Для того чтобы открыть базу через OLE, необходимо создать объект с идентификатором OLE, выполнить инициализацию базы и проверить успешность ее выполнения. Если в методе `Initialize` указать только первый параметр, то при открытии базы открывается окно выбора, в котором предлагается выбрать пользователя и ввести пароль (листинг 4.1).

Листинг 4.1. Подключение к базе данных

```
БазаOLE = СоздатьОбъект("V77.Application");
Открыта = БазаOLE.Initialize(БазаOLE.RMTrade,,);
Если Открыта = 0 Тогда
  // True=-1, False=0
  Сообщить("Не удалось открыть базу данных!");
  Возврат;
КонецЕсли;
```

Доступ к объектам базы данных

Для доступа к объектам агрегатных типов предназначен метод `CreateObject()`, который имеет единственный параметр — строковое выражение, значение которого содержит имя агрегатного типа данных, заданного в конфигураторе. Например: `"Справочник.Номенклатура"`, `"Документ.ПриходнаяНакладная"` и т. п.

Этот метод создает объект агрегатного типа данных системы "1С:Предприятие" и возвращает ссылку на него.

Для доступа к константам, перечислениям и видам субконто достаточно использовать свойства глобального контекста `Константа`, `Перечисление` или `ВидыСубконто` сервера приложения (листинг 4.2).

Листинг 4.2. Доступ к константам, перечислениям и видам субконто

```
Сообщить(БазаOLE.Константа.ДатаЗапретаРедактирования);
Сообщить(БазаOLE.Перечисление.КатегорияКонтрагента.Поставщик);
Сообщить(БазаOLE.ВидСубконто.Контрагенты);
```

Для сравнения перечислений или видов субконто разных баз данных (OLE и местной) необходимо сначала получить строковое или числовое представление данного перечисления или вида субконто с помощью методов `Идентификатор` или `ЗначениеПоНомеру` в местной базе, а затем найти соответствующее ему значение в OLE-базе (листинг 4.3).

Листинг 4.3. Поиск вида субконто в OLE-базе

```
ВыбранныйВидСубконтоИд = ВыбранныйВидСубконто.Идентификатор();
ВыбранныйВидСубконтоOLE = БазаOLE.ВидСубконто.ЗначениеПоИдентификатору
(ВыбранныйВидСубконтоИд);
```

В данном примере переменная `ВыбранныйВидСубконто` содержит значение типа `"ВидСубконто"`, которое определено вне данного кода, например, на форме.

Для доступа к справочникам и документам необходимо использовать метод `CreateObject`. После создания объекта справочника или документа к нему применимы все методы, касающиеся соответствующего объекта в среде "1С:Предприятие" (листинг 4.4).

Листинг 4.4. Доступ к справочникам

```
СправочникOLE = БазаOLE.CreateObject("Справочник.Фирмы");
СправочникOLE.ВыбратьЭлементы();
Пока СправочникOLE.ПолучитьЭлемент() Цикл
  Сообщить(СправочникOLE.Наименование);
КонецЦикла;
```

В данном примере в операторе `Сообщить` для вывода наименования элемента справочника необходимо использовать именно конструкцию `СправочникOLE.Наименование`, а не `СправочникOLE.ТекущийЭлемент()`. Иначе вместо строкового или числового представления этого элемента программа выдаст просто `"OLE"`. По аналогии также не будет работать метод `ТекущийДокумент()`. Тем не менее через OLE-объект можно использовать методы: `Новый()`, `Записать()` и др.

Доступ к документам производится так же, как и к справочникам (листинг 4.5).

Листинг 4.5. Доступ к документам

```
// Поиск контрагента в OLE-базе
СправочникOLE = БазаOLE.CreateObject("Справочник.Контрагенты");
СправочникOLE.НайтиПоКоду(ВыбКонтрагент.Код, 0);

// Создание нового документа
ДокументOLE = БазаOLE.CreateObject("Документ.Контакт");
ДокументOLE.Новый();
// Заполнение реквизита Контрагент из OLE-базы
ДокументOLE.Контрагент = СправочникOLE.ТекущийЭлемент();
ДокументOLE.Тема = "Пример";
ДокументOLE.Записать();

Если ДокументOLE.Провести()=0 Тогда
  Сообщить("Ошибка! Документ не проведен.");
КонецЕсли;
```

Важно отметить тот факт, что нельзя сравнивать или присваивать два или несколько элементов справочника или документов, находящихся в разных базах. Для поиска соответствующего элемента или документа в OLE-базе необходимо использовать методы: НайтиПоКоду, НайтиПоНаименованию и т. п. В листинге 4.5 для заполнения контрагента в документе Контакт используется элемент базы OLE, предварительно найденный по коду, соответствующему коду контрагента в местной базе. Это же касается и объектов типа "Счет".

Для доступа к регистрам используется тот же метод CreateObject (листинг 4.6).

Листинг 4.6. Доступ к регистрам

```
РегистрOLE = БазаOLE.CreateObject("Регистр.Касса");
РегистрOLE.ВыбратьИтоги();

Пока РегистрOLE.ПолучитьИтог()=1 Цикл
  Сообщить("Остаток в кассе " + РегистрOLE.Касса.Наименование " = " +
РегистрOLE.Сумма);
КонецЦикла;
```

Вычисление выражений

Для вычисления выражений, написанных на встроенном языке системы "1С:Предприятие", предназначен метод EvalExpr, который возвращает результат вычисления. Результатом выражения может быть число, строка, дата или значение любого агрегатного типа данных. Результат с неопределенным типом данных преобразуется к строковому типу. Так же с помощью метода EvalExpr можно выполнять экспортируемые функции глобального модуля (листинг 4.7).

Листинг 4.7. Вычисление выражений

```
// Выполнение функции глобального модуля
ФорматированноеИмяПользователя = БазаOLE.EvalExpr
("глДополнитьСтрокуП (" + ИмяПользователя + ", "" "", 255)");
СуммаПрописью = БазаOLE.EvalExpr("глСуммаПрописью(" + Сумма + "," +
Валюта + ")");
// Вычисление выражения используя стандартные функции
СуммаПрописью = БазаOLE.EvalExpr("Цел(Лог(2.2)) * 5");
```

Во второй строке данного примера в функцию глСуммаПрописью, в качестве параметра Валюта, необходимо передавать строку, т. к. данная функция

будет исполняться внутри подключаемого OLE-сервера, т. е. внешней подключаемой базы данных.

Выполнение операторов

Для выполнения последовательности операторов в OLE-базе используется метод `ExecuteBatch`, единственным параметром которого является строка, описывающая текст программы на встроенном языке системы "1С:Предприятие". Метод `ExecuteBatch` возвращает значение логического типа (`true` — если последовательность операторов выполнена успешно; `false` — если нет). В OLE Automation значения `true` и `false` представляются соответственно значениями `-1` (минус единица) и `0`.

Ниже приведен пример выполнения процедур в OLE-базе (листинг 4.8).

Листинг 4.8. Выполнение операторов

```
СправочникOLE = БазаOLE.EvalExpr("СоздатьСправочникФирмы()");
СправочникOLE.ВыбратьЭлементы();
Пока СправочникOLE.ПолучитьЭлемент() Цикл
  Сообщить(СправочникOLE.Наименование);
КонецЦикла;
// Выполнение процедур
БазаOLE.ExecuteBatch("ПроцедураРегл1();ПроцедураРегл2()");
```

Использование OLE Automation сервера в версии 8.0

Система "1С:Предприятие 8.0" может использоваться внешними приложениями в качестве OLE Automation сервера.

Обычно в этих целях "1С:Предприятие" используют для управления конфигурациями системы программ "1С:Предприятие" из других приложений и выполнения действий аналогичным интерактивным действиям пользователя (например, построение отчетов).

Для запуска системы "1С:Предприятие" в качестве OLE Automation сервера из внешнего приложения выполняется следующая последовательность действий:

1. Создается OLE объект с идентификатором `"V8.Application"`.
2. Выполняется инициализация системы "1С:Предприятие" методом `Connect`.
3. Вызываются свойства и методы системы "1С:Предприятие" как OLE Automation сервера.

Приведем пример доступа к данным информационной базы посредством OLE Automation сервера системы "1С:Предприятие 8.0" из Visual Basic скрипта (листинг 4.9).

Листинг 4.9. Использование OLE Automation сервера системы "1С:Предприятие" 8.0

```
' Подключение к БД средствами OLE Automation
Set cc = CreateObject("v8.Application")
cc.Connect("File=C:\Progra~1\1cv8\Db\Trade; Usr=Иванов")
' Установка видимости приложения
cc.Visible = True

' Задаем текст запроса
Text = "ВЫБРАТЬ * ИЗ Справочник.Контрагенты"
' Создаем объект Запрос
Set Query = cc.NewObject("Запрос")
Query.Text = Text
' Выполнение запроса
Set Result = Query.Execute()
Set Choose = Result.Choose()
' Выборка результата выполнения запроса
Do While Choose.Next()
  ' Пример вывода сообщения в OLE-базе
  cc.Message(Choose.Description)
Loop

' Пример вывода предупреждения в OLE-базе
cc.DoMessageBox "Форум для вопросов http://forum.erpg.ru", 10,
"Внимание!!!"
```

Данный пример последовательно выполняет все три перечисленные действия.

Некоторые версии внешних программ, в том числе и Visual Basic Script, обращающихся к программе "1С:Предприятие" посредством OLE Automation, могут неправильно интерпретировать русские идентификаторы объектов.

Примечание

Для обращения к свойствам и методам агрегатных типов данных системы "1С:Предприятие" из внешних приложений рекомендуется использовать их англоязычные синонимы.

При работе в качестве OLE Automation сервера "1С:Предприятие" предоставляет доступ ко всем свойствам и методам своего глобального контекста, а также позволяет включать и выключать пользовательский интерфейс (главное окно приложения). Поэтому объект OLE-сервер системы "1С:Предприятие" в качестве своих свойств может использовать: системные перечисления, значения констант, перечислений, справочников, документов и т. п., а также переменные, объявленные в модуле приложения с ключевым словом `Экспорт`.

Кроме того, OLE Automation сервер имеет одно дополнительное свойство и два метода с целью выполнения действий, специфичных для работы в режиме OLE Automation.

Единственное свойство `Visible` имеет значение логического типа и позволяет управлять видимостью пользовательского интерфейса системы "1С:Предприятие". По умолчанию свойство имеет значение `Ложь`.

Объект OLE-сервер системы "1С:Предприятие" в качестве своих методов может использовать системные процедуры и функции, а также процедуры и функции модуля приложения и общих модулей, объявленные с ключевым словом `Экспорт`. Кроме того, OLE-сервер системы "1С:Предприятие" имеет два дополнительных метода: `Connect` и `NewObject`.

Метод `Connect` выполняет инициализацию системы "1С:Предприятие" и имеет единственный параметр — строку, используемую для соединения с информационной базой. Метод возвращает `Истина`, если инициализация прошла успешно, или `Ложь`, если нет.

Существует два варианта подключения к OLE-серверу системы "1С:Предприятие", которые зависят от варианта использования информационной базы (файловый или клиент-серверный варианты). Приведем пример подключения файловой и клиент-серверной версии системы "1С:Предприятие" (листинг 4.10).

Листинг 4.10. Варианты подключения к OLE-серверу

```
' Подключение к БД файлового варианта
Set FileDb = CreateObject("v8.Application")
FileDb.Connect("File=C:\Progra~1\1cv8\Db\Trade; Usr=Иванов; Pwd=super")

' Подключение к БД клиент-серверного варианта
Set ServerDb = CreateObject("v8.Application")
ServerDb.Connect("Srvr=Server1C; Ref=Trade; Usr=Иванов; Pwd=super")
```

Рассмотрим параметры подключения к базе данных в файловом варианте.

- `File` — указывает на каталог базы данных.

- `Usr` — имя пользователя, под которым будет происходить авторизация.
- `Pwd` — пароль пользователя.

Для подключения к базе данных в клиент-серверном варианте используются четыре параметра.

- `Srvr` — указывает имя сервера, на котором работает сервер приложений системы "1С:Предприятие";
- `Ref` — указывает имя базы данных на сервере приложений;
- `Usr` — определяет имя пользователя, под которым будет происходить авторизация;
- `Pwd` — пароль пользователя.

Второй специальный метод OLE-сервера системы "1С:Предприятие" — `NewObject`, который создает объект агрегатного типа данных системы "1С:Предприятие" и возвращает ссылку на него. Единственный параметр метода — строковое выражение, значение которого содержит имя агрегатного типа данных, объявленного в конфигураторе.

В листинге 4.9 используется метод `NewObject` для создания объекта типа `Запрос`.

Поскольку система "1С:Предприятие" может создавать и использовать OLE Automation серверы, то из системы "1С:Предприятие" можно обращаться к другим информационным базам.

В последующем материале книги будут показаны примеры использования OLE Automation сервера системы "1С:Предприятие 8.0" из среды "1С:Предприятия".

Доступ к объектам базы данных

Как уже было сказано ранее, для создания объектов агрегатного типа используется метод `NewObject`, но для доступа к константам и перечислениям достаточно использовать свойства глобального контекста `Константы` и `Перечисления` сервера приложения (листинг 4.11).

Листинг 4.11. Доступ к константам и перечислениям

```
База = Новый СОМОбъект("V8.Application");
База.Connect(СтрокаПодключения);

// Доступ к константам
Сообщить(База.Константы.ОсновнаяВалюта.Получит().Наименование);

// Доступ к перечислениям
Сообщить(База.Перечисления.Периодичность.День);
```

Для доступа к справочникам и документам можно использовать метод NewObject. Однако можно использовать и свойства глобального контекста Справочники (листинг 4.12) и Документы (листинг 4.13). После создания объекта справочника или документа к нему применимы все методы, касающиеся соответствующего объекта в среде "1С:Предприятие".

Листинг 4.12. Доступ к справочникам

```
База = Новый COMОбъект("V8.Application");
База.Connect(СтрокаПодключения);

// Доступ к справочникам
СправочникOLE = База.Справочники.Контрагенты;
Выборка = СправочникOLE.Выбрать();
// Обход выборки элементов справочника
Пока Выборка.Следующий() Цикл
  Если Не Выборка.ЭтоГруппа Тогда
    Сообщить(Выборка.Наименование);
  КонецЕсли;
КонецЦикла;

// Создание нового элемента справочника
СправочникOLE = База.Справочники.Контрагенты;
НовыйЭлемент = СправочникOLE.СоздатьЭлемент();
НовыйЭлемент.Наименование = "Михайлов Андрей";
НовыйЭлемент.Комментарий = "Создан автоматически.";
НовыйЭлемент.Записать();
```

Доступ к документам производится так же, как и к справочникам.

Листинг 4.13. Доступ к документам

```
База = Новый COMОбъект("V8.Application");
База.Connect(СтрокаПодключения);

// Поиск контрагента в OLE-базе
СправочникOLE = База.Справочники.Контрагенты;
СпрЭлемент = СправочникOLE.НайтиПоНаименованию("Михайлов Андрей");

// Создание нового документа
```

```
ДокументOLE = База.Документы.ПоступлениеТоваров.СоздатьДокумент();
ДокументOLE.Дата = ТекущаяДата();
ДокументOLE.Контрагент = СпрЭлемент;
ДокументOLE.УстановитьНовыйНомер();
ДокументOLE.Записать();
```

Так же, как и в версии 7.7 системы "1С:Предприятие", в объектах OLE-базы данных можно использовать только ее же объекты. В примере, рассмотренном выше (листинг 4.13), для заполнения реквизита документа `Контрагент` используется элемент справочника `Контрагенты` той же OLE-базы. Это правило касается не только объектов агрегатных типов (справочников, документов, регистров и т. п.), но и универсальных коллекций значений, таких как массивы, структуры, списки значений и т. п.

Рассмотрим пример вывода курса доллара из регистра сведений (листинг 4.14).

Листинг 4.14. Доступ к регистрам сведений

```
База = Новый COMОбъект("V8.Application");
База.Connect(СтрокаПодключения);
// Доступ к регистрам сведений

// Поиск элемента справочника с наименованием "USD"
СправочникOLE = База.Справочники.Валюты;
СпрЭлемент = СправочникOLE.НайтиПоНаименованию("USD");

// Создаем структуру в OLE-базе
Структура = База.NewObject("Структура");
Структура.Вставить("Валюта", СпрЭлемент);

// Получаем курс доллара на текущую дату
РегистрСведенийOLE = База.РегистрыСведений.КурсыВалют;
Структура = РегистрСведенийOLE.ПолучитьПоследнее(ТекущаяДата(), Структура);
Сообщить("Курс доллара = " + Структура.Курс);
Сообщить("Кратность = " + Структура.Кратность);
```

В этом примере для использования фильтра применяется структура, которая создается с помощью метода `NewObject`.

Примечание

Важно иметь в виду, что если создать структуру в исходной конфигурации с помощью конструктора `Новый Структура()`, то произойдет ошибка.

Как следует из кода, приведенного в листинге 4.14, работа с регистрами полностью аналогична работе со справочниками и документами.

Использование запросов

В системе "1С:Предприятие" использование запросов является самым основным способом получения сводных данных, поэтому и при использовании системы в качестве OLE Automation сервера очень важно уметь использовать механизмы запросов.

Для выполнения запроса к данным OLE-базы достаточно с помощью метода `NewObject` создать объект `Запрос` и далее работать с ним стандартными методами.

Рассмотрим простой пример вывода списка наименований контрагентов (листинг 4.15).

Листинг 4.15. Использование запросов

```
База = Новый СОМОбъект("V8.Application");
База.Connect(СтрокаПодключения);

Текст = "ВЫБРАТЬ * ИЗ Справочник.Контрагенты";
// Создаем объект запрос
Запрос = База.NewObject("Запрос");
Запрос.Текст = Текст;
// Выполняем запрос
РезультатЗапроса = Запрос.Выполнить();
Выборка = РезультатЗапроса.Выбрать();
// Обход результата выполнения запроса
Пока Выборка.Следующий() Цикл
  Сообщить(Выборка.Наименование);
КонецЦикла;
```

Вызов пользовательских интерфейсов

В предыдущем примере показана возможность выборки данных с помощью запроса и вывода результатов запроса в основную базу. Эти же данные можно визуализировать и в базе данных, используемой в качестве OLE Automation сервера.

Для начала необходимо показать основное окно приложения с помощью свойства `Visible`, а затем можно вызывать стандартные операторы — `Сообщить`, `Предупреждение`, а также открывать формы объектов и формировать отчеты (листинг 4.16).

Листинг 4.16. Вызов пользовательских интерфейсов

```
База = Новый COMОбъект("V8.Application");
// Шаг 1. Соединение с базой данных
База.Connect(СтрокаПодключения);

// Шаг 2. Установка видимости главного окна приложения
База.Visible = Истина;

// Шаг 3. Пример вывода сообщения в OLE-базе
База.Сообщить("Пример сообщения через OLE Automation");
// Пример вывода предупреждения в OLE-базе
База.Предупреждение("Форум для вопросов
http://forum.erpg.ru",,"Внимание!!!");

// Шаг 4. Открытие формы списка справочника
Форма = База.Справочники.Контрагенты.ПолучитьФорму("ФормаСписка");
Форма.Открыть();

// Шаг 5. Создание и формирование отчета
НовыйОтчет = База.Отчеты.Монитор.Создать();
НовыйОтчетФорма = НовыйОтчет.ПолучитьФорму();
ВыбраннаяНастройка = База.ВосстановитьЗначение("Монитор_настройка");

Если ВыбраннаяНастройка<>Неопределено Тогда
  НовыйОтчетФорма.НачальноеЗначениеВыбора = ВыбраннаяНастройка;
КонецЕсли;

// Шаг 6. Открытие формы сформированного отчета
НовыйОтчетФорма.Открыть();
```

В приведенном примере последовательно выполняются следующие действия:

1. Соединение с базой данных.
2. Установка видимости главного окна приложения.
3. Вывод сообщений с помощью операторов `Сообщить` и `Предупреждение`.
4. Открытие формы списка справочника `Контрагенты`.
5. Создание и формирование отчета `Монитор основных показателей`.
6. Открытие формы сформированного отчета.

Данный пример полностью работоспособен на конфигурации "1С:Управление торговлей" (редакция 10).

Доступ к метаданным

Для доступа к метаданным можно использовать свойство глобального контекста Метаданные. Приведем пример вывода всех справочников и их реквизитов для конфигурации, используемой в качестве OLE Automation сервера (листинг 4.17).

Листинг 4.17. Доступ к метаданным

```
База = Новый COMОбъект("V8.Application");
База.Connect(СтрокаПодключения);

МетаСправочники = База.Метаданные.Справочники;
// Перебираем все справочники
Для каждого Справочник Из МетаСправочники Цикл
  Сообщить(Справочник.Имя);
  // Перебираем все реквизиты
  Для каждого Реквизит Из Справочник.Реквизиты Цикл
    Сообщить(Символы.Таб + Реквизит.Имя);
  КонецЦикла;
КонецЦикла;
```

Использование COM-соединения в версии 8.0

Основная задача использования COM-соединения для конфигурации системы "1С:Предприятие 8.0" — это обеспечение надежного и быстрого программного доступа к объектам конфигурации из внешних приложений.

COM-сервер системы "1С:Предприятие 8.0" реализован в виде библиотеки COMCntr.dll, которая выполняется в рамках исполняющего процесса.

Применение COM-соединения во многом похоже на использование OLE Automation сервера, но существуют и очень важные отличия. Приведем некоторые общесистемные отличия COM-соединения от OLE Automation.

При использовании COM-соединения:

- затрачиваются гораздо меньше системных ресурсов;
- происходит более быстрая установка соединения с базой данных;
- происходит более быстрое обращение к свойствам и методам объектов;
- полностью отсутствуют пользовательские интерфейсы.

Отличия в программном использовании COM-соединения можно определить следующим образом:

- OLE-объект создается с идентификатором "V8.COMConnector";
- отсутствует свойство Visible;
- не работают все методы системы "1С:Предприятие", касающиеся визуализации объектов;
- недоступен модуль приложения, но при этом доступен модуль внешнего соединения;
- из общих модулей доступны только те, у которых установлено свойство Внешнее соединение;
- метод Connect возвращает объект соединения с базой данных (в OLE Automation метод возвращал значение логического типа, а доступ к базе данных происходил через объект, созданный с помощью идентификатора "V8.Application").

Приведем пример подключения к базе данных системы "1С:Предприятие 8.0" с помощью COM-соединения на языке Visual Basic Script (листинг 4.18).

Листинг 4.18. Доступ к базе данных с помощью COM-соединения

```
' Подключение к БД
Set cc = CreateObject("v8.COMConnector")
Set con = cc.Connect("File=C:\Progra~1\1cv8\Db\Trade; Usr=Иванов")

' Вывод сообщения о названии конфигурации
MsgBox con.Metadata.Name, 0, "Конфигурация"

' Задаем текст запроса
Text = "ВЫБРАТЬ * ИЗ Справочник.Контрагенты"

' Создаем объект Запрос
Set Query = con.NewObject("Запрос")
Query.Text = Text

' Выполнение запроса
Set Result = Query.Execute()
Set Choose = Result.Choose()
ResultChoose = ""

' Выборка результата выполнения запроса
```

```
Do While Choose.Next()
  ' Конкатенация всех значений выборки в одну строку
  ResultChoose = ResultChoose & vbCrLf & Choose.Description
Loop

' Вывод результата выполнения запроса
MsgBox ResultChoose, 0, "Результат выборки"
```

Достаточно сравнить данный пример подключения к базе данных с примером, приведенным в листинге 4.9, чтобы понять, что при использовании OLE Automation доступ к объектам базы данных происходит через переменную `cc`, а при COM-соединении — через переменную `con`.

Во всем остальном использовать COM-соединение можно так же, как и OLE Automation. Чтобы удостовериться в этом, рассмотрим пример доступа к справочникам (листинг 4.19), аналогичный примеру, приведенному в листинге 4.12.

Листинг 4.19. Доступ к справочникам

```
v8 = Новый COMОбъект("V8.COMConnector");
База = v8.Connect(СтрокаПодключения);

// Доступ к справочникам
СправочникOLE = База.Справочники.Контрагенты;
Выборка = СправочникOLE.Выбрать();
// Обход выборки элементов справочника
Пока Выборка.Следующий() Цикл
  Если Не Выборка.ЭтоГруппа Тогда
    Сообщить(Выборка.Наименование);
  КонецЕсли;
КонецЦикла;

// Создание нового элемента справочника
СправочникOLE = База.Справочники.Контрагенты;
НовыйЭлемент = СправочникOLE.СоздатьЭлемент();
НовыйЭлемент.Наименование = "Михайлов Андрей";
НовыйЭлемент.Комментарий = "Создан автоматически.";
НовыйЭлемент.Записать();
```

Данный пример отличается от приведенного в листинге 4.12 только способом создания объекта и методом подключения к базе данных.

Работа с Microsoft Word

Как уже было сказано ранее, Microsoft Word является OLE-сервером, который можно использовать для автоматизации множества задач.

Для начала рассмотрим, каковы программные идентификаторы основных объектов Microsoft Word и что представляет собой его объектная модель.

Для OLE-сервера Microsoft Word доступны непосредственно следующие объекты:

- `Application` — с помощью этого программного идентификатора создается экземпляр Word без открытых документов;
- `Document` — с помощью этого программного идентификатора создается экземпляр Word с одним вновь созданным документом.

Все остальные объекты Word являются так называемыми внутренними объектами. Это означает, что они не могут быть созданы "сами по себе". Так, объект `Paragraph` (абзац) не может быть создан отдельно от содержащего его документа.

Если вспомнить, что основное назначение Word — работа с документами, легко понять иерархию его объектной модели. Основным объектом в ней, как и во всех других приложениях Microsoft Office, является объект `Application`, содержащий коллекцию `Documents` объектов типа `Document`. Каждый объект типа `Document` содержит коллекцию `Paragraphs` объектов типа `Paragraph`, `Bookmarks` типа `Bookmark`, `Characters` типа `Character` и т. д. Манипуляция документами, абзацами, символами, закладками реально осуществляется путем обращения к свойствам и методам этих объектов.

Рассмотрим основные методы работы с объектом Microsoft Word, которые будут использоваться практически во всех примерах (листинг 4.20).

Листинг 4.20. Основные методы работы с Microsoft Word

```
// Создание объекта MS Word
Word = Новый COMОбъект("Word.Application");
// Установка видимости приложения
Word.Visible = Истина;
// Добавление нового документа в приложение
Word.Documents.Add();
// Получение активного документа
Документ = Word.ActiveDocument();
// Добавить параграф
Документ.Paragraphs.Add();
```

```
// Добавить текст
Документ.Paragraphs(НомерПараграфа).Range().InsertAfter(Текст);
// Стилевое оформление
Документ.Paragraphs(НомерПараграфа).Range().Style="Заголовок 2";
// Открытие сужествующего документа
Word.Documents.Open(ИмяФайла);
// Сохранение документа
Документ.Save();
Документ.SaveAs(ИмяФайла);
// Закрытие документа
Документ.Close();
// Закрытие приложения
Word.Quit();
```

Создание и открытие документов MS Word

Для создания примеров использования Microsoft Word можно применять код создания контроллера, приведенный в листинге 4.20. Начнем сразу с создания и открытия документов.

Создать новый документ Word можно, используя метод `Add` коллекции `Documents` объекта `Application` — `Word.Documents.Add()`. Для создания документа с помощью шаблона нужно указать путь к шаблону в качестве параметра метода `Add`: `Word.Documents.Add("C:\Templates\NewStyle.dot")`.

Для открытия уже существующего документа следует воспользоваться методом `Open` коллекции `Documents`: `Word.Documents.Open("C:\MyWordFile.doc")`.

Отметим, что свойство `ActiveDocument` объекта `Word.Application` указывает на текущий активный документ среди одного или нескольких открытых. Помимо этого к документу можно обращаться по его порядковому номеру с помощью метода `Item`; например, ко второму открытому документу можно обратиться так: `Word.Documents.Item(2)`. Необходимо отметить, что нумерация членов коллекций в Microsoft Office начинается с единицы.

Сделать документ активным можно с помощью метода `Activate`: `Word.Documents.Item(1).Activate()`.

Следующее, чему следует научиться, — это сохранять документ Word и закрывать сам Word.

Сохранение, печать и закрытие документов

Закрытие документа может быть осуществлено с помощью метода `Close`: `Word.Documents.Item(2).Close()` или `Word.ActiveDocument.Close()`.

Метод `Close` имеет несколько необязательных параметров, влияющих на правила сохранения документа. Первый из них влияет на то, сохраняются ли внесенные в документ изменения, и может принимать три значения:

- `0` — не сохранять изменения;
- `-1` — сохранять изменения;
- `-2` — вывести диалоговое окно с соответствующим вопросом.

Второй параметр влияет на формат сохраняемого документа:

- `0` — сохранить в формате Word;
- `1` — сохранить в исходном формате документа;
- `2` — вывести диалоговое окно **Сохранить как**.

Третий параметр принимает значения `Истина` или `Ложь` и влияет на то, пересылать ли документ следующему пользователю по электронной почте. Если эта функция не используется, этот параметр можно проигнорировать.

Таким образом, при использовании указанных выше параметров закрыть документ можно, например, так — `Word.ActiveDocument.Close(4294967295, 2)`. Просто сохранить документ, не закрывая его, можно с помощью метода `Save` — `Word.ActiveDocument.Save()`.

Этот метод также имеет несколько необязательных (в случае позднего связывания) параметров, первый из которых равен `Истина`, если документ сохраняется автоматически, и `Ложь`, если нужно выводить диалоговое окно для получения подтверждения пользователя о сохранении изменений (если таковые были сделаны). Второй параметр влияет на формат сохраняемого документа, и список его возможных значений совпадает со списком значений второго параметра метода `Close`.

Закрыть приложение Microsoft Word можно с помощью метода `Quit` объекта `Word.Application`. Этот метод, в общем случае, имеет параметры, совпадающие с параметрами метода `Close` объекта `Document`.

Вывод документа на устройство печати можно осуществить с помощью метода `PrintOut` объекта `Document`, например: `Word.ActiveDocument.PrintOut()`.

Если нужно изменить параметры печати, следует указать значения соответствующих параметров метода `PrintOut`.

Примечание

Для Microsoft Word их около двадцати параметров печати. Описание их можно найти на сайте Microsoft.

Вставка текста и объектов в документ и форматирование текста

Для создания абзацев в документе можно использовать коллекцию `Paragraphs` объекта `Document`, представляющую набор абзацев данного документа. Добавить новый абзац можно с помощью метода `Add` этой коллекции: `Word.ActiveDocument.Paragraphs.Add()`.

Для вставки собственно текста в документ, тем не менее, применяется не объект `Paragraph`, а объект `Range`, представляющий любую непрерывную часть документа (в том числе и вновь созданный абзац). Этот объект может быть создан разными способами. Например, можно указать начальный и конечный символы диапазона (если таковые имеются в документе), указать номер абзаца, или указать несколько абзацев, следующих подряд (листинг 4.21).

Листинг 4.21. Вставка текста в документ Microsoft Word

```
// 1-й способ (указать начальный и конечный символы диапазона)
Rng = Word.ActiveDocument.Range(2,4); //со 2-го по 4-й символы
// 2-й способ (указать номер абзаца, например, только что созданного):
Rng = Word.ActiveDocument.Paragraphs.Item(1).Range;
// 3-й способ (указать несколько абзацев, следующих подряд)
Rng = Word.ActiveDocument.Range( Word.ActiveDocument.Paragraphs.Item(3).
Range.Start,  Word.ActiveDocument.Paragraphs.Item(5).Range.End);
// Вставка текста
Rng.InsertAfter("Это вставляемый текст");
```

Вставить текст можно с помощью методов `InsertBefore` (перед диапазоном) или `InsertAfter` (после диапазона) объекта `Range`.

Помимо объекта `Range` текст можно вставлять с помощью `Selection`, являющимся свойством объекта `Word.Application` и представляющим собой выделенную часть документа (этот объект создается, если пользователь выделяет часть документа с помощью мыши, и может быть также создан с помощью приложения-контроллера). Сам объект `Selection` можно создать, применив метод `Select` к объекту `Range`, например — `Word.ActiveDocument.Paragraphs.Item(3).Range.Select()`. В данном примере в текущем документе выделяется третий абзац.

Если необходимо вставить строку текста в документ вместо выделенного фрагмента текста, либо перед ним, это можно сделать с помощью фрагмента кода, который приведен в листинге 4.22.

Листинг 4.22. Замена выделенного текста

```
// Признак замены выделенного фрагмента при вставке текста
App.Options.ReplaceSelection = Истина;
// Получить выделенный фрагмент
Sel = App.Selection;
// Вставить текст
Sel.TypeText("Это текст, которым мы заменим выделенный фрагмент");
// Конец абзаца
Sel.TypeParagraph();
```

Отметим, что если свойство `Options.ReplaceSelection` объекта `Word.Application` имеет значение `Истина`, выделенный текст будет заменен на новый (этот режим действует по умолчанию). Если же нужно, чтобы текст был вставлен перед выделенным фрагментом, а не вместо него, следует установить значение этого свойства равным `Ложь`.

Символ конца абзаца при использовании объекта `Selection` может быть вставлен с помощью метода `TypeParagraph`.

К объекту `Selection`, так же как и к объекту `Range`, можно применить методы `InsertBefore` и `InsertAfter`. В этом случае, в отличие от предыдущего, вставляемый текст станет частью выделенного фрагмента текста.

С помощью объекта `Selection`, используя его свойство `Font`, а также и свойства объекта `Font` (`Bold`, `Italic`, `Size` и т. п.), можно отформатировать текст. Например, таким образом можно вставить строку, выделенную жирным шрифтом (листинг 4.23).

Листинг 4.23. Форматирование текста

```
// Установка полужирного шрифта
Sel.Font.Bold = Истина;
Sel.TypeText("Это текст, который мы выделим  жирным шрифтом.");
// Отмена полужирного шрифта
Sel.Font.Bold = Ложь;
// Признак конца параграфа
Sel.TypeParagraph();
// Установка стиля текста
Sel.Style = "Заголовок 1";
Sel.TypeText("Это текст, который станет заголовком");
Sel.TypeParagraph();
```

Для наложения на вставляемый текст определенного заранее стиля можно использовать свойство `Style` этого же объекта (листинг 4.24).

Нередко документы Word содержат данные других приложений.

Простейший способ вставить такие данные в документ — использование метода `Paste` объекта `Range`. Естественно, в этом случае в буфере обмена уже должны содержаться вставляемые данные. Если нужно поместить в буфер обмена часть документа Word, это можно сделать с помощью метода `Copy` объекта `Range`. Пример копирования и вставки данных приведен в листинге 4.24.

Листинг 4.24. Копирование и вставка данных в документ

```
// Копирование данных
Rng = Word.Selection.Range;
Rng.Copy();
// Вставка данных
Rng = Word.Selection.Range;
Rng.Collapse(0);
Rng.Paste();
```

Параметр метода `Collapse` указывает на то, где, в начале или в конце исходного фрагмента, окажется новый объект `Range` или `Selection`:

- `1` — новый объект находится в начале фрагмента;
- `0` — новый объект находится в конце фрагмента.

Перемещение курсора по тексту

Перемещать курсор по тексту можно с помощью метода `Move` объектов `Range` и `Selection`. Этот метод имеет два параметра. Первый указывает на то, в каких единицах измеряется перемещение — в символах (по умолчанию) или словах, предложениях, абзацах и др. Второй параметр указывает, на сколько единиц при этом нужно переместиться (это число может быть и отрицательным; по умолчанию оно равно `1`). Примеры таких перемещений курсора приведены в листинге 4.25.

Листинг 4.25. Перемещение курсора по тексту

```
// Перемещение курсора на один символ вперед
Rng.Move();
// Перемещение курсора на три абзаца вперед
Rng.Move(4,3);
```

```
// Перемещение курсора на одно слово вперед
Rng.Move(2);
// Перемещение курсора на десять слов назад
Rng.Move(2, -10);
```

Второй параметр метода `Move` может принимать следующие значения:

- `1` — символ;
- `2` — слово;
- `3` — предложение;
- `4` — абзац;
- `6` — часть документа (например, колонтитул, оглавление и др.);
- `8` — раздел;
- `9` — колонка таблицы;
- `10` — строка таблицы;
- `12` — ячейка таблицы;
- `15` — таблица.

Нередко для перемещения по тексту используются закладки. Создать закладку в текущей позиции курсора можно путем добавления члена коллекции `Bookmarks` объекта `Document` с помощью метода `Add`. При этом надо указать имя закладки в качестве его параметра.

Проверить существование закладки в документе можно с помощью метода `Exists`, а переместиться на нее — с помощью метода `Goto` объектов `Document`, `Range` или `Selection`. Пример работы с закладками приведен в листинге 4.26.

Листинг 4.26. Работа с закладками

```
// Добавить новую закладку
Word.ActiveDocument.Bookmarks.Add("MyBookmark");
// Переместить курсор на закладку
Rng = Word.ActiveDocument.Goto(-1, 2, ,"MyBookmark");
// Вставить текст после закладки
Rng.InsertAfter("Текст, вставленный после закладки");
```

В методе `Goto` данного примера первый параметр указывает на необходимость перейти к указанной закладке. Второй — определяет, что нужно искать следующий объект в тексте.

Отметим, что с помощью метода Goto можно перемещаться не только на указанную закладку, но и на другие объекты (рисунки, грамматические ошибки и др.). Направление перемещения тоже может быть различным. Поэтому список констант, которые могут быть использованы в качестве параметров данного метода, довольно велик.

Создание таблиц

Создавать таблицы можно двумя способами. Первый заключается в вызове метода Add коллекции Tables объекта Document и последовательном заполнении ячеек данными. Этот способ при позднем связывании работает довольно медленно.

Второй способ — более "быстрый". Сущность его заключается в создании текста из нескольких строк, содержащих подстроки с разделителями и последующей конвертации такого текста в таблицу с помощью метода ConvertToTable объекта Range.

Примечание

В качестве разделителя можно использовать любой или почти любой символ. Нужно только, чтобы он заведомо не встречался в данных, которые будут помещены в будущую таблицу.

В листинге 4.27 приведен пример создания этим способом таблицы, состоящей из трех строк и трех столбцов (в качестве разделителя, являющегося первым параметром метода ConvertToTable, используется запятая).

Листинг 4.27. Создание таблицы

```
Rng = Word.Selection.Range;
Rng.Collapse(0);
Rng.InsertAfter("1, 2, 3");
Rng.InsertParagraphAfter();
Rng.InsertAfter("4,5,6");
Rng.InsertParagraphAfter();
Rng.InsertAfter("7,8,9");
Rng.InsertParagraphAfter();
Rng.ConvertToTable(",");
```

Отметим, что внешний вид таблицы можно изменить с помощью свойства Format, а также с помощью свойств коллекции Columns, представляющей колонки таблицы, и коллекции Rows, представляющей строки таблицы объекта Table.

Обращение к свойствам документа

Свойства документа можно получить с помощью коллекции `BuiltInDocumentProperties` объекта `Document`, пример работы с которой приведен в листинге 4.28.

Листинг 4.28. Обращение к свойствам документа

```
// Получаем коллекцию BuiltInDocumentProperties
Prop = Word.ActiveDocument.BuiltInDocumentProperties;
// Выводим некоторые свойства
Сообщить("Название: " + Prop[1].Value);
Сообщить("Автор: " + Prop[3].Value);
Сообщить("Шаблон: " + Prop[6].Value);
```

Ниже перечислены все возможные индексы коллекции `BuiltInDocumentProperties`:

- 1 — название;
- 2 — назначение;
- 3 — автор;
- 4 — ключевые слова;
- 5 — комментарии;
- 6 — шаблон;
- 7 — автор, редактировавший текст последним;
- 8 — версия;
- 9 — имя приложения;
- 10 — когда последний раз документ был выведен на устройство печати;
- 11 — время создания;
- 12 — время, когда документ был сохранен в последний раз
- 13 — суммарное время редактирования;
- 14 — число страниц;
- 15 — число слов;
- 16 — число символов;
- 17 — правила доступа к документу;
- 18 — категория;
- 19 — формат документа;

- 20 — менеджер;
- 21 — компания;
- 22 — число байт;
- 23 — число строк;
- 24 — число абзацев;
- 25 — число слайдов;
- 26 — число комментариев;
- 27 — число скрытых слайдов;
- 28 — число мультимедиаклипов;
- 29 — путь к гипертекстовым ссылкам;
- 30 — число символов без учета пробелов.

Итак, сейчас были рассмотрены основные операции, которые наиболее часто применяются при автоматизации Microsoft Word. В дальнейшем будут рассмотрены примеры практического применения Microsoft Word в качестве OLE-сервера.

Выгрузка метаданных в MS Word

С помощью объекта `Word.Application`, предоставляемого OLE-сервером MS Word, можно программно создавать текстовые документы, включающие в себя форматирование текстов, таблицы, рисунки и другие объекты, доступные пользователям данного редактора.

Приведем пример формирования отчета о конфигурации, который будет сформирован в MS Word (листинг 4.29).

Листинг 4.29. Выгрузка метаданных в MS Word для версии 8.0

```
Попытка
  Word = Новый COMОбъект("Word.Application");
Исключение
  Сообщить("Не удалось открыть Word");
  Возврат;
КонецПопытки;
Сообщить("Ждите... Это может занять продолжительное время...");

// Добавим новый документ
Word.Documents.Add();
Документ = Word.ActiveDocument();
```

```
// Определяем все возможные объекты метаданных
ВсеТипы = Новый Структура();
ВсеТипы.Вставить("ОбщиеМодули", "Общие модули");
ВсеТипы.Вставить("ОбщиеФормы", "Общие формы");
ВсеТипы.Вставить("ОбщиеМакеты", "Общие макеты");
ВсеТипы.Вставить("ОбщиеКартинки", "Общие картинки");
ВсеТипы.Вставить("Роли", "Роли");
ВсеТипы.Вставить("Интерфейсы", "Интерфейсы");
ВсеТипы.Вставить("Стили", "Стили");
ВсеТипы.Вставить("Языки", "Языки");
ВсеТипы.Вставить("ПланыОбмена", "Планы обмена");
ВсеТипы.Вставить("Константы", "Константы");
ВсеТипы.Вставить("КритерииОтбора", "Критерии отбора");
ВсеТипы.Вставить("Справочники", "Справочники");
ВсеТипы.Вставить("Документы", "Документы");
ВсеТипы.Вставить("ЖурналыДокументов", "Журналы документов");
ВсеТипы.Вставить("Перечисления", "Перечисления");
ВсеТипы.Вставить("Отчеты", "Отчеты");
ВсеТипы.Вставить("Обработки", "Обработки");
ВсеТипы.Вставить("ПланыВидовХарактеристик", "Планы видов характеристик");
ВсеТипы.Вставить("ПланыСчетов", "Планы счетов");
ВсеТипы.Вставить("ПланыВидовРасчета", "Планы видов расчета");
ВсеТипы.Вставить("РегистрыСведений", "Регистры сведений");
ВсеТипы.Вставить("РегистрыБухгалтерии", "Регистры бухгалтерии");
ВсеТипы.Вставить("РегистрыРасчета", "Регистры расчета");

// Определяем все возможные параметры
ВсеПараметры = Новый Структура();
ВсеПараметры.Вставить("Измерения", "Измерения");
ВсеПараметры.Вставить("Ресурсы", "Ресурсы");
ВсеПараметры.Вставить("Реквизиты", "Реквизиты");
ВсеПараметры.Вставить("Формы", "Формы");
ВсеПараметры.Вставить("Макеты", "Макеты");
ВсеПараметры.Вставить("Графы", "Графы");
ВсеПараметры.Вставить("ТабличныеЧасти", "Табличные части");
ВсеПараметры.Вставить("Значения", "Значения");

// Перебор типов метаданных
```

```
Для Каждого ТипОбъекта Из ВсеТипы Цикл
  Состояние(ТипОбъекта.Значение);

  Если Метаданные[ТипОбъекта.Ключ].Количество() > 0 Тогда
    // Добавляем новый параграф
    Документ.Paragraphs.Add();
    Номер = Документ.Paragraphs.Count();
    Документ.Paragraphs(Номер-1).Range(). InsertAfter( ТипОбъекта.Значение);
    Документ.Paragraphs(Номер).Range().Style="Заголовок 1";

    // Перебираем конкретные объекты
    Для каждого Объект Из Метаданные[ТипОбъекта.Ключ] Цикл
      // Добавляем новый параграф
      Документ.Paragraphs.Add();
      Номер = Документ.Paragraphs.Count();
      Документ.Paragraphs(Номер-1).Range().InsertAfter(Объект.Синоним);
      Документ.Paragraphs(Номер).Range().Style="Заголовок 2";

      // Перебираем параметры метаданных
      Для каждого Параметр Из ВсеПараметры Цикл
        Попытка
          Количество = Объект[Параметр.Ключ].Количество();
        Исключение
          Продолжить;
        КонецПопытки;

        Если Количество > 0 Тогда
          // Добавляем новый параграф
          Документ.Paragraphs.Add();
          Номер = Документ.Paragraphs.Count();
          Документ.Paragraphs(Номер-1).Range().InsertAfter
          (Параметр.Значение + " (" + Количество + ")");
          Документ.Paragraphs(Номер).Range().Style="Заголовок 3";

          // Добавляем новый параграф
          Документ.Paragraphs.Add();
          Номер=Документ.Paragraphs.Count();
          // Добавляем новую таблицу
```

```
Документ.Tables.Add(Документ. Paragraphs(Номер).Range(),
Количество+1,4);

Счетчик = 1;
// Заполняем шапку таблицы
Таблица = Документ.Tables(Документ.Tables.Count());
Таблица.Cell(Счетчик,1).Range().InsertAfter("Имя");
Таблица.Cell(Счетчик,2).Range().InsertAfter("Синоним");
Таблица.Cell(Счетчик,3).Range().InsertAfter("Тип");
Таблица.Cell(Счетчик,4).Range().InsertAfter("Комментарий");
// Устанавливаем жирный шрифт для шапки таблицы
Документ.Range( Таблица.Cell( Счетчик,1). Range.Start,
Таблица.Cell(Счетчик,4).Range.End).Font.Bold = Истина;

Для каждого Значение Из Объект[Параметр.Ключ] Цикл
  Счетчик = Счетчик + 1;
  // Заполняем содержимое таблицы
  Попытка
    Таблица.Cell(Счетчик,1).Range().InsertAfter(Значение.Имя);
  Исключение
  КонецПопытки;
  Попытка
    Таблица.Cell( Счетчик,2)
    .Range().InsertAfter( Значение.Синоним);
  Исключение
  КонецПопытки;
  Попытка
    СтрокаТипов = "";
    // Выводим все типы
    Для каждого Тип Из Значение.Тип.Типы() Цикл
      СтрокаТипов = СтрокаТипов + " " + Строка(Тип);
    КонецЦикла;
    Таблица.Cell(Счетчик,3).Range().InsertAfter(СтрокаТипов);
  Исключение
  КонецПопытки;
  Попытка
    Таблица.Cell( Счетчик,4).Range().
    InsertAfter( Значение.Комментарий);
  Исключение
  КонецПопытки;
```

```
            КонецЦикла;
        КонецЕсли;
    КонецЦикла;
КонецЦикла;
КонецЕсли;
КонецЦикла;

// Показываем документ
Word.Visible = Истина;
// Устанавливаем активность документа
Word.Activate();
Word = Неопределено;
```

В данном примере используется коллекция `Tables`, которая предназначена для доступа ко всем таблицам документа. С помощью метода `Add` данного свойства можно добавлять новые таблицы в документ. Данный метод имеет три параметра:

1. Первый параметр — область документа, в которую помещается таблица.
2. Второй параметр — количество строк.
3. Третий параметр — количество столбцов.

Для того чтобы заполнить ячейки таблицы значениями, нужно сначала получить ссылку на данную таблицу, указав в коллекции `Tables` номер необходимой таблицы, а затем с помощью метода таблицы `Cell` указать, в какую область ячейки необходимо добавить текст.

В примере специально не описывается работа с метаданными, т. к. подразумевается, что читатель знаком с данным объектом. Для версии 7.7 системы "1С:Предприятие" приведенный код будет отличаться только принципами работы с метаданными. Все, что касается объекта `Word.Application` одинаково работает в обеих версиях системы.

Динамическое создание и выполнение макросов MS Word

С помощью объектов `Word.Application` и `Excel.Application` можно динамически добавлять макросы в документ MS Word или Excel и выполнять их на OLE-сервере.

Перед добавлением макроса необходимо проверить, существует ли доступ к VB-проекту. Для этого в MS Word 2002 (в других версиях последовательность действий может быть другой) необходимо выбрать пункт **Параметры** в меню **Сервис**; на закладке **Безопасность** нажать кнопку **Защита от макросов**

и в появившемся окне установить флажок **Доверять доступ к Visual Basic Project**.

Приведем пример добавления макроса в документ MS Word и запуска его на OLE-сервере (листинг 4.30).

Листинг 4.30. Добавление макроса в документ MS Word и запуск его на OLE-сервере

```
// Определяем текст макроса на языке VBA
ТекстМакроса = "Sub VBAFunction()
|Selection.TypeText Text:=""Добрый день, уважаемый читатель!""
|Selection.TypeParagraph
|Selection.TypeText Text:=""Добро пожаловать на наш форум""
|Application.Keyboard (1033)
|Selection.TypeText Text:="": http://forum.erpg.ru""
|Selection.TypeParagraph
|Selection.MoveUp Unit:=wdLine, Count:=2
|Selection.EndKey Unit:=wdLine, Extend:=wdExtend
|Selection.Style = ActiveDocument.Styles(""Заголовок 1"")
|Selection.MoveDown Unit:=wdLine, Count:=1
|Selection.HomeKey Unit:=wdLine
|Selection.EndKey Unit:=wdLine, Extend:=wdExtend
|Selection.Style = ActiveDocument.Styles(""Заголовок 2"")
|ActiveDocument.SaveAs FileName:=""C:/Example.doc"", FileFormat:= _
|    wdFormatDocument, LockComments:=False, Password:="""",
|    AddToRecentFiles:= _
|    True, WritePassword:="""", ReadOnlyRecommended:=False,
|    EmbedTrueTypeFonts:= _
|    False, SaveNativePictureFormat:=False, SaveFormsData:=False, _
|    SaveAsAOCELetter:=False
|End Sub";

// Создание объекта
Word = Новый COMОбъект("Word.Application");
// Добавление нового документа
Word.Documents.Add();
// Определяем ссылку на VB-проект
VBComponents = Word.ActiveDocument.VBProject.VBComponents;
// Добавляем новый макрос
Count = VBComponents.Count();
```

```
VBComponents.Add(1);
VBComponents.Item(Count + 1).CodeModule.InsertLines(1, ТекстМакроса);
// Выполняем макрос с именем VBAFunction
Word.Application.Run("VBAFunction");
// Закрываем OLE-сервер
Word.Quit();
// Очищаем переменную
Word = Неопределено;
```

Данный пример создает новый макрос с именем `VBAFunction` и запускает его на OLE-сервере. Процедура `VBAFunction` создает новый документ в формате MS Word, пишет в него две строки, форматирует их и записывает созданный документ в файл Example.doc, который будет находиться в корневом каталоге диска C:.

Проверка орфографии текстов в MS Word

Как уже было сказано ранее, текстовые процессоры и другие программы предоставляют множество полезных возможностей. Например, самый популярный текстовый редактор Microsoft Word позволяет выполнять проверку орфографии. Эту очень важную функцию редактора можно использовать в конфигурациях системы "1С:Предприятие" там, где необходимо проверить орфографию введенного текста.

Для проверки орфографии необходимо создать OLE-объект `Word.Application` и вызвать его метод `CheckSpelling`. Данный метод содержит множество параметров, но обязательным является только один — строка с текстом, который будет проверяться. Метод возвращает `Истина`, если ошибок обнаружено не будет, и `Ложь`, если текст содержит орфографические ошибки.

Рассмотрим пример работы метода `CheckSpelling` (листинг 4.31).

Листинг 4.31. Проверка наличия орфографических ошибок в тексте

```
Попытка
  Ворд = Новый COMОбъект("Word.Application"); //Создаем объект MS Word
Исключение
  Предупреждение("Microsoft Word не установлен!",,"Ошибка!");
  Возврат Ложь;
КонецПопытки;

// Проверяем строку на ошибки
```

```
ТекстБезОшибок = Ворд.CheckSpelling(Текст);

Если ТекстБезОшибок Тогда
  Сообщить("Ошибок не обнаружено!");
Иначе
  Сообщить("Текст содержит орфографические ошибки!");
КонецЕсли;
```

Данный пример позволяет определить, содержит текст орфографические ошибки или нет, но не дает знать, в каких именно словах были допущены эти ошибки. Для определения каждого неправильно написанного слова необходимо добавить новый документ в MS Word, перенести в него проверяемый текст и проверить с помощью метода `CheckSpelling` каждое слово, после чего закрыть документ (листинг 4.32).

Листинг 4.32. Проверка орфографии слов в тексте

```
// Проверяем строку на ошибки
ТекстБезОшибок = Ворд.CheckSpelling(Текст);

Если ТекстБезОшибок Тогда // Нет ошибок
 Сообщить("Ошибок не обнаружено!");

Иначе // Обнаружены ошибки, будем проверять по словам
  // Запомним, был ли Word активен
  Ворд_Активен = Ворд.Visible;
  // Добавляем новый документ
  ВремДок = Ворд.Documents.Add();
  // Получаем пустую область в начале документа
  МояОбласть = ВремДок.Range(0, 0);
  // Добавляем в эту область нашу строку
  МояОбласть.InsertBefore(Текст);
  // Подсчитываем число слов в области
  ЧислоСлов = МояОбласть.Words.Count;

  Для НомерСлова = 1 По ЧислоСлов Цикл
    // Выбираем очередное слово
    ПровОбласть = МояОбласть.Words(НомерСлова);
    ПровСлово = СокрЛП(ПровОбласть.Text);
```

```
        // Проверяем наше слово на ошибки
        СловоБезОшибок = Ворд.CheckSpelling(ПровСлово);

    Если НЕ СловоБезОшибок Тогда
      Сообщить("Ошибка в слове: " + ПровСлово);
    КонецЕсли;
  КонецЦикла;

  ВремДок.Close(wdDoNotSaveChanges,,);
  Если Не Ворд_Активен Тогда
    Ворд.Quit();
  КонецЕсли;

  Сообщить("Текст содержит орфографические ошибки!");
КонецЕсли;
```

Данный пример уже выводит те слова, в которых были обнаружены ошибки. Но MS Word, помимо проверки слов, позволяет предлагать еще и варианты замены неправильно написанных слов. Рассмотрим пример проверки орфографии слов и вывода вариантов их замены (листинг 4.33).

Листинг 4.33. Проверка орфографии слов в тексте с выводом возможных замен

```
Для НомерСлова = 1 По ЧислоСлов Цикл
  // Выбираем очередное слово
  ПровОбласть = МояОбласть.Words(НомерСлова);
  ПровСлово = СокрЛП(ПровОбласть.Text);
  // Проверяем наше слово на ошибки
  СловоБезОшибок = Ворд.CheckSpelling(ПровСлово);

  Если НЕ СловоБезОшибок Тогда
    Сообщить("Ошибка в слове: " + ПровСлово);

    // Получаем варианты замен, но их может и не быть
    // Проверяем орфографию, указав, что возвращать нужно множество замен
    Замены = ПровОбласть.GetSpellingSuggestions(,Истина,,0);
    // Получаем число возможных замен
    ЧислоЗамен = Замены.Count;
    Если ЧислоЗамен > 0 Тогда
```

```
            Для НомерЗамены = 1 По ЧислоЗамен Цикл
              СловоЗамены = СокрЛП(Замены.Item(НомерЗамены).Name);
              Сообщить(" Вариант замены: " + СловоЗамены);
            КонецЦикла;
          КонецЕсли;
        КонецЕсли;
      КонецЦикла;
```

С помощью предложенных примеров можно реализовать полноценную проверку орфографии в любых конфигурациях системы "1С:Предприятие" версий 7.7 и 8.0. Единственное требование к системе — должно быть установлено приложение Microsoft Word.

Работа с Microsoft Excel

С помощью объекта `Excel.Application`, предоставляемого OLE-сервером MS Excel, можно программно создавать электронные таблицы, а также использовать все функции MS Excel, предоставляемые пользователям.

Основным, в объектной модели Excel, является объект `Application`, содержащий коллекцию `Workbooks` объектов типа `WorkBook`. Каждый объект типа `WorkBook` содержит коллекцию объектов `WorkSheets` типа `WorkSheet`, коллекцию объектов `Charts` типа `Chart` и др. Манипуляция рабочими книгами, их листами, ячейками, диаграммами и др. осуществляется путем обращения к свойствам и методам этих объектов.

Запуск MS Excel, создание и открытие рабочих книг

Для создания примеров использования Microsoft Excel можно использовать следующий код создания контроллера:

```
Excel = Новый COMОбъект("Excel.Application").
```

Создать новую рабочую книгу Excel можно, используя метод `Add` коллекции `Workbooks` объекта `Application`:

```
Excel.WorkBooks.Add().
```

Для создания рабочей книги на основе шаблона следует указать его имя в качестве первого параметра метода `Add`:

```
Excel.WorkBooks.Add("C:\Templates\tm.xlt").
```

В качестве первого параметра этого метода можно также использовать следующие константы:

- `-4109` — рабочая книга состоит из листа с диаграммой;
- `-4167` — рабочая книга состоит из листа с данными.

В этом случае рабочая книга будет содержать лист того типа, который задан указанной константой (график, обычный лист с данными и др.).

Для открытия уже существующего документа следует воспользоваться методом Open коллекции WorkBooks:

```
Excel.Documents.Open("C:\MyExcelFile.xls").
```

Отметим, что свойство ActiveWorkBook объекта Excel.Application указывает на текущую активную рабочую книгу среди одной или нескольких открытых. Помимо этого к рабочей книге можно обращаться по ее порядковому номеру. Например, ко второй открытой рабочей книге можно обратиться с помощью записи:

```
Excel.WorkBooks[2].
```

Сделать рабочую книгу активной можно с помощью метода Activate: `Excel.WorkBooks[2].Activate()`.

Следующее, чему следует научиться, — это сохранять рабочие книги в файлах.

Сохранение, печать и закрытие рабочих книг MS Excel

Закрытие документа может быть осуществлено с помощью метода Close:

`Excel.WorkBooks[2].Close()` или `Excel.ActiveWorkBook.Close()`.

Метод Close имеет несколько необязательных (в случае позднего связывания) параметров, влияющих на правила сохранения рабочей книги. Первый из параметров принимает значения Истина или Ложь и влияет на то, сохранять ли изменения, внесенные в рабочую книгу. Второй параметр — имя файла, в котором нужно сохранить рабочую книгу (если в нее были внесены изменения). Третий параметр также принимает значения Истина или Ложь и влияет на то, будет ли пересылаться документ следующему пользователю по электронной почте.

Просто сохранить рабочую книгу, не закрывая ее, можно с помощью метода Save или SaveAs:

```
Excel.ActiveWorkBook.Save();
```

```
Excel.ActiveWorkBook.SaveAs("C:\MyWorkBook.xls").
```

Метод SaveAs имеет более десятка параметров, влияющих на то, как именно сохраняется документ (под каким именем, с паролем или без него, какова кодовая страница для содержащегося в ней текста и др.).

Закрыть среду Excel можно с помощью метода Quit объекта Excel.Application. В случае Excel этот метод параметров не имеет.

Вывод документа Excel на устройство печати можно осуществить с помощью метода PrintOut объекта WorkBook, например:

```
Excel.ActiveWorkBook.PrintOut().
```

Если нужно изменить параметры печати, следует указать значения соответствующих параметров метода `PrintOut` (в случае Excel их восемь).

Обращение к листам и ячейкам

Обращение к листам рабочей книги производится с помощью коллекции `WorkSheets` объекта `WorkBook`. Каждый член этой коллекции представляет собой объект `WorkSheet`. К члену этой коллекции можно обратиться по его порядковому номеру, например:

```
Excel.WorkBooks[1].WorkSheets[1].Name = "Страница 1".
```

Приведенная выше запись иллюстрирует, как можно изменить имя листа рабочей книги.

К листу рабочей книги можно обратиться и по имени, например:

```
Excel.WorkBooks[1].WorkSheets["Лист1"].Name = "Страница 1".
```

Обращение к отдельным ячейкам листа производится с помощью коллекции `Cells` объекта `WorkSheet`. Например, добавить данные в ячейку `B1` можно следующим образом:

```
Excel.WorkBooks[1].WorkSheets["Лист1"].Cells[1,2].Value="25".
```

Здесь первая из координат ячейки указывает на номер строки, вторая — на номер столбца.

Добавление формул в ячейки производится аналогичным способом:

```
Excel.WorkBooks[1].WorkSheets["Лист1"].Cells[3,2].Value=" =SUM(B1:B2)".
```

Очистить ячейку можно с помощью метода `ClearContents`.

Форматирование текста в ячейках производится с помощью свойств `Font` и `Interior` объекта `Cell` и их свойств. Например, фрагмент кода, приведенный в листинге 4.34, выводит текст в ячейке красным жирным шрифтом Courier размера 16 на желтом фоне.

Листинг 4.34. Форматирование текста в ячейках

```
Excel.WorkBooks[1].WorkSheets[1].Cells[3,2].Interior.ColorIndex = 6;
Excel.WorkBooks[1].WorkSheets[1].Cells[3,2].Font.ColorIndex = 3;
Excel.WorkBooks[1].WorkSheets[1].Cells[3,2].Font.Name = "Courier";
Excel.WorkBooks[1].WorkSheets[1].Cells[3,2].Font.Size = 16;
Excel.WorkBooks[1].WorkSheets[1].Cells[3,2].Font.Bold = Истина;
```

Свойство `ColorIndex` может принимать числовые значения от 1 до 56. Ниже приведены значения только основных цветов:

- `1` — белый;
- `2` — черный;

- 3 — красный;
- 5 — синий;
- 6 — желтый;
- 7 — лиловый;
- 10 — зеленый;
- 33 — голубой;
- 39 — сиреневый;
- 46 — оранжевый.

Обратиться к текущей ячейке можно с помощью свойства `ActiveCell` объекта `Excel.Application`, а узнать местоположение ячейки можно с помощью свойства `Address` объекта `Cell`, например:

```
Сообщить(Excel.ActiveCell.Address).
```

Помимо обращения к отдельным ячейкам, можно манипулировать прямоугольными областями ячеек с помощью объекта `Range`. В листинге 4.35 показан пример заполнения прямоугольного участка текстом и изменение цвета шрифта в ячейках.

Листинг 4.35. Манипулирование прямоугольной областью ячеек

```
Excel.WorkBooks[1].WorkSheets[2].Range["A1:C5"].Value = "MAV";
Excel.WorkBooks[1].WorkSheets[2].Range["A1:C5"].Font.ColorIndex = 3;
```

Объект `Range` также часто используется для копирования прямоугольных областей через буфер обмена. В листинге 4.36 приведен пример, иллюстрирующий копирование такой области.

Листинг 4.36. Копирование и вставка прямоугольной области

```
Excel.WorkBooks[1].WorkSheets[2].Range["A1:C5"].Copy();
Excel.WorkBooks[1].WorkSheets[2].Range["A11:C15"].Select();
Excel.WorkBooks[1].WorkSheets[2].Paste();
```

Обратите внимание на то, что диапазон, куда копируются данные, предварительно выделяется с помощью метода `Select`.

Отметим, что примерно таким же образом можно копировать данные и из других приложений (например, из Microsoft Word).

Довольно часто при автоматизации Excel используются его возможности, связанные с построением диаграмм. В дальнейшем будет показано, как это сделать.

Создание диаграмм

Диаграммам Excel соответствует объект `Chart`, который может располагаться как на отдельном листе, так и на листе с данными. Если объект `Chart` располагается на листе с данными, ему соответствует член коллекции `ChartObjects` объекта `WorkSheet` и создание диаграммы нужно начать с добавления элемента в эту колекцию:

```
Chart =
Excel.WorkBooks[1].WorkSheets[2].ChartObjects.Add(10,50,400,400)
```

Параметрами этого метода являются координаты левого верхнего угла диаграммы и ее размеры в пунктах (1/72 дюйма).

Если же диаграмма располагается на отдельном листе (не предназначенном для хранения данных), то ее создание нужно начать с добавления элемента в коллекцию `Sheets` объекта `Application` (которая отличается от коллекции `WorkSheets` тем, что содержит листы всех типов, а не только листы с данными):

```
Excel.WorkBooks[1].Sheets.Add(,,1,-4109).
```

В этом случае первый параметр метода `Add` указывает порядковый номер листа, перед которым нужно поместить данный лист (или листы, если их несколько). Второй параметр — порядковый номер листа, после которого нужно поместить данный лист (используется обычно либо первый, либо второй параметр). Третий параметр определяет, сколько нужно создать листов, а четвертый — какого типа должен быть лист. Значения четвертого параметра совпадают со значениями первого параметра метода `Add` коллекции `WorkBooks` объекта `Application`, и при использовании имен соответствующих констант следует определить их в приложении-контроллере.

Простейший способ создать диаграмму, с точки зрения пользователя, — создать ее с помощью соответствующего эксперта на основе прямоугольной области с данными. Точно так же можно создать диаграмму и посредством контроллера автоматизации. Для этой цели у объекта `Chart`, являющегося свойством объекта `ChartObject` (члена коллекции `ChartObjects`), имеется метод `ChartWizard`. Первым параметром этого метода является объект `Range`, содержащий диапазон ячеек для построения диаграммы, а вторым — числовой параметр, указывающий, какого типа должна быть эта диаграмма (листинг 4.37). Возможные значения параметра, отвечающего за тип диаграммы, можно найти в справочном файле.

Листинг 4.37. Создание диаграммы с помощью мастера

```
// Создание диаграммы
Ch.Chart.ChartWizard(Excel.WorkBooks[1].WorkSheets[2].Range["A1:C5"], -4100);
// Оформление диаграммы
```

```
Ch.Chart.HasTitle = 1;
Ch.Chart.HasLegend = Ложь;
Ch.Chart.ChartTitle.Text = "Пример диаграммы Excel";
Ch.Chart.Axes(1).HasTitle = Истина;
Ch.Chart.Axes(1).AxisTitle.Text = "Подпись вдоль оси абсцисс";
Ch.Chart.Axes(2).HasTitle = Истина;
Ch.Chart.Axes(2).AxisTitle.Text = "Подпись вдоль оси ординат";
```

У объекта `Chart` имеется множество свойств, отвечающих за внешний вид диаграммы. С их помощью можно изменить ее точно так же, как пользователи делают это вручную. В листинге 4.37 приводится пример создания заголовка диаграммы и подписей вдоль ее осей (отметим, что оси есть не у всех типов диаграмм).

Еще один способ создания диаграммы — определить все ее параметры с помощью свойств объекта `Chart`, включая определение точек (значений) и серий (рядов), на основе которых она должна быть построена. Данные для серии обычно находятся в объекте `Range`, содержащем строку или столбец данных, а добавление серии к диаграмме производится путем добавления члена к коллекции `SeriesCollection` (листинг 4.38).

Листинг 4.38. Определение параметров диаграммы с помощью свойств объекта `Chart`

```
Excel.WorkBooks[1].Sheets.Add(,,1,-4109);
Excel.WorkBooks[1].Sheets[1].ChartType := -4102;
Rng = App.WorkBooks[1].WorkSheets[2].Range["B1:B5"];
Excel.WorkBooks[1].Sheets[1].SeriesCollection.Add(Rng);
```

В данном примере к диаграмме, созданной на отдельном листе, специально предназначенном для диаграмм, добавляется одна серия на основе диапазона ячеек другого листа.

Далее будут рассмотрены примеры практического применения Microsoft Excel.

Выгрузка метаданных в MS Excel

Рассмотрим пример выгрузки метаданных произвольной конфигурации системы "1С:Предприятие 8.0" в лист Microsoft Excel (листинг 4.39).

Листинг 4.39. Выгрузка метаданных в MS Excel для версии 8.0

```
Попытка
  Excel = Новый COMОбъект("Excel.Application");
Исключение
```

```
  Сообщить("Не удалось открыть Excel");
  Возврат;
КонецПопытки;

// Добавляем новый лист
Excel.Application.Workbooks.Add(1);
Лист = Excel.ActiveSheet;
// Задаем имя закладки
Лист.Name = "Данные";
НомерСтроки = 0;

// Перебор типов метаданных
Для Каждого ТипОбъекта Из ВсеТипы Цикл
  Состояние(ТипОбъекта.Значение);

  Если Метаданные[ТипОбъекта.Ключ].Количество() > 0 Тогда
    // Добавляем новую строку
    НомерСтроки = НомерСтроки + 1;
    // Устанавливаем текст в ячейку
    Лист.Cells(НомерСтроки, 1).Value = ТипОбъекта.Значение;

    // Перебираем конкретные объекты
    Для каждого Объект Из Метаданные[ТипОбъекта.Ключ] Цикл
      // Добавляем новую строку
      НомерСтроки = НомерСтроки + 1;
      // Устанавливаем текст в ячейку
      Лист.Cells(НомерСтроки, 1).Value = Объект.Синоним;

      // Перебираем параметры метаданных
      Для каждого Параметр Из ВсеПараметры Цикл
        Попытка
          Количество = Объект[Параметр.Ключ].Количество();
        Исключение
          Продолжить;
        КонецПопытки;

        Если Количество > 0 Тогда
          // Добавляем новую строку
          НомерСтроки = НомерСтроки + 1;
```

```
            Лист.Cells(НомерСтроки, 1).Value = Параметр.Значение +
            " (" + Количество + ")";

            // Добавляем новую таблицу
            НомерСтроки = НомерСтроки + 1;
            Лист.Cells(НомерСтроки, 1).Value = "Имя";
            Лист.Cells(НомерСтроки, 2).Value = "Синоним";
            Лист.Cells(НомерСтроки, 3).Value = "Тип";
            Лист.Cells(НомерСтроки, 4).Value = "Комментарий";

            Для каждого Значение Из Объект[Параметр.Ключ] Цикл
              НомерСтроки = НомерСтроки + 1;
              // Заполняем содержимое таблицы
              Попытка
                Лист.Cells(НомерСтроки, 1).Value = Значение.Имя;
              Исключение
              КонецПопытки;
              Попытка
                Лист.Cells(НомерСтроки, 2).Value = Значение.Синоним;
              Исключение
              КонецПопытки;
              Попытка
                СтрокаТипов = "";
                // Выводим все типы
                Для каждого Тип Из Значение.Тип.Типы() Цикл
                  СтрокаТипов = СтрокаТипов + " " + Строка(Тип);
                КонецЦикла;
                Лист.Cells(НомерСтроки, 3).Value = СтрокаТипов;
              Исключение
              КонецПопытки;
              Попытка
                Лист.Cells(НомерСтроки, 4).Value = Значение.Комментарий;
              Исключение
              КонецПопытки;
            КонецЦикла;
          КонецЕсли;
        КонецЦикла;
      КонецЦикла;
    КонецЕсли;
```

```
КонецЦикла;

// Показываем документ
Excel.Visible = Истина;
Excel = Неопределено;
```

Данный пример работает аналогично примеру, приведенному в листинге 4.21, за исключением того, что данные выгружаются в лист MS Excel. Основной недостаток данного примера в том, что все тексты ячеек не отформатированы и представлены одним цветом, что затрудняет чтение таблицы.

В листинге 4.40 показан пример цветового оформления ячеек MS Excel и задание в них определенного шрифта.

Листинг 4.40. Оформление ячеек в MS Excel

```
// Получаем ссылку на определенную ячейку
Ячейка = Лист.Cells(НомерСтроки, НомерКолонки);
// Устанавливаем отступ в 2 символа
Ячейка.InsertIndent(2);
// Устанавливаем размер шрифта
Ячейка.Font.Size = 12;
// Устанавливаем цвет
Ячейка.Font.ColorIndex = 5;
// Устанавливаем жирный шрифт
Ячейка.Font.Bold = Истина;
// Устанавливаем ширину колонки
Ячейка.ColumnWidth = 20;
// Очищаем все границы
Ячейка.Borders(5).LineStyle = -4142;
// Устанавливаем левую границу
Ячейка.Borders(7).LineStyle = 1;
// Устанавливаем верхнюю границу
Ячейка.Borders(8).LineStyle = 1;
// Устанавливаем нижнюю границу
Ячейка.Borders(9).LineStyle = 1;
// Устанавливаем правую границу
Ячейка.Borders(10).LineStyle = 1;
// Устанавливаем выравнивание по левому краю
Ячейка.VerticalAlignment = -4160;
// Устанавливаем перенос слов
Ячейка.WrapText = Истина;
```

В первой строке данного примера в переменную Ячейка помещается ссылка на конкретную ячейку с координатами НомерСтроки и НомерКолонки. После этого через данную переменную устанавливается размер шрифта, цвет текста, ширина ячейки, границы, положение текста и т. д.

Создание сводных таблиц MS Excel

Большое количество пользователей, работающих с системой "1С:Предприятие", некоторые отчеты формируют и анализируют в Microsoft Excel. Самое удобное средство анализа данных в MS Excel — это сводные таблицы (рис. 4.1) и сводные диаграммы.

	A	B	C	D	E
1	Тип	Расход	Текущие расходы по филиалу		
2	Автор	MAV			
3					
4	Сумма, $		Месяц		
5	Счет	Аналитика	08.2004	09.2004	Общий итог
6	сч. 91		-4688,02	-407,66	-5095,68
7	сч. 60		-4812	-9116,2	-13928,2
8	сч. 68			-1739,4	-1739,4
9	сч. 69			-2929,64	-2929,64
10	сч. 70	Зарплата за август		-17028,9	-17028,9
11		Аванс за сентябрь		-12228,41	-12228,41
12	сч. 70 Итог			-29257,31	-29257,31
13	сч. 73			-1854,6	-1854,6
14	сч. 10		-346,61	-627,48	-974,09
15	сч. 20			-390	-390
16	сч. 26			-383,18	-383,18
17	Общий итог		-9846,63	-46705,47	-56552,1
18					

Рис. 4.1. Пример сводной таблицы MS Excel

С помощью встроенного языка системы "1С:Предприятие" и OLE-сервера MS Excel можно формировать сводные таблицы и диаграммы. Рассмотрим пример их формирования (листинг 4.41).

Листинг 4.41. Создание сводной таблицы MS Excel для версии 7.7

```
Функция СформироватьСводнуюТаблицу(ТЗ, Заголовок, Период, Условия)
  Перем Значение;

  Если ТЗ.КоличествоСтрок() < 2 Тогда
    Предупреждение("Невозможно сформировать сводную таблицу, содержащую
```

```
менее двух строк!", 5);
    Возврат 0;
  КонецЕсли;

  Попытка
    // Запускаем Excel-Server
    Excel = СоздатьОбъект("Excel.Application");
  Исключение
    Сообщить(ОписаниеОшибки(),"!");
    Возврат 0;
  КонецПопытки;

  Попытка
    Excel.Application.Workbooks.Add(1);
    Excel.ActiveSheet.Name = "Данные";

    // Выводим данные
    Для СтрокаТЗ = 1 По ТЗ.КоличествоСтрок() Цикл
      Для КолонкаТЗ = 1 По ТЗ.КоличествоКолонок() Цикл
        Значение = ТЗ.ПолучитьЗначение(СтрокаТЗ, КолонкаТЗ);
        Excel.ActiveSheet.Cells(СтрокаТЗ+1, КолонкаТЗ).Value = Значение;
        Если СтрокаТЗ = 1 Тогда
          ТЗ.ПолучитьПараметрыКолонки(КолонкаТЗ,"",0,0,Значение,0,"",1);
          Excel.ActiveSheet.Cells(1, КолонкаТЗ).Value = Значение;
        КонецЕсли;
      КонецЦикла;
    КонецЦикла;

    // Выводим сводную таблицу
    Excel.Application.ActiveWorkbook.PivotCaches().Add(1, "Данные!R1C1:R" +
    (1+ТЗ.КоличествоСтрок()) + "C" +
     ТЗ.КоличествоКолонок()). CreatePivotTable("", "СводнаяТаблица");
    Excel.ActiveSheet.Name = "Таблица";

    // Далее идет цветовое оформление
    Excel.Rows("1:4").Select();
    Excel.Selection.Insert(-4121);
    Excel.ActiveSheet.Cells(1, 1).Value = Заголовок;
    Excel.ActiveSheet.Cells(1, 1).Font.Size = 16;
```

```
    Excel.ActiveSheet.Cells(1, 1).Font.ColorIndex = 3;
    Excel.ActiveSheet.Cells(1, 1).Font.Bold = 1;
    Excel.ActiveSheet.Cells(1, 1).Font.Italic = 1;
    Excel.ActiveSheet.Cells(1, 1).Font.Underline = 2;

    Excel.ActiveSheet.Cells(2, 1).Value = Период;
    Excel.ActiveSheet.Cells(2, 1).Font.Size = 8;
    Excel.ActiveSheet.Cells(2, 1).Font.ColorIndex = 3;

    Excel.ActiveSheet.Cells(3, 1).Value = Условия;
    Excel.ActiveSheet.Cells(3, 1).Font.Size = 8;

    Excel.ActiveSheet.Cells(7, 1).Select();

    // Все готово. Показываем окно Excel
    Excel.Application.Visible = 1;
  Исключение
    Excel.Quit();
    Сообщить(ОписаниеОшибки(),"!");
    Возврат 0;
  КонецПопытки;

  Возврат 1;
КонецФункции //СформироватьСводнуюТаблицу
```

В данном примере функция `СформироватьСводнуюТаблицу` имеет четыре параметра.

- `ТЗ` — таблица значений, которая будет использоваться в качестве источника данных для формирования сводной таблицы. Имена колонок таблицы значений будут совпадать с именами колонок сводной таблицы MS Excel;
- `Заголовок` — название отчета (тип `Строка`), которое будет выводиться в первой строке таблицы Excel;
- `Период` — период формирования отчета (тип `Строка`), которое будет выводиться во второй строке таблицы Excel;
- `Условия` — условия и комментарии к отчету (тип `Строка`), которые будут выводиться в третьей строке таблицы Excel.

Функция возвращает `1`, если сводная таблица была успешно сформирована, и `0` — иначе.

Работа с Microsoft PowerPoint

С помощью объекта PowerPoint.Application, предоставляемого OLE-сервером MS PowerPoint, можно программно открывать и манипулировать презентациями.

Основным в объектной модели PowerPoint является объект Application, содержащий коллекцию Presentations (презентации) объектов типа Presentation. Каждый объект типа Presentation содержит коллекцию Slides (слайды) объектов типа Slide, соответствующих слайдам презентации. Слайды, в свою очередь, содержат коллекции Shapes типа Shape, соответствующие элементам слайдов презентации. Манипуляция презентациями, слайдами и их элементами осуществляется путем обращения к свойствам и методам этих объектов.

Запуск MS PowerPoint и открытие презентаций

Для открытия уже существующей презентации документа следует воспользоваться методом Open коллекции Presentations:

```
App.Presentations.Open("C:\MyPresentation.ppt");
```

Следует отметить, что свойство ActivePresentation объекта PowerPoint.Application указывает на текущую активную презентацию среди одной или нескольких открытых. Кроме того, к рабочей книге можно обращаться по ее порядковому номеру. Например, ко второй открытой рабочей книге можно обратиться так:

```
App.Presentations.Item(2).
```

В листинге 4.42 приведен пример запуска MS PowerPoint и открытие презентации.

Листинг 4.42. Запуск MS PowerPoint и открытие презентаций

```
// Создание объекта приложения PowerPoint
App = Новый COMОбъект("PowerPoint.Application");
// Установка видимости окна приложения
App.Visible = Истина;
// Открытие презентации
App.Presentations.Open("C:\Фундамент.ppt");
// Уничтожение объекта приложения
App.Quit();
```

Отметим также, что в случае PowerPoint, в отличие от Word и Excel, объект Application не имеет метода Activate для того, чтобы сделать актив-

ной конкретную презентацию среди нескольких открытых. Для решения этой задачи следует обращаться к коллекции `Windows` объекта `Presentation` либо к объектам `DocumentWindow` и `SlideShowWindow`, например: `App.Presentations.Item(1).Windows.Item(1).Activate()`.

Сохранение, печать и закрытие презентаций

Закрытие презентации можно осуществить, используя метод `Close`: `App.Presentations.Item(2).Close()` или `App.ActivePresentation.Close()`.

Обратите внимание на то, что в случае `PowerPoint` метод `Close` закрывает презентацию, не предлагая пользователю сохранить изменения. К тому же он не имеет параметров, с помощью которых можно было бы повлиять на возможность сохранения изменений.

Примечание

В случае Word и Excel метод `Close` обладал параметрами, влияющими на возможность сохранения документа перед его закрытием.

Для сохранения презентации следует воспользоваться методом `Save` или `SaveAs`. Например:

```
App.Presentations.Item(2).Save();
App.Presentations.Item(2).SaveAs("C:\Фундамент-Копия.ppt").
```

В общем случае метод `SaveAs` имеет три параметра, влияющих на то, как именно сохраняется презентация. Первый из них (обязательный) и представляет собой строку, содержащую имя файла, в котором сохраняется презентация. Если в этой строке путь к файлу не указан, файл сохраняется в текущем каталоге.

Второй параметр (необязательный). Он указывает, в каком формате сохраняется презентация. Этот параметр может принимать значения:

- `1` — формат текущей версии PowerPoint;
- `2` — формат PowerPoint (версия 7);
- `3` — формат PowerPoint (версия 4);
- `4` — формат PowerPoint (версия 3);
- `5` — сохранить как шаблон;
- `6` — формат RTF;
- `7` — формат SlideShow (*.pps);
- `8` — формат PowerPoint AddIn (*.ppa);
- `10` — формат PowerPoint 4 Far East (версия для Китая, Японии, стран Юго-Восточной Азии);

- 11 — формат по умолчанию, определяемый текущими настройками PowerPoint;
- 12 — формат HTML;
- 13 — формат HTML (версия 3);
- 14 — формат HTML для текстов в двухбайтовой кодировке;
- 15 — формат WMF;
- 16 — формат GIF;
- 17 — формат JPG
- 18 — формат PNG;
- 19 — формат BMP.

Третий параметр указывает, нужно ли внедрять в презентацию используемые в ней шрифты. Он может принимать значения Истина (внедрять шрифты) или Ложь (не внедрять шрифты). По умолчанию используется значение Ложь.

Закрыть PowerPoint можно с помощью метода Quit объекта PowerPoint.Application. В случае PowerPoint этот метод параметров не имеет.

Вывод документа PowerPoint на устройство печати можно осуществить с помощью метода PrintOut объекта Presentation, например:

```
App.Presentations.Item(2).PrintOut().
```

Если нужно изменить параметры печати, следует указать значения соответствующих (необязательных) параметров метода PrintOut. В случае PowerPoint их пять. Первые два параметра указывают номера слайдов, которые должны быть напечатаны. Третий параметр — представляет собой строку с именем файла, если вывод происходит в файл вместо принтера. Четвертый параметр (целое число) указывает, сколько экземпляров документа печатать. Пятый параметр определяет, должен ли быть напечатанный документ разобран по экземплярам, и принимает значения Истина (по умолчанию) или Ложь. Так, для вывода документа с пятого по двадцатый слайды в файл C:\MyOutput.prn в трех экземплярах можно использовать следующий код:

```
App.Presentations.Item(2).PrintOut(5,20,"c:\MyOutput.prn",3).
```

Однако, помимо параметров метода PrintOut, влияющих на то, какой принтер используется и сколько экземпляров печатается, на режим печати презентации влияет также свойство PrintOptions объекта Presentation. Это свойство представляет собой объект PrintOption, имеющий, в свою очередь, набор свойств, влияющих на то, в каком виде печатается презентация (слайды, выдачи, заметки и др.), печатаются ли рамки вокруг слайдов, сколько слайдов на странице располагается в случае печати выдач, печатается ли фон слайдов и др. Указанный набор свойств примерно отражает, что

пользователь PowerPoint может изменить, выбрав пункт **Print** из меню PowerPoint (в том числе и имя принтера, имя презентации, и число копий).

Демонстрация слайдов

Для показа слайдов используется метод `Run` объекта `SlideShowSettings`, являющегося свойством объекта `Presentation`:

```
App.Presentations.Item(1).SlideShowSettings.Run().
```

Для установки режима показа слайдов также используется объект `SlideShowSettings`. Его свойство `RangeType` указывает, какой именно фрагмент презентации нужно демонстрировать. Возможные значения этого свойства следующие:

- `1` — вся презентация;
- `2` — выделенный диапазон слайдов;
- `3` — именованная демонстрация.

Свойства `StartingSlide` и `EndingSlide` объекта `SlideShowSettings` содержат номера первого и последнего слайда демонстрируемого фрагмента. Эти свойства имеет смысл применять в тех случаях, если свойство `RangeType` этого же объекта равно `2`.

Свойство `AdvanceMode` объекта `SlideShowSettings` указывает, каким образом производится смена слайдов при демонстрации:

- `1` — ручная смена слайдов;
- `2` — смена слайдов в соответствии с временем показа каждого слайда;
- `3` — запись времени показа слайдов.

Свойство `LoopUntilStopped` объекта `SlideShowSettings`, принимающее значение `Истина` или `Ложь`, указывает, демонстрируются ли слайды непрерывно до нажатия пользователем клавиши <Esc>. Это свойство целесообразно применять, если свойство `AdvanceMode` объекта `SlideShowSettings` установлено равным `2`. Отметим также, что время показа слайдов, если таковое не записано пользователем вручную, можно установить программно, с помощью свойства `SlideShowTransition` объекта `Slide`. В листинге 4.43 приведен пример использования описанных выше свойств и методов.

Листинг 4.43. Демонстрация слайдов

```
Для Номер = 2 по 7 Цикл
 App.Presentations.Item(1).Slides.Item(Номер).SlideShowTransition.
 AdvanceTime = 1;
  App.Presentations.Item(1).Slides.Item(Номер).SlideShowTransition.
  AdvanceOnTime = Истина;
```

```
КонецЦикла;
App.Presentations.Item(1).SlideShowSettings.StartingSlide = 2;
App.Presentations.Item(1).SlideShowSettings.EndingSlide = 7;
App.Presentations.Item(1).SlideShowSettings.AdvanceMod = 2;
App.Presentations.Item(1).SlideShowSettings.LoopUntilStopped = Истина;
App.Presentations.Item(1).SlideShowSettings.RangeType = 2;
App.Presentations.Item(1).SlideShowSettings.Run();
```

В данном фрагменте кода время показа слайдов со второго по седьмой устанавливается равным одной секунде, далее указывается, что эти слайды будут демонстрироваться в соответствии с заданным временем непрерывно до нажатия клавиши <Esc>, а затем запускается показ слайдов.

Свойство `SlideShowTransition` объекта `Slide` может быть использовано для определения анимационных эффектов при смене слайдов. В листинге 4.44 приведен фрагмент кода, иллюстрирующий, каким образом устанавливаются анимационные эффекты и звуковое сопровождение при смене слайда.

Листинг 4.44. Звуковое сопровождение при смене слайда

```
App.Presentations.Item(1).Slides.Item(2).SlideShowTransition.EntryEffect = 770;
App.Presentations.Item(1).Slides.Item(2).SlideShowTransition.SoundEffect.
ImportFromFile("C:\Program Files\NetMeeting\blip.wav");
```

Различные значения свойства `EntryEffect` можно найти в интерфейсном модуле или справочном файле системы PowerPoint.

А теперь кратко остановимся на управлении поведением объектов при показе слайдов. Для управления анимацией объектов на слайдах следует использовать свойство `AnimationSettings` объекта `Shape`. Это свойство возвращает объект `AnimationSettings`, свойства которого и отвечают за анимацию данного объекта. Так, свойство `AdvanceMode` указывает, каким образом появляется объект, а свойство `AdvanceTime` — через какое время после показа слайда появляется объект. Свойство `TextLevelEffect` определяет, до какого уровня текста происходит анимация объекта. И наконец, свойство `Animate` (принимающее значение `Истина` или `Ложь`) указывает, должен ли объект вообще отображаться с анимацией. Пример управления поведением объекта на слайде приведен в листинге 4.45.

Листинг 4.45. Управление поведением объекта на слайде

```
App.Presentations.Item(1).Slides.Item(3).Shapes.Item(1).
AnimationSettings.AdvanceMode = 2;
App.Presentations.Item(1).Slides.Item(3).Shapes.Item(1).
AnimationSettings.AdvanceTime = 1;
```

```
App.Presentations.Item(1).Slides.Item(3).Shapes.Item(1).
AnimationSettings.TextLevelEffect = 16;
App.Presentations.Item(1).Slides.Item(3).Shapes.Item(1).
AnimationSettings.Animate = Истина;
```

Этот фрагмент кода устанавливает автоматическую анимацию первого объекта на третьем слайде через одну секунду после показа слайда.

Работа с Adobe Photoshop

С помощью OLE-сервера Adobe Photoshop можно автоматизировать выполнение действий, связанных с обработкой растровых изображений.

Для создания объекта приложения Adobe Photoshop используется конструкция, приведенная в листинге 4.46.

Листинг 4.46. Создание и уничтожение объекта приложения Adobe Photoshop

```
// Создание объекта приложения Photoshop
Photoshop = Новый COMОбъект("Photoshop.Application");
// Установка видимости окна приложения
Photoshop.Visible = Истина;
// Уничтожение объекта приложения
Photoshop.Quit();
```

Важно отметить то, что OLE-сервер Adobe Photoshop может создать только один экземпляр приложения, т. е. при попытке создать еще один объект — установится связь с открытым ранее приложением.

Для открытия существующего файла используется метод `Open` объекта `Photoshop`. В листинге 4.47 приведен пример открытия файла с последующим его закрытием.

Листинг 4.47. Открытие и закрытие файла картинки в приложении Adobe Photoshop

```
// Создание объекта приложения Photoshop
Photoshop = Новый COMОбъект("Photoshop.Application");
// Открытие картинки
PhotoDoc = Photoshop.Open("C:\picture.jpg");
// Закрытие документа
PhotoDoc.Close();
// Уничтожение объекта приложения
Photoshop.Quit();
```

Над открытым документом можно производить различные действия, в том числе и выполнение определенных в Adobe Photoshop операций (Action). Для выполнения операции над открытым документом необходимо вызвать метод `PlayAction` объекта `Photoshop` с указанием в параметре названия выполняемой операции. В листинге 4.48 приведен пример открытия документа Photoshop с именем `"Quadrant Colors"`, выполнение над ним операции и последующим сохранением документа.

Листинг 4.48. Запуск выполнения операции по имени

```
// Создание объекта приложения Photoshop
Photoshop = Новый COMОбъект("Photoshop.Application");
// Открытие картинки
PhotoDoc = Photoshop.Open("C:\picture.jpg");
// Запуск операции (Action) по имени
Photoshop.PlayAction("Quadrant Colors");
// Сохранение картинки
PhotoDoc.SaveTo("C:\NewPicture.psd");
// Закрытие документа
PhotoDoc.Close();
// Уничтожение объекта приложения
Photoshop.Quit();
```

Существенный недостаток данного примера состоит в том, что метод `SaveTo` не сохраняет автоматически документ, а открывает стандартный диалог сохранения файла. Для полной автоматизации всех действий над документом рекомендуется все операции (`Action`) заканчивать сохранением файла или установить отдельную операцию, которая будет сохранять документ в файл со всеми произведенными изменениями.

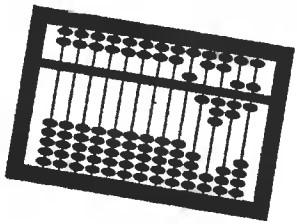
Помимо выполнения операции, с указанием ее имени, можно выполнить все определенные в Adobe Photoshop операции, перебрав коллекцию `Actions` объекта `Photoshop` (листинг 4.49).

Листинг 4.49. Запуск выполнения всех определенных в Photoshop операций

```
// Создание объекта приложения Photoshop
Photoshop = Новый COMОбъект("Photoshop.Application");
// Открытие картинки
PhotoDoc = Photoshop.Open("C:\picture.jpg");
// Перебор коллекции операций и запуск каждой из них
```

```
Для каждого Action из Photoshop.Actions Цикл
  КодОтвета = Вопрос("Запустить " + Action.Name +
  "?",РежимДиалогаВопрос.ДаНетОтмена,,,"Запустить операцию?");
  Если КодОтвета = КодВозвратаДиалога.Да Тогда
    Action.Play();
  ИначеЕсли КодОтвета = КодВозвратаДиалога.Отмена Тогда
    Прервать;
  КонецЕсли;
КонецЦикла;
// Закрытие документа
PhotoDoc.Close();
// Уничтожение объекта приложения
Photoshop.Quit();
```

В данном примере перебираются все элементы коллекции `Actions`. Для каждой операции (`Action`) появляется вопрос с просьбой подтвердить ее выполнение, при утвердительном ответе она выполняется.



Глава 5

Использование ADO

Понятие ADO

ADO (Microsoft ActiveX Data Objects) — это набор библиотек, содержащих COM-объекты. Основное их назначение — реализация прикладного программного интерфейса для доступа к базам данных и использование в клиентских приложениях. ADO использует библиотеки OLE DB, предоставляющие низкоуровневый интерфейс для доступа к данным. OLE DB, со своей стороны, обеспечивают доступ к данным с помощью COM-интерфейсов.

OLE DB и ADO являются частью универсального механизма доступа к данным Microsoft (Microsoft Universal Data Access), который обеспечивает высокопроизводительный доступ к различным источникам информации (включая реляционные и нереляционные базы данных), в том числе к данным, хранящимся на мэйнфреймах, данным электронной почты и файловой системы, текстовым, графическим и др. Для многих современных приложений, которые используют различную информацию, характерно подобное разнообразие ее источников. Более того, вполне очевидно, что могут появляться новые форматы данных и способы их хранения. Поэтому разумным требованием к универсальному механизму доступа к ним могла бы стать возможность поддержки не только существующих в настоящее время их форматов и источников, но и форматов данных, которые будут созданы в будущем.

Назначение универсального механизма доступа к данным фирмы Microsoft — предоставить доступ к перечисленным их источникам с помощью единой модели доступа.

Рассмотрим основные компоненты архитектуры универсального механизма доступа к данным Microsoft и обсудим их более детально.

- *Microsoft ActiveX Data Objects (ADO)* — представляет собой программный интерфейс для доступа к данным из приложений. С точки зрения программирования, ADO и его расширения являются упрощенным высокоуровневым объектно-ориентированным интерфейсом к OLE DB.

- *OLE DB* — это низкоуровневый интерфейс для доступа к данным. ADO применяет OLE DB, но можно использовать OLE DB и напрямую, минуя ADO.
- *Open Database Connectivity (ODBC)* — стандартный способ доступа к реляционным базам данных. Этот компонент универсального механизма доступа оставлен с целью обеспечения совместимости с прежними версиями программного обеспечения. В современных приложениях применению ODBC-драйверов предпочитают использование OLE DB-провайдеров.

Для доступа к источнику данных с помощью OLE DB требуется, чтобы на компьютере, где используется клиентское приложение, был установлен OLE DB-провайдер для данной СУБД. OLE DB-провайдер представляет собой динамическую библиотеку DLL, загружаемую в адресное пространство клиентского приложения и используемую для доступа к источнику данных. Для каждого типа СУБД нужен собственный OLE DB-провайдер, так как провайдеры базируются на функциях клиентских API (Application Programming Interface, программный интерфейс приложения), разных для различных СУБД.

ADO представляет собой высокоуровневый программный интерфейс для доступа к OLE DB-интерфейсам. Он позволяет манипулировать данными с помощью любых OLE DB-провайдеров, входящих как в состав Microsoft Data Access Components, так и в состав программных продуктов, разработанных сторонними производителями. ADO содержит набор объектов, используемых для соединения с источником информации с целью чтения, добавления, удаления или модификации предоставляемых им данных.

- Объект `Connection` применяется для установки связи с источником данных. Он представляет единственную сессию. Этот объект позволяет изменить параметры соединения с базой данных, а также начать или завершить транзакцию. Используя объект `Connection`, можно выполнять команды (например, SQL-запросы) с помощью метода `Execute`. Если команда возвращает набор данных, то автоматически создается объект `Recordset`, который возвращается в результате выполнения этого метода.
- Объект `Error` используется для получения сведений об ошибках, возникающих в процессе выполнения приложения.
- Объект `Command` представляет собой команду, которую можно выполнить в источнике данных. Команда может содержать SQL-предложение или вызов хранимой процедуры. В последнем случае для определения параметров процедуры может быть использована коллекция `Parameters` объекта `Command`.
- Объект `RecordSet` — это набор записей, полученных из источника данных, и может быть использован для добавления, удаления, изменения, просмотра записей. Данный объект может быть открыт непосредственно или создан с помощью объектов `Connection` или `Command`.

- Объект `Field` — это колонка в наборе данных, представленных объектом `Recordset`. Объект может быть использован для получения значений конкретного поля таблицы, его модификации, извлечения метаданных, таких как имя колонки и тип данных.
- Объект `Record` — представляет одну запись внутри объекта `Recordset` и может быть использован для работы с гетерогенными и иерархическими данными (добавлен в библиотеку ADO 2.5, являющуюся составной частью операционной системы Windows 2000).
- Объект `Stream` представляет двоичные данные, связанные с объектом `Record`. Например, если объект `Record` представляет собой файл, то его объект `Stream` должен содержать данные внутри этого файла (добавлен в ADO 2.5).

Соединение с источником данных

Для установки связи с источником данных используется объект `Connection`. Рассмотрим два варианта создания соединения — явного (листинг 5.1) и неявного (листинг 5.2).

Листинг 5.1. Создание явного соединения

```
// Создание объекта установки связи с источником данных
Connection = Новый COMОбъект("ADODB.Connection");

// Подключение к источнику данных
Попытка
  Connection.Open(СтрокаПодключения);
Исключение
  Сообщить(ОписаниеОшибки());
  Возврат;
КонецПопытки;

// Создание объекта выполнения команды
Command = Новый COMОбъект("ADODB.Command");
// Указание активного соединения
Command.ActiveConnection = Connection;
// Определение текста команды
Command.CommandText = "SELECT * FROM titles";
// Определение типа команды
Command.CommandType = 1;
```

```
// Создание объекта набора записей
RecordSet = Новый COMОбъект("ADODB.RecordSet");
// Выполнение и получение набора данных
RecordSet = Command.Execute();

// Перебор данных
Пока RecordSet.EOF() = 0 Цикл
  Сообщить(RecordSet.Fields(0).Value);
  RecordSet.MoveNext();
КонецЦикла;

// После того как набор записей уже не нужен, его следует закрыть
RecordSet.Close();
Connection.Close();
```

Листинг 5.2. Создание неявного соединения

```
// Создание объекта выполнения команды
Command = Новый COMОбъект("ADODB.Command");
// Указание активного соединения без использования объекта Connection
Command.ActiveConnection = СтрокаПодключения;

// Определение текста команды
Command.CommandText = "SELECT * FROM titles";
// Определение типа команд
Command.CommandType = 1;

// Создание объекта набора записей
RecordSet = Новый COMОбъект("ADODB.RecordSet");
// Выполнение и получение набора данных
RecordSet.Open(Command);

// Перебор данных
Пока RecordSet.EOF() = 0 Цикл
  Сообщить(RecordSet.Fields(0).Value);
  RecordSet.MoveNext();
КонецЦикла;
```

```
// После того как набор записей уже не нужен, его следует закрыть
RecordSet.Close();
Connection.Close();
```

Из приведенных примеров видно, что основное отличие явного соединения от неявного заключается в том, что в последнем случае не используется объект `Connection`. Для получения более структурированного и понятного кода рекомендуется использовать явное соединение.

В обоих примерах для описания источника данных использовалась строковая переменная `СтрокаПодключения`. Значение данной переменной можно формировать вручную, либо с помощью программы связи с данными. Программа связи с данными используется для создания и управления подключениями между локальным компьютером и хранилищами данных OLE DB. Запустить программу связи с данными можно двумя способами — вручную и программно.

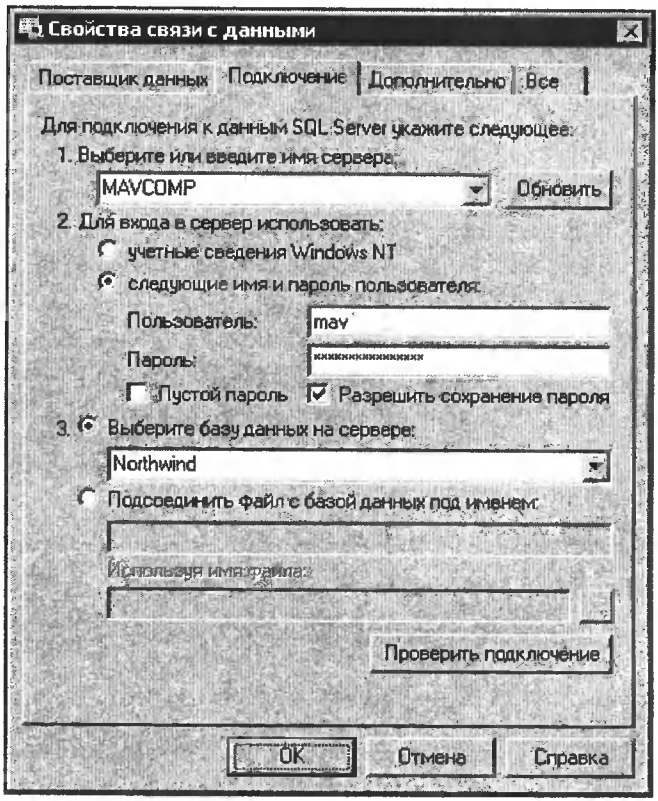


Рис. 5.1. Диалог программы связи с данными

Для ручного запуска достаточно создать пустой текстовый файл с расширением UDL и в любом файловом менеджере запустить его. Этом случае откроется диалог (рис. 5.1), в котором настраиваются параметры соединения, а после нажатия кнопки **OK** сформировавшаяся строка соединения запишется в созданный UDL-файл.

Пример программного запуска утилиты настройки связи с данными приведен в листинге 5.3.

Листинг 5.3. Запуск программы связи с данными

```
// Программа для формирования и редактирования строки соединения
DataLinks = Новый COMОбъект("DataLinks");
// Объекту Connection устанавливаем введенное значение строки соединения
Connection.ConnectionString = СтрокаПодключения;
// Открываем диалог программы связи с данными
DataLinks.PromptEdit(Connection);
// Возвращаем новое значение строки соединения
СтрокаПодключения = Connection.ConnectionString;
Сообщить(СтрокаПодключения);
```

Важно отметить, что при программном запуске утилиты связи с данными все манипуляции со строкой соединения происходят через объект `Connection`. Данная особенность еще раз говорит в пользу того, что лучше использовать явное соединение с источником данных (см. листинг 5.1).

Например, для доступа к базе данных Microsoft SQL Server строка соединения будет выглядеть следующим образом:

"Provider=SQLOLEDB.1; Password=erpg; Persist Security Info=True; User ID=mav; Initial Catalog=Northwind; Data Source=MAVCOMP"

Если же строка соединения записана во внешнем UDL-файле, то строка соединения будет выглядеть иначе:

"File Name=C:\DataLinks\MyDataLink.udl"

Рассмотрим основные свойства строки соединения.

- Provider — названия поставщика OLE DB.
- User ID — имя пользователя.
- Password — пароль пользователя.
- Persist Security Info — признак сохранения пароля в строке соединения (True/False).
- Initial Catalog — используемая база данных.

- Data Source — название источника данных (например, имя сервера, имя файла и т. п.);
- File Name — полное имя UDL-файла.

Все остальные свойства зависят от конкретного OLE DB-провайдера.

Продолжая рассматривать пример, приведенный в листинге 5.1, стоит остановиться на объекте `Connection`. Это самый важный объект ADO, отвечающий за связь между приложением и базой данных. У него очень много методов и свойств, и одно из них, `ConnectionString`, описанное ранее. Оно определяет собственно строку инициализации ODBC-соединения.

Рассмотрим основные свойства объекта `Connection`.

- `ConnectionTimeout` — задает время в секундах, в течение которого ожидается установление ODBC-соединения. По умолчанию равно 15. Конкретное значение имеет смысл устанавливать в том случае, когда сервер, вследствие своей загруженности или высокого трафика, не успевает обрабатывать запросы. Если установить значение равное 0, то сервер будет ждать установления соединения бесконечно долго.
- `CommandTimeout` — задает время в секундах, в течение которого ожидается исполнение запросов. По умолчанию равно 30.
- `ConnectionString` — строка инициализации ODBC-соединения.
- `DefaultDatabase` — база данных по умолчанию для данного соединения.
- `State` — определяет состояние соединения (`0` — закрыто, `1` — открыто).

Рассмотрим основные методы объекта `Connection`.

- `Open` — метод открывает доступ к базе данных и имеет три необязательных параметра:
  - `ConnectionString` — строка соединения с источником данных, причем если параметр `ConnectionString` не задан, то используется свойство `connection.ConnectionString`;
  - `UserID` — имя пользователя;
  - `Password` — пароль (обычно `UserID` и `Password` задаются непосредственно в `ConnectionString`, поэтому их можно не задавать).
- `Close` — разрывает связь с базой данных, при этом закрываются также все объекты типа `Recordset`, связанные с данным объектом, однако память, отведенная под объект `Connection`, не освобождается, т. е. в дальнейшем можно снова выполнить команду `Open` для повторного доступа к базе данных.
- `Execute` — метод исполняет SQL-запрос или хранимую процедуру и возвращает объект типа `Recordset`, причем возможно задание трех параметров:
  - `CommantText` — строка, содержащая SQL-запрос, имя таблицы, хранимой процедуры или прочий исполняемый текст;

- `RecordsAffected` — переменная, в которую возвращается количество записей в результате исполнения запроса;
- `Options` — определяет тип параметра `CommandText` (полезен для оптимизации исполнения запроса) и может иметь значения:
  - ◊ `1` — SQL-запрос или другое текстовое выражение;
  - ◊ `2` — таблица;
  - ◊ `4` — хранимая процедура;
  - ◊ `8` — тип параметра неизвестен.

Примечание

Важно отметить, что объект `RecordSet`, созданный с помощью этого метода, доступен только для чтения и его сканирование может производиться только вперед по записям. Для того чтобы иметь возможность вставлять или изменять записи, а также двигаться вперед и назад по ним, необходимо непосредственно создать этот объект и задать некоторые его свойства до исполнения соответствующего запроса.

Выполнение SQL-запросов и хранимых процедур

Ранее было сказано, что выполнить SQL-запрос или хранимую процедуру можно с помощью метода `Execute` объекта `Connection`, но наиболее правильно для этих целей использовать специализированный объект — `Command`. Это связано с тем, что объект `Command` специально предназначен для обработки параметров SQL-операторов, а также хранимых процедур и при возврате объекта `RecordSet` минимизирует количество информации, которой обмениваются поставщик данных и приложение.

Объект `Command` описывает команду (запрос или оператор), которая обрабатывается источником данных. Благодаря этому объекту обеспечивается простота и эффективность работы с запросами или хранимыми процедурами. С помощью объекта `Command` можно определить SQL-строку, хранимую процедуру или имя таблицы.

Как и многие другие объекты модели ADO, объект `Command` может быть создан либо с помощью объекта `Connection` (листинг 5.1), либо независимо от него (листинг 5.2).

После того как объект `Command` будет создан и подключен к поставщику данных, его можно использовать для выполнения процедур и SQL-операторов, которые будут либо воздействовать на данные в источнике данных, либо возвращать объект `Recordset`.

Рассмотрим пример добавления новой записи в таблицу базы данных (листинг 5.4).

Листинг 5.4. Добавление новой записи в таблицу базы данных

```
ТекстЗапроса = "
|INSERT
|INTO    Users(ParentId, IsFolder, FIO, Login, Password, Description)
|VALUES  (" + 2 + "," + 1 + ",'" + Пользователь.Наименование + "','" +
Пользователь.Логин + "','pass','Произвольное описание')";

Command = СоздатьОбъект("ADODB.Command");
Command.ActiveConnection = Connection;
Command.CommandText = ТекстЗапроса;
Command.CommandType = 1;
// Выполнение запроса
RecordSet = Command.Execute();
```

Примечание

Отдельно на тексте запроса останавливаться не будем, так как подразумевается, что читатель знаком с языком SQL.

Рассмотрим основные свойства объекта `Command`.

- `ActiveConnection` — данное свойство определяет соединение, с которым работает объект `Command`, и содержит строку, описывающую соединение либо указывающую на открытый в данный момент объект `Connection`. В первом случае создается новое соединение, предназначенное специально для объекта `Command` и объекта `Recordset`, который также создается.
- `CommandText` — это свойство определяет SQL-оператор, имя хранимой процедуры либо имя таблицы, с которой будет работать объект `Command`. Как правило, оно содержит вызовы хранимых процедур или SQL-операторов, кроме этого может включать и специфические команды, поддерживаемые поставщиком данных.
- `CommandTimeout` — данное свойство предназначено для определения интервала времени ожидания объектом `Command` результатов обращения к поставщику данных. По истечении этого периода объект `Command` отменяет попытку установить соединение и генерирует сообщение об ошибке. Стандартное значение данного свойства равно 30 секундам. Однако оно может быть и больше, если предполагается, что процесс загрузки данных поставщиком занимает много времени.

- `CommandType` — данное свойство используется для оптимизации процесса обработки объекта `Command` поставщиком данных. При указании конкретного типа команды, определенной свойством `CommandText` объекта `Command`, поставщик данных освобождается от распознавания типа выполняемой команды. Аналогично аргументу `Option` метода `Execute` объекта `Connection`, данное свойство может принимать следующие значения:
  - `1` — SQL-запрос или прочее текстовое выражение;
  - `2` — таблица;
  - `4` — хранимая процедура;
  - `8` — тип параметра неизвестен.
- `State` — свойство возвращает множество значений сумм одной или нескольких констант `ObjectStateEnum`. Эти константы описывают — является ли объект `Command` открыт, закрыт, в соединении и т. п. Возможные значения:
  - `0` — объект закрыт;
  - `1` — объект открыт;
  - `2` — соединяется;
  - `4` — выполняется;
  - `8` — выбран.

Рассмотрим основные методы объекта `Command`.

- `Execute` — метод исполняет SQL-запрос или хранимую процедуру. Возвращает объект типа `Recordset`.
- `CreateParameter` — данный метод используется для создания новых объектов `Parameter`, которые добавляются в семейство `Parameters` объекта `Command` до того, как будет выполнен запрос. Объект `Parameter` описывает параметр, передаваемый в SQL-оператор или хранимую процедуру, которая указана в свойстве `CommandText` объекта `Command`.

Рассмотрим более подробно использование параметров, передаваемых в хранимые процедуры. Для передачи параметров процедуре необходимо выполнить следующие действия.

1. Вызвать метод `CreateParameter` и создать новый объект `Parameter`.
2. Добавить объект в семейство `Parameters` объекта `Command`.
3. Воспользовавшись методом `Execute` объекта `Command`, получить объект `Recordset`.

Рассмотрим пример выполнения перечисленных действий (листинг 5.5).

Листинг 5.5. Передача параметров хранимой процедуре

```
// Создание нового объекта Command
Command = Новый COMОбъект("ADODB.Command");
// Организация связи с поставщиком данных для объекта Command
Command.ActiveConnection = Connection;
// Задание хранимой процедуры
Command.CommandText = "SP_GET_AUTHOR_INFO";
// Применение метода CreateParameter объекта Command для
// создания нового объекта Parameter, который будет
// использован при выполнении хранимой процедуры.
Parameter = Command.CreateParameter("@USERNAME", 200, 1, 12, "MAV");
// Добавление в семейство Parameters объекта Command нового параметра
Command.Parameters.Append(Parameter);
// Получение объекта Recordset
Recordset = Command.Execute();
```

Метод `CreateParameter` применяется следующим образом:

```
CreateParameter(Name, Type, Direction, Size, Value).
```

Рассмотрим параметры этого метода.

- `Name` — строковое значение, определяющее имя создаваемого параметра.
- `Type` — тип параметра, передаваемого в объект `Command`. Хотя тип значения параметра, передаваемого методу `CreateParameter`, является вариантным, включение этого необязательного аргумента часто влияет на операцию, выполняемую поставщиком данных. Все возможные значения перечислены в табл. 5.1.
- `Direction` — если вам не приходилось использовать хранимые процедуры, то вам не известен тот факт, что они могут как принимать параметры, так и возвращать их. Например, если применяется хранимая процедура для вычисления значений, получаемых при обмене валюты, она будет возвращать результат вычислений в выходном параметре. Данный аргумент определяет способ использования параметров хранимой процедуры: для ввода данных, для вывода значений, или же и для того и для другого. Все возможные значения перечислены в табл. 5.2.
- `Size` — если объекту `Parameter` передаются данные переменной длины, то прежде чем добавить этот объект в семейство `Parameters`, нужно задать значение его свойства `Size`, иначе произойдет ошибка.
- `Value` — этим аргументом устанавливается значение параметра, передаваемого объекту `Connection` (ADO позволяет задавать для данного аргу-

мента длинные двоичные значения). Например, если требуется получить значение, возвращаемое хранимой процедурой — необходимо добавить параметр с именем Return: `ReturnParameter = Command.CreateParameter("Return",3,4)`. Соответственно результат, возвращенный хранимой процедурой, будет находиться в свойстве `ReturnParameter.Value`.

Таблица 5.1. *Типы параметров, используемые в объектах ADO*

Константа ADO	Значение	Описание
adBigInt	20	8-байтовое целое со знаком
adBinary	128	Двоичное значение
adBoolean	11	Булево значение
adBSTR	8	Символьная строка, заканчивающаяся символом Null (в кодировке Unicode)
adChar	129	Строковое значение
adCurrency	6	Денежная сумма (8-байтовое целое со знаком, кратное 10 000)
adDate	7	Дата
adDBDate	133	Дата (в формате "ггггммчч")
adDBTime	134	Время (в формате "ччммсс")
adDBTimeStamp	135	Точная дата и время (в формате "гггг_мм_чч_чч_мм_сс" плюс тысячные доли секунды)
adDecimal	14	Десятичное число с фиксированными размерами целой и дробной частей
adDouble	5	Значение с плавающей точкой двойной точности
adEmpty	0	Значение не определено
adError	10	32-битовый код ошибки
adGUID	72	Полностью уникальный идентификатор (GUID)
adIDispatch	9	Указатель на интерфейс IDispatch OLE-объекта
adInteger	3	4-байтовое целое со знаком
adIUnknown	13	Указатель на интерфейс IUnknown OLE-объекта
adLongVarBinary	205	Длинное двоичное значение (только для объекта Parameter)
adLongVarChar	201	Длинное строковое значение, заканчивающееся символом Null (только для объекта Parameter)

Таблица 5.1 (окончание)

Константа ADO	Значение	Описание
adLongVarWChar	203	Длинное строковое значение (только для объекта Parameter)
adNumeric	131	Десятичное число фиксированной точности
adSingle	4	Значение одинарной точности с плавающей точкой
adSmallInt	2	2-байтовое целое со знаком
adTinyInt	16	1-байтовое целое со знаком
adUnsignedBigInt	21	8-байтовое целое без знака
adUnsignedInt	19	4-байтовое целое без знака
adUnsignedSmallInt	18	2-байтовое целое без знака
adInsignedTinyInt	17	1-байтовое целое без знака
adUserDefined	132	Переменная, определяемая пользователем
adVarBinary		Двоичное значение (только для объекта Parameter)
adVarChar	200	Строковое значение (только для объекта Parameter)
adVariant	12	Вариантная переменная OLE-автоматизации
adVarWChar	202	Символьная строка Unicode, заканчивающаяся символом Null (только для объекта Parameter)
adWChar	130	Символьная строка Unicode, заканчивающаяся символом Null

***Таблица 5.2.** Допустимые значения аргумента `Direction` метода `CreateParameter`*

Константа ADO	Значение	Описание
adParamUnknown	0	Назначение параметра не определено
adParamInput	1	Входной параметр (по умолчанию) — используется для передачи информации в хранимую процедуру
adParamOutput	2	Выходной параметр — применяется для получения данных из хранимой процедуры посредством SQL-параметра OUTPUT
adParamInputOutput	3	Входной и выходной параметр — используется как для передачи информации в хранимую процедуру, так и для получения от нее значений
adParamReturnValue	4	Возвращаемое значение — применяется для перехвата кода завершения хранимой процедуры

Обращение к результатам запроса

Ранее очень часто встречалось использование объекта `RecordSet`. Объект типа `RecordSet` представляет таблицу — результат запроса к базе данных. Экземпляр объекта создается командой:

```
Новый COMОбъект("ADODB.RecordSet").
```

Для перемещения по строкам, или записям, таблицы, объект использует курсор, указывающий на текущую запись. При открытии объекта курсор указывает на самую первую запись. Для удобства работы можно разбивать таблицу на страницы, варьируя их размер. По умолчанию, если не задано свойство `CursorType`, перемещаться по записям можно только вперед.

Для того чтобы определить список полей таблицы, используется коллекция `Fields` объекта `RecordSet`. Каждый элемент коллекции представляет собой объект типа `Field`. Чтобы вывести список полей таблицы, можно использовать следующий код (листинг 5.6).

Листинг 5.6. Вывод списка полей, входящих в `RecordSet`

```
// Создание объекта выполнения команды
Command = Новый COMОбъект("ADODB.Command");
// Указание активного соединения
Command.ActiveConnection = Connection;
// Определение текста команды
Command.CommandText = "SELECT * FROM titles";
// Определение типа команды
Command.CommandType = 1;

// Создание объекта набора записей
RecordSet = Новый COMОбъект("ADODB.RecordSet");
// Выполнение и получение набора данных
RecordSet = Command.Execute();

// Перебор полей RecordSet
Для НомерКолонки = 0 По RecordSet.Fields.Count-1 Цикл
  Сообщить("Имя колонки" + RecordSet.Fields.Item(НомерКолонки).Name);
КонецЦикла;
```

Использовать данный прием очень удобно в том случае, когда заранее неизвестен состав полей выборки.

Рассмотрим следующий пример, в котором демонстрируется вывод всех значений полей всех записей, причем состав полей, как и в предыдущем примере, заранее неизвестен (листинг 5.7).

Листинг 5.7. Выборка результата запроса

```
// Перебор записей выборки
КоличествоПолей = RecordSet.Fields.Count;
Пока RecordSet.EOF() = 0 Цикл
  // Перебор полей выборки
  Для НомерКолонки = 0 По КоличествоПолей-1 Цикл
    ИмяПоля = RecordSet.Fields.Item(НомерКолонки).Name;
    Значение = RecordSet.Fields(ИмяПоля).Value;
    Сообщить("Поле: " + ИмяПоля + " = " + Значение);
  КонецЦикла;
  // Переводим курсор на следующую запись
  RecordSet.MoveNext();
КонецЦикла;
```

Для большего понимания данного примера и возможностей объекта `RecordSet` рассмотрим свойства данного объекта.

- `ActiveConnection` — объект типа `Connection`, к которому привязан данный `Recordset`.
- `Source` — строка-источник запроса, т. е. SQL-выражение, имя таблицы или хранимой процедуры.
- `Filter` — фильтр, т. е. та часть SQL-запроса, которая находится в составе `WHERE`.
- `CursorType` — тип курсора. Значениями свойства, которые можно изменять только до открытия объекта, могут быть:
  - `0` — является значением по умолчанию (Forward only-курсор) и позволяет перемещаться только вперед по записям таблицы (является наиболее оптимальным в тех случаях, когда нужно провести лишь единственный проход по записям);
  - `1` — определяет подобие динамическому курсору (Keyset-курсор) за исключением того, что вы не можете видеть записи таблицы, добавленные другими пользователями базы данных;
  - `2` — позволяет любые перемещения по таблице, а также добавление, изменение и удаление записей, которые видны всем пользователям базы данных (Динамический курсор);

- 3 — позволяет производить поиск по таблице и создавать отчет (Статический курсор — копия таблицы), причем добавления, изменения и удаления, проведенные другими пользователями, невидимы (полезен, например, при постраничном выводе таблицы с возможностью перемещаться вперед или назад по страницам).

- State — определяет состояние объекта:
  - 0 — закрыт;
  - 1 — открыт.
- BOF — логическая величина (Истина или Ложь). Определяет факт того, что курсор находится перед первой записью таблицы.
- EOF — логическая величина. Определяет, что курсор находится за последней записью таблицы.
- EditMode — определяет состояние редактирования текущей записи (проведен ли метод Update для сохранения текущих изменений). Свойство может принимать следующие значения:
  - 0 — показывает, что редактирование не проводилось;
  - 1 — показывает, что данные в текущей записи редактировались, но изменения еще не сохранены;
  - 2 — показывает, что текущая запись добавлена с помощью метода AddNew, но до сих пор не сохранена.
- RecordCount — определяет количество записей в таблице (доступно, только если CursorType = 3).
- AbsolutePosition — перемещает курсор на новую запись под данным номером. Должен иметь значение в интервале от 1 до RecordCount (доступно, только если CursorType = 3).
- Bookmark — закладка в виде числа. Никак не связана с номером записи.

Примечание

Можно при сканировании по таблице запомнить значение Bookmark в некоторой временной переменной, а затем вернуться обратно на данную запись (доступно, только если CursorType = 3).

- PageSize — задает число записей таблицы на странице. По умолчанию равен 10 (доступно, только если CursorType = 3).
- PageCount — определяет количество страниц. Автоматически пересчитывается при изменении PageSize (доступно, только если CursorType = 3);
- AbsolutePage — перемещает курсор на начало данной страницы (доступно, только если CursorType = 3).

Помимо перечисленных свойств, объект RecordSet имеет ряд методов.

- Open — исполняет запрос и открывает таблицу-результат. Общий формат вызова: RecordSet.Open(Sourse, ActiveConnection, CursorType), причем необязательные параметры Source, ActiveConnection и CursorType здесь обозначают то же, что и соответствующие свойства объекта Recordset. Кроме того, параметр ActiveConnection может быть не только объектом типа Connection, но и просто строкой инициализации — ConnectionString.
- Close — закрывает объект.
- Requery — заново пересчитывает таблицу. Эквивалентен последовательному исполнению методов Close и Open.
- Clone — возвращает объект типа Recordset (копию исходного объекта). Полезен в случаях, когда нужно использовать одновременно несколько записей из одной таблицы.
- MoveFirst, MoveLast, MoveNext, MovePrevious — перемещают курсор на первую, последнюю, следующую или предыдущую запись таблицы соответственно.
- Move — перемещает курсор на новое положение. Общий формат вызова: Recordset.Move(NumRecords, Start), причем NumRecords здесь — количество записей, на которые нужно переместить курсор. Положительное значение означает перемещение вперед, отрицательное — назад по записям. Параметр Start определяет точку отсчета, откуда нужно переместить курсор.

Примечание

Может быть либо закладкой, либо одним из возможных значений: 0 — значение по умолчанию, точка отсчета — текущая запись; 1 — отсчет идет от первой записи таблицы; 2 — отсчет идет от последней записи таблицы.

- AddNew — добавляет запись. Можно сначала добавить запись, затем определить значение полей. А можно это сделать и одной командой: RecordSet.AddNew(Fields, Values). Здесь Fields — имя или массив имен полей, которые нужно определить, Values — значение или массив значений этих полей (чтобы сохранить добавленную запись, необходимо выполнить метод Update).
- Delete — удаляет текущую запись или группу записей в зависимости от параметра. Формат записи: Recordset.Delete(AffectRecords), где параметр AffectRecords может принимать одно из возможных значений:
  - 1 — (значение по умолчанию) удаляет только текущую запись;
  - 2 — удаляет записи, удовлетворяющие свойству Filter, которое должно быть задано заранее.

- `Update` — сохраняет изменения в текущей записи. Изменения в записи можно произвести предварительно или непосредственно в команде: `Recordset.Update(Fields, Values)`. Параметр `Fields` определяет здесь имя или массив имен полей, которые нужно определить, параметр `Values` — значение или массив значений этих полей. Если курсор перемещен с добавленной или измененной записи, сервер автоматически выполнит метод `Update`.
- `CancelUpdate` — отменяет любые изменения или добавления записей, проведенные до использования метода `Update`.

Работа со структурой базы данных (ADOX)

Для работы со структурой базы данных существует технология *ADO Extensions for Data Definition and Security* (ADOX). *ADOX* — представляет собой набор объектов, позволяющих манипулировать метаданными в базах данных и управлять объектами, отвечающими за безопасность.

ADOX предоставляет более универсальный способ манипуляции метаданными, не требующий знания SQL для того, чтобы получить структуру базы данных или даже создать новые объекты.

Примечание

Обратите внимание на то, что ADOX работает далеко не со всеми базами данных — его функциональность ограничена Microsoft Access и Microsoft SQL Server, а также несколькими другими СУБД.

ADOX обладает собственной объектной моделью, состоящей из следующих объектов:

- `Catalog` — представляет всю схему базы данных;
- `Table` — представляет таблицу в базе данных;
- `Column` — представляет колонку в таблице;
- `Index` — представляет индекс внутри таблицы;
- `Key` — представляет первичный или внешний ключ;
- `Group` — представляет группу пользователей;
- `User` — представляет индивидуального пользователя базы данных внутри группы;
- `Procedure` — представляет хранимую процедуру внутри базы данных;
- `View` — является представлением (view) внутри базы данных.

Иерархия объектов ADOX начинается с объекта `Catalog`. Этот объект содержит коллекции таблиц, представлений, процедур, пользователей и групп

и может быть использован для открытия существующей базы данных (с помощью объекта ADO Connection), а также для создания новой. Имея объект Catalog, можно работать с таблицами, процедурами и представлениями. Например, просматривая коллекцию Tables, можно узнать, какие таблицы имеются в базе данных, а также получить более детальные сведения о таблицах, изучив коллекции Columns, Indexes и Keys объекта Table. Изучая свойства объектов базы данных, можно получить сведения о метаданных и, в частности, сохранить их в отдельном файле или куда-либо перенести. Используя коллекции Users и Groups, можно манипулировать правилами доступа к данным, создавая отдельных пользователей или группы пользователей базы данных.

Чтение структуры базы данных

Рассмотрим пример вывода информации о таблицах, представлениях и процедурах базы данных (листинг 5.8).

Листинг 5.8. Вывод информации о таблицах, представлениях и хранимых процедурах

```
// Начальная инициализация
Catalog = Новый COMОбъект("ADOX.Catalog");
Catalog.ActiveConnection = СтрокаПодключения;

// Выводим таблицы
Для НомерТаблицы = 0 По Catalog.Tables.Count-1 Цикл
  ИмяТаблицы = Catalog.Tables.Item(НомерТаблицы).Name;
  Сообщить("Таблица: " + ИмяТаблицы);
КонецЦикла;

// Проверка текущего источника данных на поддержку коллекции Views
Попытка
  КоличествоПредставлений = Catalog.Views.Count;
  ПредставленияДоступны = Истина;
Исключение
  ПредставленияДоступны = Ложь;
КонецПопытки;

Если ПредставленияДоступны Тогда
  // Выводим представления
  Для НомерПредставления = 0 По Catalog.Views.Count-1 Цикл
```

```
    ИмяПредставления = Catalog.Views.Item(НомерПредставления).Name;
    Сообщить("Представление: " + ИмяПредставления);
  КонецЦикла;
КонецЕсли;

// Выводим процедуры
Для НомерПроцедуры = 0 По Catalog.Procedures.Count-1 Цикл
  ИмяПроцедуры = Catalog.Procedures.Item(НомерПроцедуры).Name;
  Сообщить("Процедура: " + ИмяПроцедуры);
КонецЦикла;
```

В приведенном примере имеются три цикла для просмотра коллекций `Tables`, `Views` и `Procedures` объекта `Catalog`. Отметим, что перед циклом просмотра коллекции `Views` следует выполнить проверку того, доступны ли представления для текущего источника данных с помощью конструкции `Попытка...Исключение...КонецПопытки`. Это делается, чтобы избежать ошибок и исключительных ситуаций, которые могут возникнуть, если ADOX не поддерживает коллекцию `Views` для текущего источника данных.

Коллекция `Tables` содержит один или более объектов `Table`, свойствами которых являются коллекции `Columns`, `Indexes` и `Keys`, и их также следует просмотреть (листинг 5.9).

Листинг 5.9. Вывод информации о полях, индексах и ключах таблиц

```
// Начальная инициализация
Catalog = Новый COMОбъект("ADOX.Catalog");
Catalog.ActiveConnection = СтрокаПодключения;

// Выводим таблицы
Для НомерТаблицы = 0 По Catalog.Tables.Count-1 Цикл
  Table = Catalog.Tables.Item(НомерТаблицы);
  ИмяТаблицы = Table.Name;
  Сообщить("Таблица: " + ИмяТаблицы);

  // Выводим поля таблицы
  Если Table.Columns.Count > 0 Тогда
    Для НомерПоля = 0 По Table.Columns.Count-1 Цикл
      Column = Table.Columns.Item(НомерПоля);
      ИмяПоля = Column.Name;
      ТипПоля = Column.Type;
```

```
        ДлинаПоля = Column.DefinedSize;
        Сообщить("Имя поля: " + ИмяПоля);
        Сообщить("Тип поля: " + ТипПоля);
        Сообщить("Длина поля: " + ДлинаПоля);
      КонецЦикла;
    КонецЕсли;

    // Выводим индексы таблицы
    Если Table.Indexes.Count > 0 Тогда
      Для НомерИндекса = 0 По Table.Indexes.Count-1 Цикл
        Index = Table.Indexes.Item(НомерИндекса);
        ИмяИндекса = Index.Name;
        Сообщить("Имя индекса: " + ИмяИндекса);
      КонецЦикла;
    КонецЕсли;

    // Выводим ключи таблицы
    Если Table.Keys.Count > 0 Тогда
      Для НомерКлюча = 0 По Table.Keys.Count-1 Цикл
        Key = Table.Keys.Item(НомерКлюча);
        ИмяКлюча = Key.Name;
        Сообщить("Имя ключа: " + ИмяКлюча);
      КонецЦикла;
    КонецЕсли;
  КонецЦикла;
```

В данном примере в цикле выборки полей таблицы свойство `Column.Type` возвращает число, соответствующее типу, определенному в табл. 5.1. Ниже будет приведен полный список свойств объекта `Column`.

- `Attributes` — содержит характеристики поля.
- `DefinedSize` — содержит максимальный размер поля.
- `NumericScale` — содержит сведения о положении десятичной точки для числового поля.
- `ParentCatalog` — указывает на имя каталога, к которому принадлежит поле.
- `Precision` — содержит максимальную точность данных в поле.
- `RelatedColumn` — для ключевых полей содержит имя связанного поля.
- `SortOrder` — указывает порядок сортировки в данных для поля.
- `Type` — содержит тип данных, хранящихся в поле.

Объект Index содержит следующие свойства.

- Clustered — указывает, является ли индекс кластерным.
- IndexNulls — указывает, как обрабатываются значения Null.
- PrimaryKey — указывает, реализует ли данный индекс первичный ключ.
- Unique — указывает, должен ли быть уникальным ключ, реализованный в данном индексе.

И наконец, рассмотрим список свойств объекта Key.

- DeleteRule — указывает, каким образом обрабатывается удаление записи, содержащей первичный ключ.
- RelatedTable — для внешнего ключа указывает имя связанной таблицы.
- Type – содержит тип ключа.
- UpdateRule — указывает, как производится обновление записи, содержащей первичный ключ.

Создание баз данных и их объектов

Первый шаг при создании новой базы данных — создание нового экземпляра объекта Catalog. Это позволяет определить не только тип создаваемой базы данных (с помощью OLE DB-провайдера), но и местоположение файла базы данных. В листинге 5.10 показано, как это можно сделать для базы данных Microsoft Access.

Листинг 5.10. Создание новой базы данных

```
// Описание переменных подключения
ИмяФайла = "C:\demo.mdb";
СтрокаПодключения = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" +
ИмяФайла;
// Создадим экземпляр объекта ADOX Catalog
Catalog = Новый COMОбъект("ADOX.Catalog");
// Создадим новый MDB-файл
Catalog.Create(СтрокаПодключения);
```

В приведенном выше примере создается новая база данных заданного типа в заранее заданном каталоге. Затем можно добавить в эту базу данные таблиц и поля. Чтобы сделать это, необходимо:

1. Создать новый экземпляр объекта Table и добавить его в коллекцию Tables объекта Catalog.
2. Создать новый экземпляр объекта Column и добавить его в коллекцию Columns объекта Table.

Следующий пример (листинг 5.11) показывает, как можно реализовать эту последовательность действий.

Листинг 5.11. Создание новых полей в базе данных

```
// 1. Создать новый экземпляр объекта Table
Table = Новый COMОбъект("ADOX.Table");
// Имя новой таблицы
ИмяТаблицы = "NewTable";
Table.Name = ИмяТаблицы;
// Добавление новой таблицы
Catalog.Tables.Append(Table);

// 2. Создать новый экземпляр объекта Column
// Имя новой колонки
ИмяПоля = "NewColumn";
// Добавление нового поля (3-adInteger)
Catalog.Tables(ИмяТаблицы).Columns.Append(ИмяПоля, 3);
```

Теперь рассмотрим работу с индексами и ключами. Для того чтобы создать ключ, используется объект `ADOX.Key`. Под объектами `Key` подразумеваются первичные (`Primary key`) и внешние (`Foreign key`) ключи.

Приведем пример созданий внешнего ключа (листинг 5.12).

Листинг 5.12. Создание внешнего ключа

```
ForeignKey = Новый COMОбъект("ADOX.Key");
ForeignKey.Name = "CustOrder";
ForeignKey.Type = 1; // adKeyForeign

// Далее указывается ссылка на связанную таблицу
ForeignKey.RelatedTable = "Customers";
// Определяем имя ключевого поля
ForeignKey.Columns.Append("CustomerId");
// Определяем имя поля в связанной таблице
ForeignKey.Columns("CustomerId").RelatedColumn = "CustomerId";
// Определяем правила контроля за каскадными обновлениями
ForeignKey.UpdateRule = adRICascade;

// Добавляем ключ в коллекцию ключей
Catalog.Tables("NewTable").Keys.Append(ForeignKey);\
```

В данном примере свойство `ForeignKey.Type` может принимать следующие значения:

- `1` (`adKeyPrimary`) — первичный ключ;
- `2` (`adKeyForeign`) — внешний ключ;
- `3` (`adKeyUnique`) — уникальный ключ.

Свойство `ForeignKey.UpdateRule` может принимать следующие значения:

- `0` (`adRINone`) — никаких действий каскадно не производится (используется по умолчанию);
- `1` (`adRICascade`) — произвести каскадное обновление поля;
- `2` (`adRISetNull`) — значение внешнего ключевого поля выставляется в `Null`;
- `3` (`adRISetDefault`) — значение внешнего ключевого поля выставляется в значение по умолчанию.

Подобным же образом осуществляется работа с управлением индексами.

Примечание

Можно особо отметить следующую особенность — у таблицы есть коллекция индексов, а уже у каждого индекса есть коллекция полей, входящих в этот индекс. Для простых индексов, состоящих из одного поля, в коллекции колонок содержится одна колонка, для составных индексов в коллекции колонок индекса содержатся колонки, входящие в этот индекс.

Приведем пример добавления к таблице составного индекса, состоящего из двух полей: `Field1` и `Filed2` (листинг 5.13).

Листинг 5.13. Создание составного индекса

```
Index = Новый COMОбъект("ADOX.Index");

// Зададим параметры индекса
Index.Name = "Index1"
Index.Columns.Append("Field1");
Index.Columns.Append("Field2");

// Добавляем индекс в коллекцию
Catalog.Tables("NewTable").Indexes.Append(Index);
```

Удаление объектов базы данных

Для удаления объектов базы данных используется метод `Delete`. Приведем пример удаления индекса, ключа, поля и таблицы (листинг 5.14).

Листинг 5.14. Удаление объектов базы данных

```
ИмяТаблицы = "NewTable";
// Удаление индекса
Catalog.Tables(ИмяТаблицы).Indexes.Delete(ИмяИндекса);
// Удаление ключа
Catalog.Tables(ИмяТаблицы).Keys.Delete(ИмяКлюча);
// Удаление поля
Catalog.Tables(ИмяТаблицы).Columns.Delete(ИмяПоля);
// Удаление таблицы
Попытка
  Catalog.Tables.Delete(ИмяТаблицы);
Исключение
  Сообщить(ОписаниеОшибки());
  Возврат;
КонецПопытки;
```

В данном примере удаление таблицы взято в операторные скобки `Попытка...Исключение...КонецПопытки` для исключения системных ошибок, связанных с невозможностью удаления таблицы. Подобная ошибка может появиться при попытке удалить таблицу, содержащую внешний ключ. Вообще говоря, удаление любых объектов баз данных рекомендуется обрамлять конструкцией `Попытка...Исключение...КонецПопытки`.

Доступ к данным Microsoft Excel через OLE DB

Достаточно часто требуется сформировать отчет и вывести его не только на печать, но и выгрузить в таблицу Microsoft Excel. Также иногда возникает необходимость импортировать данные Excel в систему "1С:Предприятие". Этого можно добиться двумя способами, либо используя технологию OLE Automation, описание и примеры использования которой рассматривались в *главе 4*, либо через объекты ADO.

Использование технологии ADO, при работе с таблицами Microsoft Excel, практически ничем не отличается от доступа к другим источникам данных. Для подключения к Excel можно использовать код:

```
Provider=Microsoft.Jet.OLEDB.4.0;
Data Source=C:\file.xls;
Extended Properties="Excel 8.0;HDR=NO;".
```

Параметр HDR здесь обозначает наличие заголовка в первой строке таблицы, причем YES — определяет присутствует заголовока; NO — нет.

Для выгрузки данных в таблицы Microsoft Excel необходимо создать файл с расширением XLS. Это можно сделать вручную, либо программно с помощью технологии ADOX. Рассмотрим пример создания файла Microsoft Excel через объекты ADOX (листинг 5.15).

Листинг 5.15. Создание MS Excel файла средствами ADOX

```
// Определение имени файла и строки соединения
ИмяФайла = "C:\ExcelDemoFile.xls";
СтрокаПодключения = "Provider=Microsoft.Jet.OLEDB.4.0; Data Source=" +
ИмяФайла + "; Extended Properties=""Excel 8.0;HDR=NO;""";

// Начальная инициализация
Catalog = Новый COMОбъект("ADOX.Catalog");
Catalog.ActiveConnection = СтрокаПодключения;

// Создание новой таблицы
Table = Новый COMОбъект("ADOX.Table");
// Имя таблицы
Table.Name = "TestTable";

Для Ном = 1 По 5 Цикл
  // Создание новой колонки
  Column = Новый COMОбъект("ADOX.Column");
  // Имя колонки
  Column.Name = "Col" + Ном;
  // Тип данных колонки
  Если Ном = 2 Тогда
    Column.Type = 202; // adVarWChar
  Иначе
    Column.Type = 5; // adDouble
  КонецЕсли;

  // Присоединение колонки к таблице
  Table.Columns.Append(Column);
  Column = Неопределено;
КонецЦикла;

// Присоединение созданной таблицы (листа) к книге Excel
Catalog.Tables.Append(Table);
```

В данном примере создается файл C:\ExcelDemoFile.xls, в который добавляется лист с именем `TestTable` и инициализируются пять колонок. Свойство `Type` объекта `Column` определяет тип колонки. Для источника данных Microsoft Excel это свойство может принимать всего шесть значений, описанных в табл. 5.3.

***Таблица 5.3.** Допустимые значения свойства `Type` объекта `Column` для* MS Excel

Константа ADO	Значение	Описание
`adDouble`	5	Значение с плавающей точкой двойной точности
`adDate`	7	Дата
`adCurrency`	6	Денежная сумма (8-байтовое целое со знаком, кратное 10 000)
`adBoolean`	11	Булево значение
`adVarWChar`	202	Символьная строка Unicode, заканчивающаяся символом `Null`
`adLongVarWChar`	203	Длинное строковое значение

После того как файл Excel создан, можно выполнять действия по добавлению данных в какой-либо лист книги Excel. Данные действия можно выполнять двумя способами, либо через метод `AddNew` объекта `Recordset` (листинг 5.16), либо через SQL-запросы, используя конструкцию `INSERT` (листинг 5.17).

Листинг 5.16. Запись данных в лист MS Excel средствами ADO

```
// Создание соединения ADO для работы с книгой Excel
ConnectionExcel = Новый COMОбъект("ADODB.Connection");
ConnectionExcel.Open(СтрокаПодключения);

// Создание объекта набора записей
RecordsetExcel = Новый COMОбъект("ADODB.Recordset");
RecordsetExcel.Open("Select * from TestTable", ConnectionExcel, 1, 3);

Для Ном = 1 По 10 Цикл
  // Добавление новой записи
  RecordsetExcel.AddNew();
  RecordsetExcel.Fields(0).Value = 11;
  RecordsetExcel.Fields(1).Value = "Строка №" + Ном;
```

```
    RecordsetExcel.Fields(2).Value = 2.30;
    RecordsetExcel.Fields(3).Value = Ном;
    RecordsetExcel.Fields(4).Value = Ном * 2.2;

    // Обновление изменений в книге Excel
    RecordsetExcel.UpDate();
КонецЦикла;

// Закрываем соединение
ConnectionExcel.Close();
ConnectionExcel = Неопределено;
```

В данном примере метод `AddNew` добавляет очередную строку в лист Excel. Важно отметить, что в один лист невозможно добавить более 65 000 строк — это ограничение приложения Microsoft Excel. Еще одна важная особенность добавления информации в ячейки листа заключается в том, что Excel не поддерживает автопреобразование типов значений ячеек. То есть, если с помощью свойства `Type` объекта `Column` для колонки соответствующей ячейки был определен числовой тип, то при попытке записать в эту ячейку строку — возникнет ошибка. Так же нельзя использовать для записи колонки, которые не были созданы, т. е. если было создано всего пять колонок, то можно записывать информацию только в первые пять колонок, хотя при открытии таблицы в приложении Microsoft Excel пользователю будут видны большое количество колонок.

Рассмотрим пример добавления записей, используя SQL-запросы (листинг 5.17).

Листинг 5.17. Запись данных в лист MS Excel через SQL-запросы

```
// Создание объекта установки связи с источником данных
Connection = Новый COMОбъект("ADODB.Connection");

// Подключение к источнику данных
Попытка
    Connection.Open(СтрокаПодключения);
Исключение
    Сообщить(ОписаниеОшибки());
    Возврат;
КонецПопытки;

// Создание объекта выполнения команды
```

```
Command = Новый COMОбъект("ADODB.Command");
// Указание активного соединения
Command.ActiveConnection = Connection;
// Определение текста команды
Command.CommandText = "INSERT INTO TestTable VALUES(20, 'Новая строка',
33.3, 44.4, 55.5)";
// Определение типа команды
Command.CommandType = 1;
// Выполнение запроса
Command.Execute();

// Закрываем соединение
Connection.Close();
Command = Неопределено;
```

Данный пример аналогичен примеру, приведенному в листинге 5.4, за исключением того, что в конструкции запроса `INSERT` нельзя указать имена колонок.

Для чтения данных из листа Excel необходимо выполнить ряд стандартных действий. Пример такой выборки приведен в листинге 5.18.

Листинг 5.18. Чтение данных MS Excel средствами ADO

```
// Чтение данных
// Создание объекта установки связи с источником данных
Connection = Новый COMОбъект("ADODB.Connection");

// Подключение к источнику данных
Попытка
  Connection.Open(СтрокаПодключения);
Исключение
  Сообщить(ОписаниеОшибки());
  Возврат;
КонецПопытки;

// Создание объекта выполнения команды
Command = Новый COMОбъект("ADODB.Command");
// Указание активного соединения
Command.ActiveConnection = Connection;
```

```
// Определение текста команды
Command.CommandText = "SELECT * FROM TestTable";
// Определение типа команды
Command.CommandType = 1;

// Создание объекта набора записей
RecordSet = Новый COMОбъект("ADODB.RecordSet");
// Выполнение и получение набора данных
RecordSet = Command.Execute();

// Перебор данных
Пока RecordSet.EOF() = 0 Цикл
  Сообщить("Строка: " + Recordset.Fields(1).Value);
  Сообщить("Число: " + Recordset.Fields(4).Value);
  RecordSet.MoveNext();
КонецЦикла;

// После того как набор записей уже не нужен, его следует закрыть
RecordSet.Close();
Connection.Close();
```

Доступ к данным Microsoft Project через OLE DB

Для доступа к данным Microsoft Project можно использовать объекты ADO. Общие принципы, применяемые для доступа к данным MS Project, такие же, что и для других источников данных, но есть некоторые особенности и ограничения.

Некоторые аспекты использования средства доступа OLE DB для Microsoft Project уникальны и должны быть приняты во внимание во избежание нежелательных результатов.

- Существует четыре типа данных:
  - `text` — текстовый;
  - `number` — числовой;
  - `boolean` — логический;
  - `date` — дата-время (в том виде, как они отображаются в интерфейсе, например 27.12.1999 10:30).

- Поля длительности возвращают число минут, умноженное на 10 (например, 8 часов — это 4800), а поля трудозатрат — число минут, умноженное на 1000 (8 часам соответствует 480 000).
- Даты, отображающиеся в интерфейсе как NA (нет данных), возвращаются как значение `0`.
- Формула в настраиваемом поле, которая в интерфейсе привела бы к результату `#ERROR`, возвращает значение по умолчанию для данного поля.
- Настраиваемое поле, для которого не установлено никакого значения, возвращает значение, принимаемое по умолчанию.
- Поле индикатора настраиваемого поля в случае, когда индикатор не установлен, возвращает значение `-1`.
- Значения трудозатрат для материальных ресурсов выражаются в единицах, определенных в интерфейсе, а не числом минут, умноженным на 1000.
- Оператор `SELECT` без предложения `WHERE` возвращает пустые строки таблиц ресурсов и задач. Если задать предложение `WHERE`, эти строки уже не будут возвращены, даже если они подходят по всем остальным условиям.

В реализации средства доступа OLE DB для Microsoft Project 2002 действует ряд ограничений.

- Не поддерживается доступ для чтения и записи.
- Не поддерживаются запросы сразу к нескольким таблицам. Для каждой таблицы, к которой требуется доступ, необходимо использовать отдельный запрос.
- Доступ к OLE DB осуществляется с помощью курсоров (наборов записей) последовательного доступа. Наборы записей с последовательным доступом не поддерживают такие методы, как `MovePrevious`, `MoveFirst` или `MoveLast`. Кроме того, наборы записей с последовательным доступом не поддерживают использование свойства `RecordCount`.
- Объединения не поддерживаются, однако аналогичные возможности предоставляют сформированные наборы записей, устанавливающие ранее не существовавшие отношения между ключами, полями или наборами строк. Можно также создавать иерархические наборы записей в табличном формате.
- Не поддерживаются операторы — `ANY`, `LIKE` и `IS NOT`.
- Не поддерживаются статистические функции — `Sum`, `Avg`, `Min`, `Max`, `Count` и `StDev`.
- Средство доступа OLE DB снабжено механизмом контроля времени, который сигнализирует, когда следует выгружать файлы, но делает это не раньше, чем произойдет событие загрузки. Если в поле `TimeBeforeUnload` установлено значение `1`, то не проверяется — имеется ли открытый файл,

до попытки открыть другой. В итоге этот файл блокируется для чтения. Во избежание возникновения данной ситуации надо создать фиктивный файл, чтобы заставить средство доступа, после завершения работы над текущим проектом, загружать некий несуществующий файл. Таким образом, файл, который был в работе, будет выгружен, а блокировка чтения будет установлена на фиктивный файл.

Среда ADO обеспечивает доступ к OLE DB с использованием набора объектов, событий, методов и свойств. Возможны два способа использования ADO:

- доступ к поставщику данных на локальном компьютере;
- доступ к поставщику с сервера Microsoft Project Server.

В листинге 5.19 показан доступ к файлу Microsoft Project на локальном компьютере, а также отображаются некоторые сведения о проекте.

Листинг 5.19. Подключение к проекту и вывод его свойств

```
// Определение параметров соединения
Провайдер = "Microsoft.Project.OLEDB.10.0";
ИмяФайла = "C:\example.mpp";
// Если файл не защищен, то в строке подключения параметры
// User ID (пользователь) и Password (пароль) указывать не обязательно
Пользователь = "";
Пароль = "";

// Установка соединения с проектом
Попытка
  Connection = Новый COMОбъект("ADODB.Connection");
  Connection.Open("Provider=" + СокрЛП(Провайдер) +
                  ";Project Name=" + СокрЛП(ИмяФайла) +
                  ";User ID=" + СокрЛП(Пользователь) +
                  ";Password=" + СокрЛП(Пароль));
Исключение
  Сообщить(ОписаниеОшибки());
  Возврат;
КонецПопытки;

// Запрос к проекту
Command = Новый COMОбъект("ADODB.Command");
Command.ActiveConnection = Connection;
Command.CommandText = "SELECT * FROM Project";
```

```
RecordSet = Новый COMОбъект("ADODB.RecordSet");
RecordSet = Command.Execute();

// Получение свойств проекта
Название = RecordSet.Fields("ProjectTitle").Value;
ДатаНачала = RecordSet.Fields("ProjectStartDate").Value;
ДатаКонца = RecordSet.Fields("ProjectFinishDate").Value;

// Вывод свойств проекта
Сообщить("Название проекта: " + Название);
Сообщить("Дата начала: " + ДатаНачала);
Сообщить("Дата завершения: " + ДатаКонца);
```

В данном примере происходит подключение к файлу проекта C:\example.mpp и выводится название проекта, даты начала и завершения проекта из таблицы `Project` (проект). Эта таблица обеспечивает доступ к параметрам уровня проекта.

Таблица `Project` имеет большое количество полей. Основные из них приведены в табл. 5.4.

Таблица 5.4. *Основные поля таблицы* `Project`

Имя поля	Тип данных	Описание
`Project` (Проект)	`text`	Название проекта, отображаемое как путь к его расположению на диске, например C:\example.mpp
`ProjectUniqueID` (Уникальный идентификатор проекта)	`number`	Уникальный идентификатор проекта
`ProjectAuthor` (Автор)	`text`	Имя автора проекта. Используется для группировки аналогичных проектов
`ProjectCompany` (Организация)	`text`	Название организации, создавшей проект; используется для группировки аналогичных проектов
`ProjectCreationDate` (Дата создания)	`date`	Дата создания проекта
`ProjectStartDate` (Дата начала)	`date`	Дата и время запланированного начала проекта
`ProjectFinishDate` (Дата окончания)	`date`	Дата и время запланированного завершения проекта

Таблица 5.4 (окончание)

Имя поля	Тип данных	Описание
ProjectLastSaved (Дата сохранения)	date	Дата последнего сохранения проекта
ProjectManager (Руководитель)	text	Руководитель проекта. Используется для группировки проектов, имеющих одного руководителя
ProjectRevision (Версия)	text	Текущий номер версии файла проекта
ProjectSubject (Тема)	text	Тема проекта. Используется для группировки аналогичных проектов
ProjectTitle (Название)	text	Название проекта. Используется для группировки аналогичных проектов

Все остальные поля таблицы `Project` и их типы можно вывести с помощью объектов ADOX (листинг 5.20).

Листинг 5.20. Вывод всех полей таблицы `Project`

```
// Определение параметров соединения
Провайдер = "Microsoft.Project.OLEDB.10.0";
ИмяФайла = "C:\example.mpp";

// Начальная инициализация
Catalog = Новый COMОбъект("ADOX.Catalog");
Catalog.ActiveConnection = "Provider=" + СокрЛП(Провайдер) + ";Project
Name=" + СокрЛП(ИмяФайла);

Table = Catalog.Tables.Item("Project");
// Выводим поля таблицы
Если Table.Columns.Count > 0 Тогда
  Для НомерПоля = 0 По Table.Columns.Count-1 Цикл
    Column = Table.Columns.Item(НомерПоля);
    ИмяПоля = Column.Name;
    ТипПоля = Column.Type;
    ДлинаПоля = Column.DefinedSize;
    Сообщить("Имя поля: " + ИмяПоля);
    Сообщить("Тип поля: " + ТипПоля);
```

```
    Сообщить("Длина поля: " + ДлинаПоля);
  КонецЦикла;
КонецЕсли;
```

Основная таблица, в которой хранятся сведения о задачах, включенных в проект, имеет идентификатор — `Tasks`, основные поля которой перечислены в табл. 5.5.

Таблица 5.5. *Основные поля таблицы* `Tasks`

Имя поля	Тип данных	Описание
`TaskUniqueID` (Уникальный идентификатор)	`number`	Уникальный идентификатор задачи
`TaskActualStart` (Фактическое начало)	`date`	Дата и время фактического начала задачи
`TaskActualFinish` (Фактическое окончание)	`date`	Дата и время фактического завершения задачи
`TaskActualWork` (Фактические трудозатраты)	`number`	Объем трудозатрат, которые уже выполнены назначенными задаче ресурсами
`TaskStart` (Начало)	`date`	Дата и время запланированного начала выполнения задачи вычисляется автоматически, если у задачи есть предшественник. `TaskFinish` — дата и время запланированного завершения выполнения задачи
`TaskContact` (Контактное лицо)	`text`	Имя человека, являющегося ответственным за выполнение задачи
`TaskName` (Название)	`text`	Название задачи
`TaskNotes` (Заметки)	`text`	Введенные заметки о задаче
`TaskOutlineLevel` (Уровень структуры)	`number`	Число, определяющее уровень задачи в иерархической структуре проекта
`TaskOutlineNumber` (Номер в структуре)	`text`	Точная позиция задачи в структуре. Например, номер 7.2 означает, что данная задача является второй подзадачей седьмой суммарной задачи верхнего уровня
`TaskPercentComplete` (Процент завершения)	`number`	Текущее состояние задачи, выраженное в виде завершенной части работы в процентах по отношению к общей длительности задачи

К задачам обычно привязан один или несколько ресурсов. В таблице Assignments содержатся данные о необходимых ресурсах для выполнения всех задач. Основные поля таблицы Assignments приведены в табл. 5.6.

Таблица 5.6. *Основные поля таблицы* Assignments

Имя поля	Тип данных	Описание
TaskUniqueID (Уникальный идентификатор задачи)	number	Указатель на действительный идентификатор в таблице Tasks
AssignmentUniqueID (Уникальный идентификатор)	number	Уникальный идентификатор назначения
AssignmentResourceID (Идентификатор ресурса)	number	Указатель на действительный идентификатор в таблице Resources
AssignmentActualStart (Фактическое начало)	date	Дата и время фактического начала назначения
AssignmentActualFinish (Фактическое окончание)	date	Дата и время фактического завершения назначения
AssignmentActualWork (Фактические трудозатраты)	number	Объем работы, которая уже выполнена ресурсом задачи
AssignmentStart (Начало)	date	Запланированная дата и время начала работы назначенного ресурса
AssignmentFinish (Окончание)	date	Запланированная дата и время завершения использования ресурса по задаче
AssignmentFinishVariance (Отклонение окончания)	number	Разница между датой окончания назначения по базовому плану и запланированной датой окончания
AssignmentNotes (Заметки)	text	Заметки о назначении
AssignmentUnits (Единицы)	number	Число единиц, на которое ресурс назначается задаче. Выражается в процентах от максимального числа единиц ресурса
AssignmentWork (Трудозатраты)	number	Общий запланированный объем трудозатрат, которые ресурс должен выполнить по задаче
AssignmentResourceName (Название ресурса)	text	Название ресурса, связанного с назначением
AssignmentResourceType (Тип ресурса)	number	Тип ресурса: 0 — трудовой (люди и оборудование, назначается по умолчанию); 1 — материальный (расходные материалы, например, сталь, бетон или грунт)

Рассмотрим пример вывода дерева задач с указанием использованных ресурсов для системы "1С:Предприятие" версии 8.0 (листинг 5.21).

Листинг 5.21. Вывод дерева задач с указанием использованных ресурсов для версии 8.0

```
// Переводит номер задачи из формата xxx.xx.x в xxx.xx
Функция ПолучитьКодРодителя(Знач НомерЗадачи)
  ЧислоВхождений = СтрЧислоВхождений(НомерЗадачи, ".");
  Поз = 0;
  Если ЧислоВхождений = 0 Тогда
    Возврат "0";
  Иначе
    Для Ном = 1 По ЧислоВхождений Цикл
      Поз = Поз + Найти(Сред(НомерЗадачи, Поз + 1), ".");
    КонецЦикла;
    Возврат Лев(НомерЗадачи, Поз - 1);
  КонецЕсли;
КонецФункции

// Формирует дерево задач
Процедура СформироватьДерево()
  Провайдер = "Microsoft.Project.OLEDB.10.0";
  ИмяФайла = "C:\example.mpp";
  Пароль = "";

  Состояние("Соединение с проектом. Пожалуйста ждите...");
  Попытка
    Connection = Новый COMОбъект("ADODB.Connection");
    Connection.Open("Provider=" + СокрЛП(Провайдер) +
                    ";Project Name=" + СокрЛП(ИмяФайла) +
                    ";Password=" + СокрЛП(Пароль));
  Исключение
    Сообщить(ОписаниеОшибки());
    Возврат;
  КонецПопытки;

  // Запрос к задачам
  Command = Новый COMОбъект("ADODB.Command");
```

```
  Command.ActiveConnection = Connection;
  Command.CommandText = "SELECT
TaskName,TaskType,TaskUniqueID,TaskOutlineLevel,TaskOutlineNumber,
TaskStart,TaskFinish,TaskPercentComplete FROM Tasks";

  RecordSet = Новый COMОбъект("ADODB.RecordSet");
  RecordSet = Command.Execute();

  // Перебор данных
  Пока RecordSet.EOF() = 0 Цикл
    // Тип записи: 1-группа, 0-задача
    ТипЗадачи = RecordSet.Fields("TaskType").Value;
    // Имя задачи или группы
    ИмяЗадачи = RecordSet.Fields("TaskName").Value;
    // Уникальный идентификатор задачи или группы
    ИдентификаторЗадачи = RecordSet.Fields("TaskUniqueID").Value;
    // Уровень задачи
    УровеньЗадачи = RecordSet.Fields("TaskOutlineLevel").Value;
    // Номер задачи в формате xx.x.x
    НомерЗадачи = RecordSet.Fields("TaskOutlineNumber").Value;
    ДатаНачала = RecordSet.Fields("TaskStart").Value;
    ДатаКонца = RecordSet.Fields("TaskFinish").Value;
    ПроцентВыполнения = RecordSet.Fields("TaskPercentComplete").Value;

    Если НомерЗадачи = 0 Тогда
      СтрокаТПБазовая = ТабличноеПоле;
    Иначе
      // Поиск родительской задачи
      Результат =
      ТабличноеПоле.Строки.Найти(ПолучитьКодРодителя(НомерЗадачи),
      "Ид", Истина);
      Если Результат = Неопределено Тогда
        СтрокаТПБазовая = ТабличноеПоле;
      Иначе
        СтрокаТПБазовая = Результат;
      КонецЕсли;
    КонецЕсли;

    // Добавляем параметры задачи в дерево
    СтрокаТП = СтрокаТПБазовая.Строки.Добавить();
```

```
    СтрокаТП.Ид = НомерЗадачи;
    СтрокаТП.Задача = ИмяЗадачи;
    СтрокаТП.ДатаНачала = ДатаНачала;
    СтрокаТП.ДатаКонца = ДатаКонца;

    Ресурсы = "";
    Если ТипЗадачи = 0 Тогда // Это задача
      // Запрос по ресурсам
      Command = Новый COMОбъект("ADODB.Command");
      Command.ActiveConnection = Connection;
      Command.CommandText = "SELECT AssignmentUnits,
      AssignmentResourceName FROM Assignments WHERE AssignmentResourceID >
      0 AND TaskUniqueID = " + ИдентификаторЗадачи;

      RecordSetResource = Новый COMОбъект("ADODB.RecordSet");
      RecordSetResource = Command.Execute();

      // Перебор данных
      Пока RecordSetResource.EOF() = 0 Цикл
        ИмяРесурса =
        RecordSetResource.Fields("AssignmentResourceName"). Value;
        ПроцентЗагрузки =
        RecordSetResource.Fields("AssignmentUnits"). Value * 100;
        Ресурсы = Ресурсы + ?(Ресурсы="","","; ") +
        СокрЛП(ИмяРесурса) + " (" + ПроцентЗагрузки + "%)";
        RecordSetResource.MoveNext();
      КонецЦикла;

      RecordSetResource.Close();
    КонецЕсли;
    СтрокаТП.Ресурсы = Ресурсы;
    // Переход к следующей задачи
    RecordSet.MoveNext();
  КонецЦикла;
  RecordSet.Close();
КонецПроцедуры
```

В данном примере подразумевается, что на форме обработки имеется элемент — табличное поле с идентификатором `ТабличноеПоле`. Это табличное поле имеет четыре созданных на форме текстовых поля:

- Ид — скрытое поле, предназначенное для хранения идентификатора задачи;

- `Задача` — название задачи;
- `ДатаНачала` — запланированная дата начала выполнения задачи;
- `ДатаКонца` — запланированная дата окончания задачи;
- `Ресурсы` — строка, содержащая список используемых ресурсов и их загрузку.

Результат вывода дерева задач произвольного проекта показан на рис. 5.2.

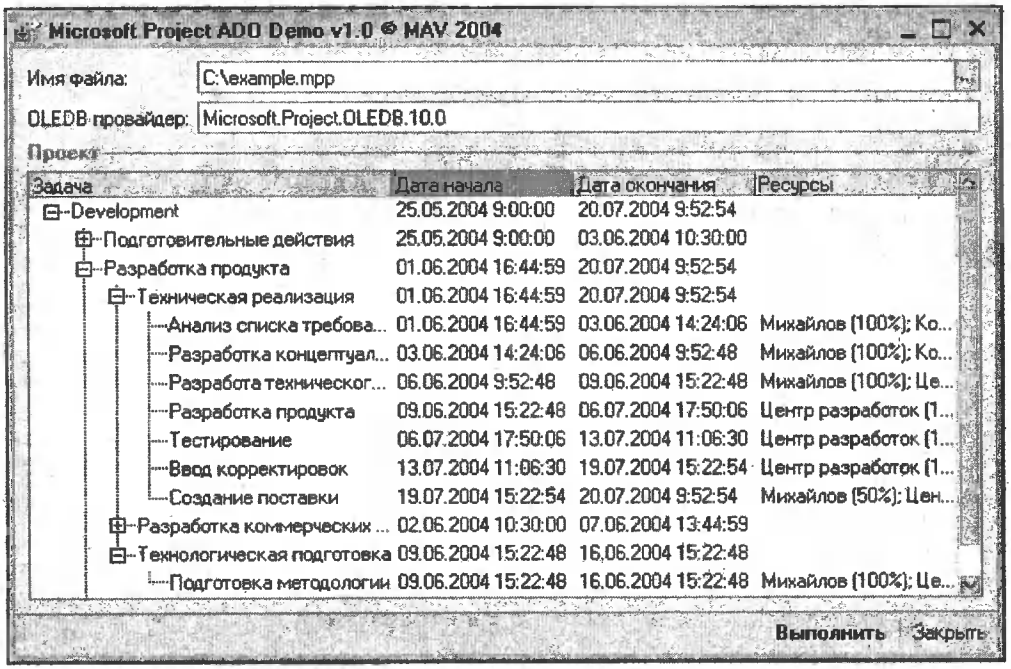


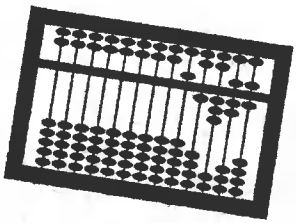
Рис. 5.2. Результат вывода дерева задач проекта

Помимо основных, ранее описанных таблиц (`Project`, `Tasks` и `Assignments`), существуют и другие, не менее важные таблицы, подробное описание которых в данной книге приводиться не будет. Рассмотрим их состав и назначение.

- `Availability` — таблица обычно используется вместе с таблицей для получения сведений о доступности ресурсов.
- `BaselineTaskSplits` — в этой таблице сохраняются базовые сведения о прерывании для конкретной задачи.
- `CalendarData` и `CalendarExceptions` — в этих таблицах хранятся все календарные данные, имеющиеся в средстве доступа Microsoft Project OLE DB.
- `Calendars` — календари, используются для определения стандартного рабочего и нерабочего времени. В проекте необходимо иметь один базовый

календарь. У задач и ресурсов могут быть свои собственные календари, но все они должны строиться на основе базового календаря. В этой таблице хранятся основные календарные данные.

- `CostRates` — таблица обычно используется вместе с таблицей `Resources` для отображения содержимого таблиц норм затрат, относящихся к ресурсу. Кроме того, данную таблицу можно использовать совместно с таблицей `Assignments` для получения сведений о таблице норм затрат, используемой назначением.
- `CustomFieldGraphicalIndicators` — таблица обычно используется вместе с таблицей `CustomFields` для получения параметров настраиваемых полей, для которых предусмотрены графические индикаторы.
- `CustomFields` (настраиваемые поля) — данная таблица используется для получения всех параметров настраиваемого поля. Для изменения этих полей необходимо извлечь глобальный корпоративный шаблон.
- `CustomFieldValueList` — эта таблица используется для получения значений таблицы подстановок для настраиваемых полей, имеющих список значений.
- `CustomOutlineCodeFields` (поля кодов структуры) — в этой таблице содержится маска для каждой таблицы подстановки настраиваемого кода структуры.
- `CustomOutlineCodeLookupTables` (таблицы подстановки) — таблица данного типа используется для получения списка значений таблиц подстановки, связанных с полями настраиваемых кодов структуры.
- `Predecessors` (предшественники) — таблица обычно используется вместе с таблицей `Tasks` для отображения подробных сведений о задачах-предшественниках.
- `Resources` — в этой таблице содержатся сведения, относящиеся к ресурсам.
- `Successors` — таблица обычно используется вместе с таблицей `Tasks` для отображения подробных сведений о задачах-последователях.
- `TaskSplits` — в этой таблице хранятся даты начала и окончания прерывания задачи.
- `WBS` (структурная декомпозиция работ) — в этой таблице хранятся определения кода СДР, а также параметры кода СДР для проекта.



Глава 6

Использование SQL-DMO

Понятие SQL-DMO

SQL-DMO (SQL Distributed Management Objects) — это предоставляемая Microsoft SQL Server объектная модель, которая основана на COM-технологии.

SQL-DMO скрывает детали структуры языка Transact-SQL и используется при написании административных приложений и сценариев для Microsoft SQL Server. Поставляемые с Microsoft SQL Server графические средства администрирования написаны с применением именно SQL-DMO, которая не является моделью интерфейса данных и не применяется для написания стандартных приложений баз данных.

SQL-DMO позволяет клиентскому приложению манипулировать такими объектами базы данных, как таблицы, процедуры и свойства сервера через интерфейс COM. Модель предоставляет приложению следующие возможности:

- управление таблицами (просмотр, создание, удаление, модификация);
- управление списком доступных серверов;
- управление правами доступа к серверам;
- управление списком баз данных на каждом сервере;
- управление сценариями таблиц базы данных;
- создание SQL-скриптов (таблиц, представлений, хранимых процедур, пользователей и их ролей);
- выполнение SQL-запросов.

Для того чтобы объекты SQL-DMO можно было использовать на конкретном компьютере, необходимо установить на нем клиентскую часть Microsoft SQL Server, который устанавливает библиотеку Microsoft SQL OLE Object Library. Данная библиотека представляет собой ActiveX-интерфейс к объектам SQL-DMO. Посредством SQL-DMO можно получать данные из таблиц SQL Server.

Особенно эффективно данную модель могут применять разработчики, использующие Microsoft SQL Server Desktop Engine (MSDE), который поставляется вместе с Access 2000. Этот сервер баз данных представляет собой промежуточный вариант SQL Server и был создан специально для сопряжения с более ранними версиями SQL Server. Хотя MSDE обладает меньшими возможностями, чем SQL Server, он поставляется бесплатно в составе Microsoft Office 2000. Однако версия MSDE, которая поставляется вместе с Microsoft Office 2000, не включает Enterprise Manager. В результате пользователи MSDE не могут оценить все преимущества графического интерфейса при управлении объектами базы данных.

В этой главе будут рассмотрены некоторые подходы к организации работы с объектами SQL Server (далее — SQL-сервер) при помощи SQL-DMO.

Получение списка доступных серверов

Для получения атрибутов SQL-сервера предназначен объект `SQLDMO.SQLServer2`. В листинге 6.1 приведен пример вывода списка доступных серверов.

Листинг 6.1. Получение списка доступных серверов

```
Попытка
  // Создаем объект SQLServer2
  // SQLServer2 - предназначен для получения атрибутов MS SQL Server
  oSQLServer2 = СоздатьОбъект("SQLDMO.SQLServer2");
Исключение
  Сообщить(ОписаниеОшибки());
  Возврат;
КонецПопытки;

oSQLServer2.LoginTimeout = 10;
oSQLServer2.ODBCPrefix = 0;

// Вывод имен общих SQL-серверов
ServerNameList = oSQLServer2.Application.ListAvailableSQLServers();
Для Ном = 1 По ServerNameList.Count Цикл
  Сообщить(ServerNameList.Item(Ном));
КонецЦикла;

// Вывод локальных общих SQL-серверов
InstanceNameList = oSQLServer2.ListInstalledInstances();
```

```
Для Ном = 1 По InstanceNameList.Count Цикл
  Сообщить(InstanceNameList.Item(Ном));
КонецЦикла;
```

В приведенном примере свойство `oSQLServer2.LoginTimeout` предназначено для определения числа секунд, во время которого будут происходить попытки подключиться. Свойство `oSQLServer2.ODBCPrefix` управляет выводом ошибок.

Перечисление спецификаций баз данных

Для того чтобы работать с объектами SQL-DMO, необходимо подключиться к выбранному серверу (листинг 6.2).

Листинг 6.2. Подключение к объекту `SQLDMO.SQLServer`

```
// Создаем экземпляр объекта сервера и подсоединиться к нему
SQLServer  =  СоздатьОбъект("SQLDMO.SQLServer");
SQLServer.LoginTimeout = -1;

Попытка
  Если ИспользоватьNTАвторизацию = 1  Тогда
    // Использовать авторизацию WinNT
    SQLServer.LoginSecure = 1;
    // При разрыве соединения - автоматически повторно не соединяться
    SQLServer.AutoReConnect = 0;
    // Подключение к серверу
    SQLServer.Connect(ИмяСервера);
  Иначе
    //  Использовать SQL Server авторизацию
    SQLServer.LoginSecure = 0;
    // При разрыве соединения - автоматически повторно не соединяться
    SQLServer.AutoReConnect = 0;
    //  Подключиться с использованием SQL Security
    SQLServer.Connect(ИмяСервера, СокрЛП(Пользователь), СокрЛП(Пароль));
  КонецЕсли;
Исключение
  Сообщить(ОписаниеОшибки());
  Возврат;
КонецПопытки;
```

В приведенном примере после объявления объекта SQLServer, из библиотеки SQL-DMO, программа генерирует экземпляр этого объекта и подсоединяет его к серверу ИмяСервера. В листинге 6.2 приведена программа, реализующая два варианта подключения к SQL-серверу — с помощью NT авторизации, и без нее. Для соединения с SQL-сервером используется метод Connect, в который передается имя сервера, а в случае подключения с помощью SQL Security, еще имя и пароль пользователя. Как правило, при разработке с применением SQL-DMO необходимо пользоваться именем пользователя и паролем, которые предоставляют широкие полномочия, поскольку SQL-DMO используется для программирования административных функций.

При определении времени подключения к серверу в свойстве SQLServer.LoginTimeout установлено значение -1, которое обозначает стандартное время завершения по истечении 60 секунд.

После того как произошло соединение, можно вывести спецификации всех существующих на сервере баз данных (листинг 6.3).

Листинг 6.3. Перечисление спецификаций баз данных

```
Для iCount = 1 По SQLServer.Databases.Count Цикл
  // Выводим только НЕ системные БД
  Если SQLServer.Databases.Item(iCount).SystemObject = 0 Тогда
       БазаДанных = SQLServer.Databases.Item(iCount);
    Сообщить("База данных:" + БазаДанных.Name);
    Сообщить("Дата создания:" + БазаДанных.CreateDate);
    Сообщить("Количество таблиц:" + БазаДанных.Tables.Count);
    Сообщить("Количество представлений:" + БазаДанных.Views.Count);
    Сообщить("Количество процедур:" + БазаДанных.StoredProcedures.Count);
  КонецЕсли;
КонецЦикла;
```

Приведенный код выводит список, в котором для каждой базы данных указывается дата ее создания, а также количество имеющихся в ней таблиц, представлений и хранимых процедур. После подключения к SQL-серверу в цикле примера просматриваются все базы данных и выводится сводная информация о них.

Получение списка таблиц и спецификаций полей

Чтобы получить доступ к спецификации каждой таблицы базы данных, необходимо создать объект SQLDMO.Table, который позволяет манипулиро-

вать со структурой конкретной таблицы. Элементы типа `Table` содержатся в коллекции `Database.Tables`. Для того чтобы перебрать все элементы коллекции, существует два способа. Первый — в цикле, с помощью метода `Item`, получать элементы коллекции, указывая в параметре порядковый номер таблицы (листинг 6.4). Свойство `Database.Tables.Count` возвращает количество таблиц в базе данных. Второй — с помощью конструкции `Для каждого...из...Цикл...КонецЦикла`, которую можно использовать только в версии 8.0 системы "1С:Предприятие".

Листинг 6.4. Получение списка таблиц

```
// Объект базы данных
Database = СоздатьОбъект("SQLDMO.Database");
Database = SQLServer.Databases(ИмяБазыДаных);

Для Ном = 1 По Database.Tables.Count Цикл
  // Выводим только НЕ системные таблицы
  Если Database.Tables.Item(Ном).SystemObject = 0 Тогда
    Сообщить(Database.Tables.Item(Ном).Name);
  КонецЕсли;
КонецЦикла;
```

В программном коде, показанном в листинге 6.4, выводятся имена всех таблиц базы данных. С помощью свойства `SystemObject` объекта `Table` отсекаются системные таблицы.

Зная имена таблиц, можно получить спецификацию каждого ее поля — имя, тип, длина поля, признак первичного ключа, признак возможности содержать пустые значения. Для этих целей у объекта `Table` существует коллекция `Columns`, которая содержит сведения о каждом поле таблицы.

Рассмотрим пример, в котором, помимо имен таблиц, выводятся названия полей и их спецификация (листинг 6.5).

Листинг 6.5. Получение спецификации полей таблиц

```
// Объект базы данных
Database = СоздатьОбъект("SQLDMO.Database");
Database = SQLServer.Databases(ИмяБазыДаных);

// Объект таблица
Table = СоздатьОбъект("SQLDMO.Table");
```

```
Для Ном = 1 По Database.Tables.Count Цикл
  // Выводим только НЕ системные таблицы
  Если Database.Tables.Item(Ном).SystemObject = 0 Тогда
    // Выводим имя таблицы
    ИмяТаблицы = Database.Tables.Item(Ном).Name;
    Сообщить("Имя таблицы:" + ИмяТаблицы);

    Table = Database.Tables(ИмяТаблицы);
      // Цикл по полям
      Для НомерПоля = 1 По Table.Columns.Count Цикл
        Column = Table.Columns.Item(НомерПоля);
        Сообщить("Имя поля: " + Column.Name);
        Сообщить("Тип данных: " + Column.DataType);
        Сообщить("Длина данных: " + Column.Length);
        Сообщить("Это первичный ключ: " + ?(Column.InPrimaryKey = -1,
"Да", "Нет"));
        Сообщить("Может принимать пустые значения: " + ?
(Column.AllowNulls = -1, "Да", "Нет"));
    КонецЦикла;
  КонецЕсли;
КонецЦикла;
```

Листинг 6.5 демонстрирует, как производится выборка элементов коллекции таблиц `Tables` внутри базы данных. Объекты `Table`, в свою очередь, обладают иерархически организованными коллекциями объектов и отдельными объектами. Каждая таблица обязательно имеет коллекцию столбцов `Columns`, но у любой таблицы может иметься единственный объект `PrimaryKey`, соответствующий первичному ключу. Коллекция `Keys` ключей таблицы содержит ссылки на все ограничения первичного ключа таблицы и внешних ее ключей.

Объект `Column` имеет следующие основные свойства:

- `Name` — имя поля;
- `DataType` — строковое представление типа значений для поля (например, `varchar`, `int` и т. п.);
- `Length` — максимально допустимая длина значения;
- `InPrimaryKey` — признак первичного ключа (`-1` — обозначает, что данное поле является первичным ключом, `0` — нет);
- `AllowNulls` — признак возможности содержать пустые значения (`-1` — поле может принимать пустые значения, `0` — не может).

Получение списка представлений

С помощью коллекции `Views` объекта `Database` можно получить доступ к представлениям, которые определены в базе данных. Для обхода элементов коллекции `Views` можно воспользоваться все тем же свойством `Count`, которое возвращает общее количество представлений, и методом `Item`, который получает объект `View` (представление) по его номеру (листинг 6.6). Так же допустим просмотр элементов коллекции с помощью конструкции `Для каждого...из...Цикл...КонецЦикла`.

Листинг 6.6. Получение списка представлений

```
// Объект базы данных
Database = СоздатьОбъект("SQLDMO.Database");
Database = SQLServer.Databases(ИмяБазыДаных);

Для Ном = 1 По Database.Views.Count Цикл
  // Выводим только НЕ системные представления
  Если Database.Views.Item(Ном).SystemObject = 0 Тогда
    Сообщить(Database.Views.Item(Ном).Name);
  КонецЕсли;
КонецЦикла;
```

Данный пример выводит список имен всех представлений базы данных.

Получение списка хранимых процедур

Получение списка хранимых процедур производится методом аналогичным тому, который использовался и для таблиц. Исключением здесь является то, что коллекция, которая содержит элементы свойств хранимых процедур, называется `StoredProcedures` (листинг 6.7).

Листинг 6.7. Получение списка хранимых процедур

```
// Объект базы данных
Database = СоздатьОбъект("SQLDMO.Database");
Database = SQLServer.Databases(ИмяБазыДаных);

Для Ном = 1 По Database.StoredProcedures.Count Цикл
  // Выводим только НЕ системные процедуры
  Если Database.StoredProcedures.Item(Ном).SystemObject = 0 Тогда
```

```
    // Выводим имя хранимой процедуры
    Сообщить(Database.StoredProcedures.Item(Ном).Name);
  КонецЕсли;
КонецЦикла;
```

Изменение структуры баз данных

Создание и удаление баз данных

Для добавления новой базы данных на SQL-сервер предназначен все тот же объект `SQLDMO.Database`. При этом затрагиваются еще два дополнительных объекта:

- `SQLDMO.DBFile` — объект физического файла базы данных;
- `SQLDMO.LogFile` — объект физического файла транзакций.

Для регистрации новой базы данных на сервере необходимо сначала создать физический файл базы данных и файл транзакции.

Рассмотрим пример создания новой базы данных с именем `NewDB` (листинг 6.8).

Листинг 6.8. Создание новой базы данных

```
// Объект базы данных
Database = СоздатьОбъект("SQLDMO.Database");
// Объект файла базы данных
DBFileData = СоздатьОбъект("SQLDMO.DBFile");
// Объект файла транзакций
LogFile = СоздатьОбъект("SQLDMO.LogFile");

// Имя новой БД
ИмяНовойБазыДанных = "NewDB";

// Установить имя новой БД
Database.Name = ИмяНовойБазыДанных;

// Определяем свойства файла БД
DBFileData.Name = ИмяНовойБазыДанных;
DBFileData.PhysicalName = SQLServer.Registry.SQLDataRoot + "\DATA\" +
ИмяНовойБазыДанных + ".mdf";
```

```
DBFileData.PrimaryFile = 1;
DBFileData.Size = 2;
DBFileData.FileGrowth = 1;

// Добавим файл базы данных
Database.FileGroups("PRIMARY").DBFiles.Add(DBFileData);

// Определяем свойства файла транзакций
LogFile.Name = ИмяНовойБазыДанных + "Log";
LogFile.PhysicalName = SQLServer.Registry.SQLDataRoot + "\DATA\" +
LogFile.Name + ".ldf";
LogFile.Size = 2;

// Добавим файл транзакций базы данных
Database.TransactionLog.LogFiles.Add(LogFile);

// Создадим БД
Попытка
  SQLServer.Databases.Add(Database);
Исключение
  Сообщить(ОписаниеОшибки());
  Возврат;
КонецПопытки;
```

Из приведенного выше кода видно, что сначала создаются объекты базы данных `Database`, физического файла `DBFileData` и файла транзакций `LogFile`. Затем, с помощью свойств `Database.FileGroups("PRIMARY").DBFiles` и `Database.TransactionLog.LogFiles`, добавляются соответственно объекты файла базы данных и файла транзакций. Только после этого база данных регистрируется на сервере.

Рассмотрим основные свойства объекта `DBFile`.

- `Name` — имя файла.
- `PhysicalName` — физическое местонахождение файла.
- `PrimaryFile` — признак основного файла базы данных.
- `Size` — начальный размер файла в мегабайтах (необязательное свойство).
- `FileGrowth` — признак, определяющий, что необходимо хранить в базе данных (таблицу, индекс, или регистрационные данные).

Рассмотрим основные свойства объекта `LogFile`.

- `Name` — имя файла.

- `PhysicalName` — физическое местонахождение файла.
- `Size` — начальный размер файла в мегабайтах (необязательное свойство).

Свойство `SQLServer.Registry.SQLDataRoot` возвращает строку с полным именем каталога, в который установлен SQL Server.

Для удаления базы данных достаточно выполнить конструкцию:

```
SQLServer.Databases.Remove(ИмяБазыДанных).
```

Во избежание фатальных ошибок рекомендуется все операции с изменением структуры баз данных обрамлять конструкцией:

```
Попытка...Исключение...КонецПопытки.
```

Создание и удаление таблиц и их полей

Для демонстрации работы механизма создания и удаления таблиц средствами SQL-DMO создадим таблицу товаров `Products`, состоящую из четырех столбцов.

Код, приведенный в листинге 6.9, показывает, как это делается, а также добавляет столбцы, с учетом достаточно специфических правил синтаксиса, которые используются для их создания. Столбцы могут относиться к следующим типам данных:

- `Int` — целочисленному;
- `Varchar` — символьному переменной длины;
- `Money` — денежному;
- `Decimal` — десятичному.

Листинг 6.9. Создание таблицы товаров `Products`

```
// Объект базы данных
Database = СоздатьОбъект("SQLDMO.Database");
// Объект новой таблицы
Table = СоздатьОбъект("SQLDMO.Table");

// Имя базы данных
ИмяБазыДанных = "TestDB";
// Имя новой таблицы
ИмяТаблицы = "Products";

Database = SQLServer.Databases(ИмяБазыДанных);
Table.Name = ИмяТаблицы;
```

```
Table.FileGroup = "PRIMARY";

// Создаем колонки

// Добавить целочисленный тип данных
Column1 = СоздатьОбъект("SQLDMO.Column");
Column1.Name = "ProdID";
Column1.Datatype = "int";
Table.Columns.Add(Column1);

// Добавить тип данных символьной строки
Column2 = СоздатьОбъект("SQLDMO.Column");
Column2.Name = "ProdName";
Column2.DataType = "varchar";
Column2.Length = 25;
Table.Columns.Add(Column2);

// Добавить еще один целочисленный тип данных
Column3 = СоздатьОбъект("SQLDMO.Column");
Column3.Name = "Price";
Column3.DataType = "money";
Table.Columns.Add(Column3);

// Добавить десятичный тип данных
Column4 = СоздатьОбъект("SQLDMO.Column");
Column4.Name = "ProdWeight";
Column4.DataType = "decimal";
Column4.NumericPrecision = 9;
Column4.NumericScale = 5;
Table.Columns.Add(Column4);

// Очистить объекты
Column1 = "";
Column2 = "";
Column3 = "";
Column4 = "";

// Перед добавлением новой таблицы в назначенную базу данных
// удалить предыдущую версию таблицы, если она существует
```

```
Попытка
  Database.Tables(ИмяТаблицы).Remove();
Исключение
КонецПопытки;

// Добавляем новую таблицу
Database.Tables.Add(Table);
```

Для определения типа данных и максимальной длины значения каждого столбца используются соответственно свойства `Datatype` и `Length`. В коде, представленном в листинге 6.9, например, столбец `ProdID` относится к типу целочисленных данных `int`; столбец `ProdName` — к символьному типу данных переменной длины `varchar`.

Доступные типы данных представлены в табл. 6.1.

***Таблица 6.1.** Типы данных полей, доступные в SQL-DMO*

Идентификатор	Описание
`bigint`	Целочисленный тип, от -2^{63} (-9 223372 036854 775808) до 2^{63} (9 223372 036854 775807)
`int`	Целочисленный тип, от -2^{31} (–2 147 483 648) до 2^{31} (2 147 483 647)
`smallint`	Целочисленный тип, от 2^{15} (–32 768) до 2^{15} (32 767)
`tinyint`	Целочисленный тип, от 0 до 255
`bit`	Целочисленный тип, 0 или 1
`decimal`	Тип с фиксированной точностью, от -10^{38} до 10^{38}
`numeric`	Эквивалентно типу `decimal`
`money`	Денежный тип, от -2^{63} (–922 337 203 685 477.5808) до 2^{63} (+922 337 203 685 477.5807)
`smallmoney`	Монетный тип, от -214 748.3648 до +214 748.3647
`float`	Числовой тип с плавающей запятой, до –1.79E+308 по 1.79E+308
`real`	Числовой тип с плавающей запятой, от -3.40E+38 до 3.40E+38
`datetime`	Дата и время, начиная с 1 января 1753 по 31 декабря 9999 с точностью до 3.33 миллисекунд
`smalldatetime`	Дата и время, начиная с 1 января 1900 по 6 июля 2079 с точностью до одной минуты
`char`	Строка фиксированной длины (не Unicode), максимально допустимая длина — 8000 символов

Таблица 6.1 (окончание)

Идентификатор	Описание
varchar	Строка переменной длины (не Unicode), максимально допустимая длина — 8000 символов
text	Строка переменной длины (не Unicode), максимально допустимая длина — 2^{31} (2 147 483 647) символов
nchar	Строка фиксированной длины (Unicode), максимально допустимая длина — 4000 символов
nvarchar	Строка переменной длины (Unicode), максимально допустимая длина — 4000 символов
ntext	Строка переменной длины (Unicode), максимально допустимая длина — 2^{30} (1 073 741 823) символов
binary	Двоичные данные фиксированной длины до 8000 байт
varbinary	Двоичные данные переменной длины до 8000 байт
image	Двоичные данные переменной длины до 2^{31} (2 147 483 647) байт
cursor	Ссылка на курсор
sql_variant	Любой тип данных, поддерживаемый SQL Server
table	Специальный тип данных, используемый в качестве хранилища результата обработки данных
timestamp	Уникальный номер в пределах базы данных
uniqueidentifier	Глобальный уникальный идентификатор (GUID)

Как видно из приведенного примера, определение столбцов таблицы реализуется в три этапа.

На первом этапе необходимо создать экземпляр столбца.

На втором — код должен настроить этот столбец, присвоив значения таким его свойствам, как название `Name` и тип данных `DataType`. Определяемые свойства могут меняться в зависимости от типа данных столбца. К примеру, для столбца, принадлежащего целочисленному типу данных `int`, достаточно настроить всего два свойства — имя и тип данных. В то же время столбец, относящийся к типу данных `varchar`, требует определения значений как минимум трех свойств — названия (`Name`), типа данных (`DataType`) и длины (`Length`).

Третий этап создания столбца предусматривает инициирование метода добавления `Add`, который и добавит созданный столбец в коллекцию столбцов `Columns` рассматриваемой таблицы.

С помощью метода `Database.Tables.Add` происходит добавление таблицы к подключенной базе данных. До начала добавления таблицы метод `Remove`

удаляет из базы данных все ранние версии этой таблицы. После этого код вычищает все дочерние объекты.

Часть программного кода, отвечающая за установление соединения с сервером, на котором будет размещена база данных, в примере не показана, так как подразумевается, что подключение было совершено ранее, по аналогии с кодом, приведенном в листинге 6.2.

Теперь рассмотрим пример создания таблицы заказов `Orders` (листинг 6.10). У этой таблицы имеется первичный ключ, построенный по столбцу с активизированным свойством идентичности `Identity`. Приведенный пример кода во многом напоминает код из листинга 6.9, за исключением перевода свойства `AllowNulls` третьего столбца в состояние `-1`.

Листинг 6.10. Создание таблицы заказов `Orders`

```
// Объект базы данных
Database = СоздатьОбъект("SQLDMO.Database");
// Объект новой таблицы
Table = СоздатьОбъект("SQLDMO.Table");

// Имя базы данных
ИмяБазыДанных = "TestDB";
// Имя новой таблицы
ИмяТаблицы = "Orders";

Database = SQLServer.Databases(ИмяБазыДанных);
Table.Name = ИмяТаблицы;
Table.FileGroup = "PRIMARY";

// Создаем колонки

// Добавить целочисленный тип данных
Column1 = СоздатьОбъект("SQLDMO.Column");
Column1.Name = "OrderID";
Column1.Datatype = "int";
Column1.AllowNulls = 0;
Column1.Identity = -1;
Column1.IdentitySeed = 1000;
Column1.IdentityIncrement = 10;
Table.Columns.Add(Column1);
```

```
// Добавить столбец со свойством Identity, который будет служить первичным
ключом таблицы
Key1 = СоздатьОбъект("SQLDMO.Key");
Key1.Name = "OrdersPK";
Key1.Type = 1;
Key1.Clustered = -1;
Key1.KeyColumns.Add(Column1.Name);
Table.Keys.Add(Key1);

// Добавить временной тип данных datetime
Column2 = СоздатьОбъект("SQLDMO.Column");
Column2.Name = "OrderDate";
Column2.DataType = "datetime";
Table.Columns.Add(Column2);

// Добавить тип данных datetime, допускающий неопределенные значения
Column3 = СоздатьОбъект("SQLDMO.Column");
Column3.Name = "ShippedDate";
Column3.DataType = "datetime";
Column3.AllowNulls = -1;
Table.Columns.Add(Column3);

// Перед добавлением новой таблицы в назначенную
// базу данных удалить предыдущую версию таблицы,
// если она существует.
Попытка
  Database.Tables(ИмяТаблицы).Remove();
Исключение
КонецПопытки;

// Добавляем новую таблицу
Database.Tables.Add(Table);

// Очистить объекты
Column1 = "";
Column2 = "";
Column3 = "";
```

Код, приведенный в листинге 6.10, создает таблицу заказов с именем Orders. Таблица Orders содержит столбец идентификатора заказа OrderID; столбец OrderDate, в который заносится дата ввода заказа, а также столбец ShippedDate, в который записывается дата отправки заказа. Столбец OrderID служит первичным ключом таблицы, а столбец ShippedDate может содержать неопределенные значения NULL. Чтобы заставить SQL Server автоматически формировать значения первичного ключа для новых строк, процедура активизирует свойство Identity, присваивая ему значение -1 (Истина или True). При этом исходное значение равно 1000, а приращение составляет 10. После добавления в проект таблицы столбца со свойством Identity создается экземпляр Key1 ключевого объекта Key. Затем присваиваются значения свойствам Name и Type объекта Key1, которые содержат соответственно название и тип данных. Все возможные значения свойства Type объекта Key приведены в табл. 6.2.

Таблица 6.2. Типы ключей SQL-DMO

Константа SQL-DMO	Значение	Описание
SQLDMOKey_Foreign	3	Внешний ключ
SQLDMOKey_Primary	1	Первичный ключ
SQLDMOKey_Unique	2	Уникальное поле, не допускающее значение NULL
SQLDMOKey_Unknown	0	Ошибочное значение

Свойству Clustered объекта Key1 присваивается значение -1 (Истина или True). В результате уникальный индекс первичного ключа строится в виде кластеризованного индекса для всей таблицы Orders. Прежде чем добавить ключ Key1 в коллекцию ключей Keys таблицы Orders, необходимо указать хотя бы один объект из класса столбцов, который будет поставлен в соответствие первичному ключу. В приведенном примере в этой роли выступает столбец OrderID, у которого активизировано свойство Identity.

Во все столбцы (листинг 6.9) и в два первых столбца (листинг 6.10) необходимо вводить определенные значения. Иное дело третий столбец — ShippedDate (листинге 6.10). В нем могут находиться неопределенные значения NULL, поскольку дата фактической отправки заказа неизвестна на момент ввода сведений о заказе. Она вводится в столбец ShippedDate позднее, когда заказ уже отправлен. Поэтому в третьем столбце присваивается значение -1 свойству AllowNulls, после чего можно помещать в него неопределенные значения. По умолчанию значение этого свойства равно 0 (Ложь или False).

Таблица товаров Products (листинг 6.9) и таблица заказов Orders (листинг 6.10) связаны между собой отношением "многие-ко-многим". Такое

отношение имеет место потому, что один и тот же товар может войти в один или несколько заказов, а каждый заказ может включать многие товары. Чтобы отразить такое отношение в разрабатываемом проекте базы данных, необходимо внести в него два изменения. Во-первых, нужно переделать проект таблицы товаров `Products` таким образом, чтобы у нее тоже появился первичный ключ. Во-вторых, необходимо добавить в базу данных новую таблицу, которая свяжет таблицы `Products` и `Orders`. В этой таблице будут храниться общие данные доменов обеих таблиц. К примеру, в ней можно хранить количество определенного товара, которое указывается в отдельной строке заказа. Новую таблицу, которая связывает таблицы `Products` и `Orders`, назовем `OrderDetails`.

В листинге 6.11 приведен пример создания таблицы связей `OrderDetails`. У этой таблицы имеются внешние ключи, которые ссылаются на таблицы `Orders` и `Products`.

Листинг 6.11. Создание таблицы с подробными сведениями о заказах `OrderDetails`

```
// Объект базы данных
Database = СоздатьОбъект("SQLDMO.Database");
// Объект новой таблицы
Table = СоздатьОбъект("SQLDMO.Table");

// Имя базы данных
ИмяБазыДанных = "TestDB";
// Имя новой таблицы
ИмяТаблицы = "OrderDetails";

Database = SQLServer.Databases(ИмяБазыДанных);
Table.Name = ИмяТаблицы;
Table.FileGroup = "PRIMARY";

// Создаем колонки

// Добавить целочисленный тип данных
Column1 = СоздатьОбъект("SQLDMO.Column");
Column1.Name = "OrderID";
Column1.Datatype = "int";
Table.Columns.Add(Column1);
```

```
// Добавить целочисленный тип данных
Column2 = СоздатьОбъект("SQLDMO.Column");
Column2.Name = "ProdID";
Column2.DataType = "int";
Table.Columns.Add(Column2);

// Добавить внешний ключ, указывающий на таблицу Orders
Key1 = СоздатьОбъект("SQLDMO.Key");
Key1.Name = "OrderIDFK";
Key1.Type = 3;
Key1.KeyColumns.Add(Column1.Name);
Key1.ReferencedTable = "Orders";
Key1.ReferencedColumns.Add("OrderID");
Table.Keys.Add(Key1);

// Добавить внешний ключ, указывающий на таблицу Products
Key2 = СоздатьОбъект("SQLDMO.Key");
Key2.Name = "ProdIDFK";
Key2.Type = 3;
Key2.KeyColumns.Add(Column2.Name);
Key2.ReferencedTable = "Products";
Key2.ReferencedColumns.Add("ProdID");
Table.Keys.Add(Key2);

// Добавить первичный ключ, состоящий из двух столбцов
Key3 = СоздатьОбъект("SQLDMO.Key");
Key3.Name = "OrderIDAndProdIDPK";
Key3.Type = 1;
Key3.Clustered = -1;
Key3.KeyColumns.Add(Column1.Name);
Key3.KeyColumns.Add(Column2.Name);
Table.Keys.Add(Key3);

// Добавить целочисленный тип данных
Column3 = СоздатьОбъект("SQLDMO.Column");
Column3.Name = "Quantity";
Column3.DataType = "int";
```

```
Table.Columns.Add(Column3);

// Перед добавлением новой таблицы в назначенную
// базу данных удалить предыдущую версию таблицы,
// если она существует.
Попытка
  Database.Tables(ИмяТаблицы).Remove();
Исключение
КонецПопытки;

// Добавляем новую таблицу
Database.Tables.Add(Table);

// Очистить объекты
Column1 = "";
Column2 = "";
Column3 = "";
```

Код, приведенный в листинге 6.11, показывает также синтаксис, применяемый при построении первичного ключа на основе нескольких столбцов. Код, создающий внешний ключ, начинается с генерации экземпляра объекта ключа `Key`. После присвоения имени этому объекту, его свойству `Type` присваивается признак внешнего ключа (значение 3). Вслед за этим добавляется свойство `Name` объекта `Key`, содержащее название этого столбца, в принадлежащую ключу коллекцию названий ключевых столбцов `KeyColumns`. В соответствии со спецификацией столбец `OrderID` таблицы `OrderDetails` назначается локальным столбцом внешнего ключа. Затем происходит назначение таблицы `Orders` и ее столбца `OrderID` соответственно таблицей и столбцом, на которые будут формироваться ссылки. После настройки всех этих свойств программный код, формирующий первый внешний ключ, завершает свою работу добавлением созданного ключа в коллекцию ключей `Keys` таблицы `OrderDetails`. После этого создается внешний ключ, который будет указывать на столбец идентификатора товара `ProdID` в таблице товаров `Products` из столбца `ProdID` таблицы `OrderDetails`.

Часть кода, в которой описан объект `Key3`, демонстрирует синтаксис, применяемый для формирования первичного ключа из нескольких столбцов таблицы. Этот блок операторов очень похож на те, в которых определяются внешние ключи таблицы.

Для того чтобы удалить поле таблицы, необходимо воспользоваться методом `Column.Remove()`, где идентификатор `Column` является объектом удаляемого поля.

Создание SQL-скриптов

В листинге 6.4 был приведен пример получения списка таблиц, имеющихся в базе данных. Аналогичным образом, через объект `Database.Tables`, можно сгенерировать SQL-скрипт, который можно использовать для создания таблиц в другой базе или на другом сервере.

В листинге 6.12 приведен пример генерации SQL-скрипта для создания таблицы `Customers`, находящейся в базе данных `Northwind`, входящий в поставку продукта Microsoft SQL Server 7.0/2000.

Листинг 6.12. Генерация SQL-скрипта создания таблицы `Customers` базы данных `Northwind`

```
Database = СоздатьОбъект("SQLDMO.Database");
Database = SQLServer.Databases("Northwind");
// Выводим скрипт
Сообщить(Database.Tables.Item("Customers").Script());
```

SQL-скрипт, созданный в результате выполнения приведенного выше кода, представлен в листинге 6.13.

Листинг 6.13. SQL-скрипт создания таблицы `Customers`

```
CREATE TABLE [Customers] (
  [CustomerID] [nchar] (5) COLLATE Cyrillic_General_CI_AS NOT NULL ,
  [CompanyName] [nvarchar] (40) COLLATE Cyrillic_General_CI_AS NOT NULL ,
  [ContactName] [nvarchar] (30) COLLATE Cyrillic_General_CI_AS NULL ,
  [ContactTitle] [nvarchar] (30) COLLATE Cyrillic_General_CI_AS NULL ,
  [Address] [nvarchar] (60) COLLATE Cyrillic_General_CI_AS NULL ,
  [City] [nvarchar] (15) COLLATE Cyrillic_General_CI_AS NULL ,
  [Region] [nvarchar] (15) COLLATE Cyrillic_General_CI_AS NULL ,
  [PostalCode] [nvarchar] (10) COLLATE Cyrillic_General_CI_AS NULL ,
  [Country] [nvarchar] (15) COLLATE Cyrillic_General_CI_AS NULL ,
  [Phone] [nvarchar] (24) COLLATE Cyrillic_General_CI_AS NULL ,
  [Fax] [nvarchar] (24) COLLATE Cyrillic_General_CI_AS NULL ,
  CONSTRAINT [PK_Customers] PRIMARY KEY  CLUSTERED
  (
    [CustomerID]
  )  ON [PRIMARY]
) ON [PRIMARY]
GO
```

В листинге 6.12 показано, что получить ссылку на объект таблицы можно не только по его номеру, как это было показано в листинге 6.4, но и по имени.

Метод `Script` возвращает строковое выражение скрипта соответствующего объекта. В листинге 6.14 приведен пример генерации полного SQL-скрипта создания базы данных, включающий:

- сценарий создания таблиц;
- сценарий создания представлений;
- сценарий создания хранимых процедур;
- сценарии создания пользователей;
- сценарии создания ролей.

Листинг 6.14. Генерация полного SQL-скрипта создания базы данных

```
Database = СоздатьОбъект("SQLDMO.Database");
Database = SQLServer.Databases(ИмяБазыДанных);
СтрокаСценария = "";

Попытка
  // Сценарий создания таблиц
  Для Ном = 1 По Database.Tables.Count Цикл
    СтрокаСценария = СтрокаСценария + Database.Tables.Item(Ном).Script();
  КонецЦикла;

  // Сценарий создания представлений
  Для Ном = 1 По Database.Views.Count Цикл
    СтрокаСценария = СтрокаСценария + Database.Views.Item(Ном).Script();
  КонецЦикла;

  // Сценарий создания хранимых процедур
  Для Ном = 1 По Database.StoredProcedures.Count Цикл
    СтрокаСценария = СтрокаСценария + Data-
base.StoredProcedures.Item(Ном).Script();
  КонецЦикла;

  // Сценарии создания пользователей
  Для Ном = 1  По Database.Users.Count Цикл
    СтрокаСценария = СтрокаСценария + Database.Users.Item(Ном).Script();
  КонецЦикла;
```

```
  // Сценарии создания ролей
  Для Ном = 1  По Database.DatabaseRoles.Count Цикл
    СтрокаСценария = СтрокаСценария +
    Database.DatabaseRoles.Item(Ном).Script();
  КонецЦикла;

  Текст = СоздатьОбъект("Текст");
  Текст.ДобавитьСтроку(СтрокаСценария);
  Текст.Показать("Полный скрипт создания базы данных",);
Исключение
  Сообщить(ОписаниеОшибки());
  Возврат;
КонецПопытки;
```

Выполнение SQL-запросов

В SQL-DMO, помимо административных задач, можно получать выборки данных с помощью SQL-запросов. Для данных целей предназначен метод `ExecuteWithResults` объекта `Database`. Метод `ExecuteWithResults` имеет единственный параметр — строку SQL-запроса и возвращает объект типа `QueryResults`.

Рассмотрим основные свойства объекта `QueryResults`.

- `Rows` — возвращает количество записей в результирующей таблице запроса;
- `Columns` — возвращает количество полей в результирующей таблице запроса;
- `ColumnType(НомерКолонки)` — возвращает тип соответствующего поля (все возможные типы перечислены в табл. 6.3);
- `ColumnName(НомерКолонки)` — возвращает имя соответствующего поля.

Таблица 6.3. *Типы полей объекта* `QueryResults`

Константа SQL-DMO	Значение	Описание
SQLDMO_DTypeBigint	-5	Целочисленный тип bigint
SQLDMO_DTypeBinary	-2	Двоичный тип фиксированной длины
SQLDMO_DTypeBit	-7	Положительный целочисленный тип
SQLDMO_DTypeChar	1	Строковый тип фиксированной длины

Таблица 6.3 (окончание)

Константа SQL-DMO	Значение	Описание
SQLDMO_DTypeDateTime	-2	Специальный тип ODBC — SQL_TIMESTAMP_STRUCT
SQLDMO_DTypeDateTime4	93	Специальный тип ODBC — SQL_TIMESTAMP_STRUCT
SQLDMO_DTypeFloat4	7	Числовой тип с плавающей запятой (4 байта)
SQLDMO_DTypeFloat8	8	Числовой тип с плавающей запятой (8 байт)
SQLDMO_DTypeGUID	-11	Глобальный уникальный идентификатор (GUID)
SQLDMO_DTypeImage	-4	Двоичный тип переменной длины
SQLDMO_DTypeInt1	-6	Положительный числовой тип (1 байт)
SQLDMO_DTypeInt2	5	Числовой тип со знаком (2 байта)
SQLDMO_DTypeInt4	4	Числовой тип со знаком (4 байта)
SQLDMO_DTypeMoney	3	Денежный тип
SQLDMO_DTypeMoney4	3	Расширенный денежный тип
SQLDMO_DTypeNText	-10	Расширенный строковый тип (Unicode)
SQLDMO_DtypeSQLVariant	-150	Любой тип данных, поддерживаемый SQL Server
SQLDMO_DTypeText	-1	Расширенный строковый тип
SQLDMO_DTypeUChar	-8	Строковый тип фиксированной длины (Unicode)
SQLDMO_DTypeUnknown	0	Ошибочный тип
SQLDMO_DTypeUVarchar	-9	Строковый тип переменной длины (Unicode)
SQLDMO_DTypeVarBinary	-3	Двоичный тип переменной длины
SQLDMO_DTypeVarchar	12	Строковый тип переменной длины

Основной метод `GetColumnString` объекта `QueryResults` позволяет получить значение результата запроса, находящееся в определенной строке и поле. Метод `GetColumnString` имеет два параметра — номер строки и номер колонки (поля).

В листинге 6.15 приведен пример выполнения SQL-запроса и вывода результата на экран в виде таблицы значений.

Листинг 6.15. Общий алгоритм выполнения SQL-запросов

```
Попытка
  ИмяБазыДанных = "Northwind";
  СтрокаЗапроса = "SELECT * FROM Customers";
  ТЗ_Запроса = СоздатьОбъект("ТаблицаЗначений");

  Состояние("Выполнение запроса...");
  QueryResults = SQLServer.Databases(ИмяБазыДанных).ExecuteWithResults
(СтрокаЗапроса);

  Состояние("Преобразование данных...");
  // Заполняем таблицу, перебирая результаты запроса по строкам
  Для НомерСтроки = 1 по QueryResults.Rows Цикл
    Если НомерСтроки = 1 Тогда
      // Формируем колонки в таблице с именами,
      // взятыми из результатов запроса и соответствующего типа
      Для НомерКолонки = 1 по QueryResults.Columns Цикл
        // Вставляем колонку
        ТЗ_Запроса.ВставитьКолонку(QueryResults.ColumnName
        (НомерКолонки), НомерКолонки, "Строка",,,
        QueryResults.ColumnName (НомерКолонки),15,);
      КонецЦикла;
    КонецЕсли;
    // Добавляем строчку
    ТЗ_Запроса.НоваяСтрока();
    // Поколоночно пишем в таблицу данные
    Для НомерКолонки = 1 по QueryResults.Columns Цикл
      ТЗ_Запроса.УстановитьЗначение(НомерСтроки, НомерКолонки,
      QueryResults.GetColumnString(НомерСтроки,НомерКолонки));
    КонецЦикла;
  КонецЦикла;

  // Показать таблицу
  ТЗ_Запроса.ВыбратьСтроку();
Исключение
  Сообщить(ОписаниеОшибки());
  Возврат;
КонецПопытки;
```

Данный пример является универсальным, т. е. в переменную СтрокаЗапроса можно поместить произвольный текст запроса, и при этом по-прежнему будет корректно выводиться результат выполнения запроса. Это достигается благодаря наличию свойств Rows, Columns и ColumnName у объекта QueryResults. В результате обработки объекта QueryResults получаем таблицу значений, содержащую результаты выполнения запроса.

Таким образом, как следует из данного примера, общий алгоритм выполнения произвольного запроса следующий:

1. С помощью метода ExecuteWithResults объекта Database получаем объект QueryResults, содержащий результат выполнения запроса.
2. Для инициализации колонок результирующей таблицы значений обходим все поля объекта QueryResults, с помощью свойства ColumnName получаем имена полей, после чего создаем одноименные колонки в таблице значений. Количество полей определяется свойством Columns.
3. Построчно обходим все записи объекта QueryResults. Количество записей определяется свойством Rows.
4. Для каждой записи обходим все поля объекта QueryResults, с помощью метода GetColumnString получаем значения соответствующих ячеек и помещаем их в результирующую таблицу значений.

Резервирование и восстановление базы данных

Для создания резервной копии базы данных предназначен объект Backup.

Рассмотрим основные свойства этого объекта.

- Action — свойство, определяющее, по отношению к чему выполнять действия резервирования данных (возможные значения свойства перечислены в табл. 6.4);
- Database — имя резервируемой базы данных;
- Files — имя файла резервных данных;
- MediaName — дополнительное описание, помогающее в идентификации резервной копии;
- BackupSetDescription — основное описание резервной копии;
- BackupSetName — идентификатор резервной копии.

Метод SQLBackup объекта Backup предназначен для создания резервной копии и имеет один параметр — ссылку на объект SQLServer.

Таблица 6.4. Возможные значения свойства `Action` объекта `Backup`

Константа SQL-DMO	Значение	Описание
`SQLDMOBackup_Database`	0	Резервирование всей базы данных
`SQLDMOBackup_Files`	2	Резервирование только определенных файлов
`SQLDMOBackup_Incremental`	1	Дифференциальное резервирование, т. е. копируются файлы, созданные или измененные с момента последнего резервирования
`SQLDMOBackup_Log`	3	Резервирование только файла транзакций

В листинге 6.16 приведен пример полного резервирования базы данных `Northwind` в файл C:\BACKUP\Northwind.bak.

Листинг 6.16. Создание резервной копии базы данных `Northwind`

```
Backup = СоздатьОбъект("SQLDMO.Backup");
Backup.Action = 0;
Backup.Database = "Northwind";
Backup.Files = "C:\BACKUP\Northwind.bak";
Backup.MediaName = "Northwind.bak " + РабочаяДата() + " " +
ТекущееВремя();
Backup.BackupSetName = "Northwind";
Backup.BackupSetDescription = "Резервная копия";
Backup.SQLBackup(SQLServer);
```

Для восстановления созданной ранее резервной копии предназначен объект `Restore`.

Рассмотрим основные свойства этого объекта.

- `Action` — описывает по отношению к чему выполнять восстановление данных (возможные значения свойства перечислены в табл. 6.5);
- `Database` — имя базы данных, для которой восстанавливаются данные;
- `Files` — имя файла резервной копии;
- `FileNumber` — порядковый номер записи резервной копии;
- `ReplaceDatabase` — признак замены резервного образа базы данных (свойство может принимать значение `-1` — истина и `0` — ложь);
- `LastRestore` — признак восстановления последних записей файла транзакций (свойство может принимать значение `-1` — истина и `0` — ложь).

Таблица 6.5. Возможные значения свойства `Action` *объекта* `Restore`

Константа SQL-DMO	Значение	Описание
`SQLDMORestore_Database`	0	Восстановление всей базы данных
`SQLDMORestore_Files`	1	Восстановление только определенных файлов
`SQLDMORestore_Log`	2	Восстановление только файла транзакций

В листинге 6.17 приведен пример восстановления базы данных `Northwind` из ранее созданного резервного файла C:\BACKUP\Northwind.bak.

Листинг 6.17. Восстановление базы данных `Northwind` из резервной копии

```
Restore = СоздатьОбъект("SQLDMO.Restore");
Restore.Action = 0;
Restore.Database = "Northwind";
Restore.Files = "C:\BACKUP\Northwind.bak";
Restore.FileNumber = 1;
Restore.ReplaceDatabase = -1;
Restore.LastRestore = -1;
Restore.SQLRestore(SQLServer);
```

Настройка ограничений доступа к данным

Для настройки ограничений доступа к базе данных в SQL-DMO существует три объекта:

- `Login` — предназначен для управления параметрами аутентификации для контроля доступа к SQL-серверу;
- `DatabaseRole` — предназначен для управления списком ролей базы данных;
- `User` — предназначен для управления списком пользователей базы данных.

Объект `Login` используется для создания или модификации регистрационных имен пользователей SQL Server и их атрибутов. Объект `Login` имеет следующие свойства.

- `Name` — имя пользователя в формате `"Имя домена\Имя пользователя"`;
- `Type` — тип пользователя (возможные значения свойства перечислены в табл. 6.6);
- `DenyNTLogin` — признак запрета Windows NT авторизации пользователя.

Примечание

Если свойство DenyNTLogin установлено в -1 (Истина), то любая попытка Windows NT авторизации при подключении к SQL-серверу отвергается им. Если свойство установлено в 0 (Ложь) — Windows NT авторизация разрешена.

***Таблица 6.6.** Возможные значения свойства Type объекта Login*

Константа SQL-DMO	Значение	Описание
SQLDMOLogin_NTGroup	1	Имя пользователя SQL Server ссылается на группу пользователей, определенную в Windows
SQLDMOLogin_NTUser	0	Имя пользователя SQL Server ссылается на пользователя, созданного в Windows
SQLDMOLogin_Standard	2	Стандартный пользователь с SQL Server авторизацией

В листинге 6.18 приведен пример создания нового пользователя SQL Server с именем "MAVCOMP\mav" и Windows NT авторизацией.

Листинг 6.18. Создание регистрационного имени пользователя

```
Login = СоздатьОбъект("SQLDMO.Login");
Login.Name = "MAVCOMP\mav";
Login.Type = 1;
SQLServer.Logins.Add(Login);
```

Как следует из приведенного примера, добавление нового пользователя производится с помощью метода Add коллекции Logins объекта SQLServer.

Если свойство Type имеет значение 2 (SQLDMOLogin_Standard), то для пользователя можно установить пароль с помощью метода SetPassword. Общий синтаксис данного метода следующий:

```
Login.SetPassword("старый пароль","новый пароль").
```

Примечание

Если ранее пароль не вводился, то в первом параметре указывается пустая строка.

Для удаления пользователя SQL-сервера в SQL-DMO предназначен метод Remove объекта User. Для удаления ранее созданного пользователя достаточно выполнить команду:

```
SQLServer.Logins("MAVCOMP\mav").Remove().
```

Второй рассматриваемый объект `DatabaseRole`. Он предназначен для управления списком ролей базы данных. Microsoft SQL Server имеет ряд стандартных ролей, описанных в табл. 6.7.

Таблица 6.7. *Список стандартных ролей Microsoft SQL Server*

Имя роли	Описание
`Db_owner`	Разрешены любые действия над базой данных
`Db_accessadmin`	Разрешено добавление и удаление пользователей базы данных
`Db_datareader`	Разрешено чтение любых таблиц
`Db_datawriter`	Разрешено вносить изменения в записи таблиц
`Db_ddladmin`	Разрешен запуск DDL (Data Definition Language — язык определения данных) команд
`Db_securityadmin`	Разрешена модификация прав и изменение ролей
`Db_backupoperator` `Backs up the database`	Разрешено выполнение резервного копирования базы данных
`Db_denydatareader`	Запрещено любое чтение данных из таблиц
`Db_denydatawriter`	Запрещено любое добавление, изменение или удаление записей в таблицах или представлениях

Помимо стандартных ролей SQL-сервер может иметь и роли приложений (Application Roles), которые в SQL-DMO можно создавать с помощью объекта `DatabaseRole`.

Рассмотрим основные свойства объекта `DatabaseRole`.

- `Name` — имя роли.
- `AppRole` — признак роли приложения. Свойство может принимать значение `-1` (Истина) и `0` (Ложь).

> **Примечание**
>
> Если свойство установлено в `-1` (Истина), то для роли необходимо заполнить свойство `Password`.

- `Password` — пароль.

В листинге 6.19 приведен пример создания новой роли базы данных.

Листинг 6.19. Создание новой роли базы данных Northwind

```
DbRole = СоздатьОбъект("SQLDMO.DatabaseRole");
DbRole.Name = "AppRole";
DbRole.AppRole = -1;
DbRole.Password = "erpg_ru";
SQLServer.Databases("Northwind").DatabaseRoles.Add(DbRole);
```

Добавление новой роли производится с помощью метода `Add` коллекции `DatabaseRoles` объекта `Database`. Удаление роли происходит при помощи метода `Remove` объекта `DatabaseRole`. Для удаления ранее созданной роли достаточно выполнить команду:

```
SQLServer.Databases("Northwind").DatabaseRoles("AppRole").Remove().
```

Для создания нового пользователя базы данных в SQL-DMO предназначен объект `User`. Объект `User` имеет следующие свойства:

- `Name` — имя пользователя;
- `Login` — регистрационное имя пользователя SQL Server;
- `Role` — имя роли пользователя.

В листинге 6.20 приведен пример создания нового пользователя с именем `"MAV"` базы данных `Northwind`.

Листинг 6.20. Создание нового пользователя базы данных Northwind

```
DbUser = СоздатьОбъект("SQLDMO.User");
DbUser.Name = "MAV";
DbUser.Login = "MAVCOMP\mav";
SQLServer.Databases("Northwind").Users.Add(DbUser);
```

Для удаления пользователя базы данных предназначен метод `Remove` объекта `User`. Для удаления ранее созданного пользователя достаточно выполнить команду:

```
SQLServer. Databases("Northwind"). Users("MAV"). Remove().
```

После того как создан пользователь базы данных, необходимо с помощью метода `Grant` определить его права. Права пользователя можно определить отдельно для базы данных, для таблиц, представлений и хранимых процедур.

В листинге 6.21 приведен пример установки прав для пользователя `"MAV"`.

Листинг 6.21. Определение прав пользователя "MAV" для базы данных Northwind

```
Database = SQLServer.Databases("Northwind");
// Установить права к базе данных Northwind
Database.Grant(128, "MAV");
// Установить права к таблице Customers
Database.Tables("Customers").Grant(63, "MAV");
// Установить права к представлению Invoices
Database.Views("Invoices").Grant(63, "MAV");
// Установить права к хранимой процедуре CustOrdersOrders
Database.StoredProcedures("CustOrdersOrders").Grant(16, "MAV");
```

В приведенном примере устанавливаются права доступа к таблице `Customers`, к представлению `Invoices`, к хранимой процедуре `CustOrdersOrders` и к базе данных `Northwind`. Результат установки полного доступа к таблице `Customers` для пользователя "MAV" приведен на рис. 6.1.

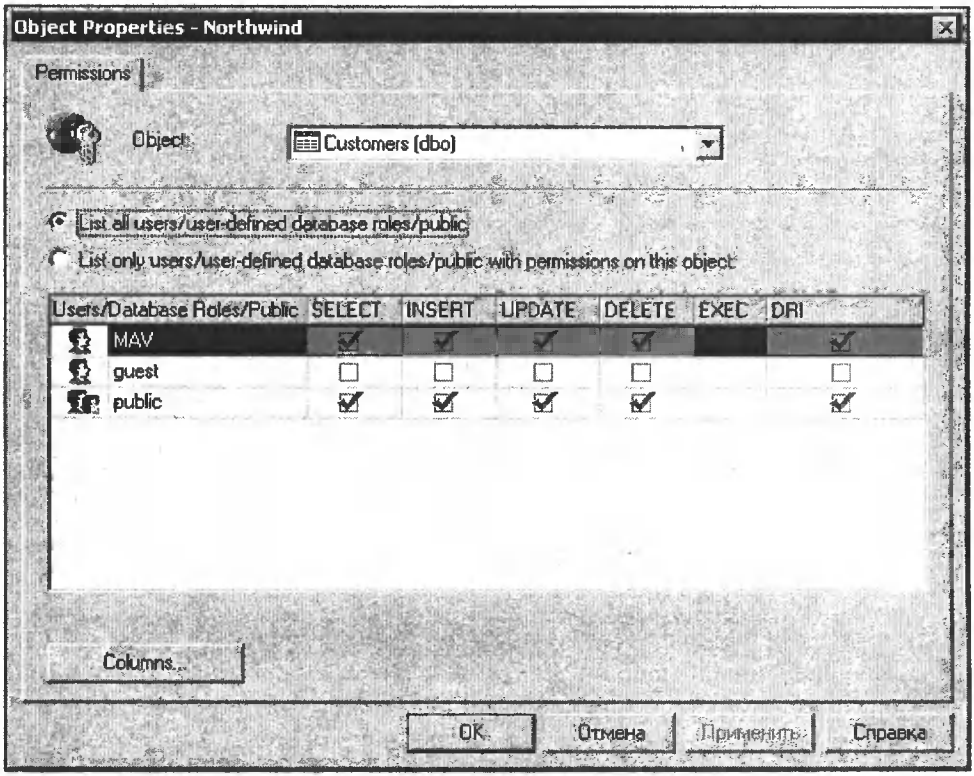


Рис. 6.1. Установка полного доступа к таблице `Customers`

Как видно из примера, приведенного в листинге 6.21, метод `Grant` имеет два параметра:

- номер разрешенных действий;
- имя пользователя базы данных.

Для разных объектов набор разрешенных действий различен.

В табл. 6.8 перечислены значения первого параметра метода `Grant` объекта `Database`.

***Таблица 6.8.** Возможные значения разрешенных действий для объекта `Database`*

Константа SQL-DMO	Значение	Описание
`SQLDMOPriv_AllDatabasePrivs`	130944	Разрешены любые действия пользователя над базой данных
`SQLDMOPriv_CreateDatabase`	256	Пользователю разрешено создание баз данных
`SQLDMOPriv_CreateDefault`	4096	Пользователю разрешено создание объекта `default` и выполнение команды `CREATE DEFAULT`
`SQLDMOPriv_CreateFunction`	65366	Разрешено создание пользовательских функций
`SQLDMOPriv_CreateProcedure`	1024	Разрешено создание хранимых процедур
`SQLDMOPriv_CreateRule`	16384	Разрешено создание ролей
`SQLDMOPriv_CreateTable`	128	Разрешено создание таблиц
`SQLDMOPriv_CreateView`	512	Разрешено создание представлений
`SQLDMOPriv_DumpDatabase`	2048	Разрешено создание резервных копий базы данных
`SQLDMOPriv_DumpTable`	32768	Оставлено для совместимости с ранними версиями SQL-DMO
`SQLDMOPriv_DumpTransaction`	8192	Разрешено создание резервных копий файла транзакций

В табл. 6.9 перечислены возможные значения разрешенных действий для объектов `Table` и `View`.

Таблица 6.9. *Возможные значения разрешенных действий для объектов* `Table` *и* `View`

Константа SQL-DMO	Значение	Описание
`SQLDMOPriv_AllObjectPrivs`	63	Разрешены любые действия пользователя над таблицей или представлением
`SQLDMOPriv_Delete`	8	Разрешено удаление записей и использование оператора DELETE
`SQLDMOPriv_Insert`	2	Разрешена вставка записей и использование оператора INSERT
`SQLDMOPriv_References`	32	Разрешено создание связей между таблицами с помощью внешних ключей
`SQLDMOPriv_Select`	1	Разрешена выборка данных с помощью оператора SELECT
`SQLDMOPriv_Update`	4	Разрешено использование оператора UPDATE

В табл. 6.10 перечислены возможные значения разрешенных действий для объекта `StoredProcedure`.

Таблица 6.10. *Возможные значения разрешенных действий для объекта* `StoredProcedure`

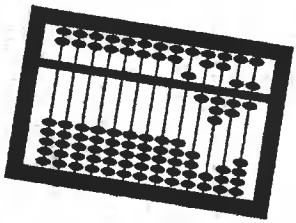
Константа SQL-DMO	Значение	Описание
`SQLDMOPriv_AllObjectPrivs`	63	Разрешены любые действия пользователя над хранимой процедурой
`SQLDMOPriv_Execute`	16	Разрешен запуск хранимой процедуры

Для того чтобы отключить соответствующее право в SQL-DMO, предназначен метод `Revoke`, который по синтаксису аналогичен методу `Grant`.

В листинге 6.22 приведен программный код, отменяющий все установленные ранее (листинг 6.21) права.

Листинг 6.22. Отмена установленных прав пользователя для базы данных Northwind

```
Database = SQLServer.Databases("Northwind");
// Отмена прав к базе данных Northwind
Database.Revoke(128, "MAV");
// Отмена прав к таблице Customers
Database.Tables("Customers").Revoke(63, "MAV");
// Отмена прав к представлению Invoices
Database.Views("Invoices").Revoke(63, "MAV");
// Отмена прав к хранимой процедуре CustOrdersOrders
Database.StoredProcedures("CustOrdersOrders").Revoke(16, "MAV");
```



Глава 7

Команды управления Windows, использующие rundll32

Понятие rundll32

Microsoft Windows имеет в своем составе утилиту командной строки Rundll32.exe, которая позволяет запускать некоторые команды, заложенные в DLL-файлах.

Данная утилита была разработана для внутреннего пользования программистами Microsoft. Однако богатые возможности этой программы дают повод использовать ее обычными программистами при разработке собственных приложений. Список команд, реализуемых Rundll32.exe, слишком обширен. Поэтому в данной главе будут приведены только наиболее значимые.

Для использования команд данной утилиты во встроенном языке системы "1С:Предприятие" достаточно вызвать оператор `ЗапуститьПриложение`, в параметре которого указать необходимую команду. Например, для открытия панели управления необходимо вызвать следующую команду:

```
ЗапуститьПриложение ("RunDLL32.EXE shell32.dll,Control_RunDLL").
```

При использовании команд rundll32 следует учитывать следующие особенности.

- В разных версиях операционных систем имеется свой набор доступных команд, поэтому в данной главе, при указании команды, будут перечислены операционные системы, в которых она доступна.
- Некоторые команды зависят от установленных в операционной системе компонент. Поэтому возможны ситуации, когда определенные команды будут недоступны, либо результат их выполнения в разных системах будет отличаться.
- Почти все команды должны вводиться с учетом регистра.

Запуск элементов панели управления

С помощью утилиты rundll32 можно запускать элементы панели управления. Все команды, отвечающие за их запуск, находятся в модуле `Control_RunDLL` библиотеки shell32.dll.

В табл. 7.1 приведены наиболее значимые команды, запускающие элементы панели управления.

***Таблица 7.1.** Команды rundll32, запускающие элементы панели управления*

Описание команды	Команда
Открывает диалог **Специальные возможности** на закладке **Экран**. Доступна в Windows 95/98/ME/NT4/2000/XP/2003	`RunDLL32.EXE shell32.dll,Control_RunDLL access.cpl,,3`
Открывает диалог **Специальные возможности** на закладке **Общие**. Доступна в Windows 95/98/ME/NT4/2000/XP/2003	`RunDLL32.EXE shell32.dll,Control_RunDLL access.cpl,,5`
Открывает диалог **Специальные возможности** на закладке **Клавиатура**. Доступна в Windows 95/98/ME/NT4/2000/XP/2003	`RunDLL32.EXE shell32.dll,Control_RunDLL access.cpl,,1`
Открывает диалог **Специальные возможности** на закладке **Мышь**. Доступна в Windows 95/98/ME/NT4/2000/XP/2003	`RunDLL32.EXE shell32.dll,Control_RunDLL access.cpl,,4`
Открывает диалог **Специальные возможности** на закладке **Звук**. Доступна в Windows 95/98/ME/NT4/2000/XP/2003	`RunDLL32.EXE shell32.dll,Control_RunDLL access.cpl,,2`
Открывает диалог **Установка и удаление программ** на закладке **Замена и удаление программ**. Доступна в Windows XP/2003	`RunDLL32.EXE shell32.dll,Control_RunDLL appwiz.cpl,,0`
Открывает диалог **Установка и удаление программ** на закладке **Установка новой программы**. Доступна в Windows 95/98/ME/NT4/2000/XP/2003	`RunDLL32.EXE shell32.dll,Control_RunDLL appwiz.cpl,,1`

Таблица 7.1 (продолжение)

Описание команды	Команда
Открывает диалог **Установка и удаление программ** на закладке **Выбор программ по умолчанию**. Доступна в Windows XP	`RunDLL32.EXE Shell32.dll,Control_ RunDLL appwiz.cpl,,3`
Открывает диалог **Установка и удаление программ** на закладке **Добавление и удаление компонентов Windows**. Доступна в Windows 95/98/ME/NT4/2000/XP/2003	`RunDLL32.EXE shell32.dll,Control_ RunDLL appwiz.cpl,,2`
Установка панели управления из резервной копии. Доступна в Windows 95/98/ME/NT4/2000/XP/2003	`RunDLL32.EXE shell32.dll,Control_ FillCache_RunDLL`
Открыть **Панель управления**. Доступна в Windows 95/98/ME/NT4/2000/XP/2003	`RunDLL32.EXE shell32.dll,Control_ RunDLL`
Открывает диалог **Экран** на закладке **Оформление**. Доступна в Windows 95/98/ME/NT4/2000/XP	`RunDLL32.EXE shell32.dll,Control_ RunDLL desk.cpl,,2`
Открывает диалог **Экран** на закладке **Фон**. Доступна в Windows 95/98/ME/NT4/2000/XP	`RunDLL32.EXE shell32.dll,Control_ RunDLL desk.cpl,,0`
Открывает диалог **Экран** на закладке **Заставка**. Доступна в Windows 95/98/ME/NT4/2000/XP/2003	`RunDLL32.EXE shell32.dll,Control_ RunDLL desk.cpl,,1`
Открывает диалог **Экран** на закладке **Настройка**. Доступна в Windows 95/98/ME/NT4/2000/XP/2003	`RunDLL32.EXE shell32.dll,Control_ RunDLL desk.cpl,,3`
Открывает диалог **Экран** на закладке **Темы**. Доступна в Windows 2003	`RunDLL32.EXE shell32.dll,Control_ RunDLL desk.cpl`
Открывает диалог **Экран** на закладке **Темы**. Доступна в Windows XP	`RunDLL32.EXE shell32.dll,Control_ RunDLL desk.cpl,,-1`
Открыть папку **Шрифты**. Доступна в Windows 95/98/ME/NT4/2000/XP/2003	`RunDLL32.EXE SHELL32.DLL,SHHelpShortcuts_ RunDLL FontsFolder`

Таблица 7.1 (продолжение)

Описание команды	Команда
Открыть папку **Шрифты**. Доступна в Windows 95/98/ME	`RunDLL32.EXE shell32.dll,Control_ RunDLL main.cpl @3`
Открывает диалог **Игровые устройства** на закладке **Контроллеры**. Доступна в Windows 95/98/ME/NT4/2000/XP/2003	`RunDLL32.EXE shell32.dll,Control_ RunDLL joy.cpl`
Открывает диалог **Свойства системы** на закладке **Общие**. Доступна в Windows 95/98/ME/NT/2000/XP	`RunDLL32.EXE shell32.dll,Control_ RunDLL sysdm.cpl @1`
Открывает программу **Диспетчер устройств**. Доступна в Windows 2000/XP/2003	`RunDLL32.EXE devmgr.dll DeviceManager_Execute`
Открывает диалог **Свойства обозревателя**. Доступна в Windows 95/98/ME/NT4/2000/XP/2003	`RunDLL32.EXE shell32.dll,Control_ RunDLL inetcpl.cpl,,#`
Открывает диалог **Свойства обозревателя** на закладке **Общие**. Доступна в Windows 95/98/ME/NT4/2000/XP/2003	`RunDLL32.EXE shell32.dll,Control_ RunDLL inetcpl.cpl,,0`
Открывает диалог **Свойства обозревателя** на закладке **Конфиденциальность**. Доступна в Windows NT4/2000/XP/2003	`RunDLL32.EXE shell32.dll,Control_ RunDLL inetcpl.cpl,,2`
Открывает диалог **Свойства обозревателя** на закладке **Безопасность**. Доступна в Windows NT4/2000/XP/2003	`RunDLL32.EXE shell32.dll,Control_ RunDLL inetcpl.cpl,,1`
Открывает диалог **Клавиатура** на закладке **Языки и раскладки**. Доступна в Windows 2000/XP/2003	`RunDLL32.EXE shell32.dll,Control_ RunDLL main.cpl @1,1`
Открывает диалог **Клавиатура** на закладке **Скорость**. Доступна в Windows 2000/XP/2003	`RunDLL32.EXE shell32.dll,Control_ RunDLL main.cpl @1,0`

Открывает диалог **Телефон и модем** на закладке **Модемы**.
Доступна в Windows 95/98/ME/NT4/2000/XP/2003
	`RunDLL32.EXE SHELL32.DLL,Control_ RunDLL modem.cpl,,add`
	
	`RunDLL32.EXE shell32.dll,Control_ RunDLL modem.cpl`
	
	`RunDLL32.EXE SysDM.cpl,InstallDevice_ RunDLL Modem`

Таблица 7.1 (продолжение)

Описание команды	Команда
Открывает диалог **Свойства мыши** на закладке **Кнопки мыши**. Доступна в Windows 2000/XP/2003	`RunDLL32.EXE shell32.dll,Control_ RunDLL main.cpl @0,0`
Открывает диалог **Свойства мыши** на закладке **Оборудование**. Доступна в Windows XP/2003	`RunDLL32.EXE shell32.dll,Control_ RunDLL main.cpl @0,4`
Открывает диалог **Свойства мыши** на закладке **Опции**. Доступна в Windows XP/2003	`RunDLL32.EXE shell32.dll,Control_ RunDLL main.cpl @0,2`
Открывает диалог **Свойства мыши** на закладке **Указатели**. Доступна в Windows XP/2003	`RunDLL32.EXE shell32.dll,Control_ RunDLL main.cpl @0,1`
Открывает диалог **Свойства мыши** на закладке **Действия**. Доступна в Windows XP/2003	`RunDLL32.EXE shell32.dll,Control_ RunDLL main.cpl @0,3`
Открывает диалог **Звуки и мультимедиа** на закладке **Звуки**. Доступна в Windows 95/98/ME/NT4/2000	`RunDLL32.EXE shell32.dll,Control_ RunDLL mmsys.cpl,,0`
Открывает диалог **Звуки и мультимедиа** на закладке **CD-ROM**. Доступна в Windows 95/98/ME/NT4	`RunDLL32.EXE shell32.dll,Control_ RunDLL mmsys.cpl,,3`
Открывает диалог **Звуки и мультимедиа** на закладке **Устройства**. Доступна в Windows 95/98/ME/NT4	`RunDLL32.EXE shell32.dll,Control_ RunDLL mmsys.cpl,,4`
Открывает диалог **Звуки и мультимедиа** на закладке **MIDI**. Доступна в Windows 95/98/ME/NT4	`RunDLL32.EXE shell32.dll,Control_ RunDLL mmsys.cpl,,2`
Открывает диалог **Звуки и мультимедиа** на закладке **Аудио**. Доступна в Windows 95/98/ME/NT4/2000	`RunDLL32.EXE shell32.dll,Control_ RunDLL mmsys.cpl,,1`
Открывает диалог **Сетевые свойства**. Доступна в Windows 95/98/ME	`RunDLL32.EXE shell32.dll,Control_ RunDLL netcpl.cpl`
Открывает диалог **Настройки ODBC** источников данных. Доступна в Windows 95/98/ME/NT4/2000/XP/2003	`RunDLL32.EXE shell32.dll,Control_ RunDLL odbccp32.cpl`

Таблица 7.1 (продолжение)

Описание команды	Команда
Открывает диалог **Пароли**. Доступна в Windows 95/98/ME	`RunDLL32.EXE shell32.dll,Control_RunDLL password.cpl`
Открывает диалог **Свойства PCMCIA**. Доступна в Windows 95/98/ME	`RunDLL32.EXE shell32.dll,Control_RunDLL main.cpl @4`
Открывает диалог **Управление питанием** на закладке **Схемы**. Доступна в Windows 95/98/ME/NT4/2000/XP	`RunDLL32.EXE 132.dll,Control_RunDLL powercfg.cpl`
Открывает диалог **Электропитание** на закладке **Схемы**. Доступна в Windows NT4/2000/XP	`RunDLL32.EXE shell32.dll,Control_RunDLL ups.cpl`
Открывает диалог **Языки и стандарты** на закладке **Денежная единица**. Доступна в Windows 95/98/ME/NT4/2000/XP/2003	`RunDLL32.EXE shell32.dll,Control_RunDLL intl.cpl,,2`
Открывает диалог **Языки и стандарты** на закладке **Дата**. Доступна в Windows 95/98/ME/NT4/2000	`RunDLL32.EXE shell32.dll,Control_RunDLL intl.cpl,,4`
Открывает диалог **Языки и стандарты** на закладке **Общие**. Доступна в Windows 95/98/ME/NT4/2000	`RunDLL32.EXE shell32.dll,Control_RunDLL intl.cpl,,0`
Открывает диалог **Языки и стандарты** на закладке **Языки и раскладки**. Доступна в Windows 95/98/ME/NT4/2000	`RunDLL32.EXE shell32.dll,Control_RunDLL intl.cpl,,5`
Открывает диалог **Языки и стандарты** на закладке **Числа**. Доступна в Windows 95/98/ME/NT4/2000XP/2003	`RunDLL32.EXE shell32.dll,Control_RunDLL intl.cpl,,1`
Открывает диалог **Языки и стандарты** на закладке **Время**. Доступна в Windows 95/98/ME/NT4/2000	`RunDLL32.EXE shell32.dll,Control_RunDLL intl.cpl,,3`
Открывает диалог **Звуки и мультимедиа**. Доступна в Windows 95/98/ME/NT4	`RunDLL32.EXE shell32.dll,Control_RunDLL mmsys.cpl @1`
Открывает диалог **Звуки и аудиоустройства** на закладке **Аудио**. Доступна в Windows XP/2003	`RunDLL32.EXE shell32.dll,Control_RunDLL mmsys.cpl,,2`

Таблица 7.1 (продолжение)

Описание команды	Команда
Открывает диалог **Звуки и аудиоустройства** на закладке **Оборудование**. Доступна в Windows XP/2003	`RunDLL32.EXE shell32.dll,Control_ RunDLL mmsys.cpl,,4`
Открывает диалог **Звуки и аудиоустройства** на закладке **Звуки**. Доступна в Windows XP/2003	`RunDLL32.EXE shell32.dll,Control_ RunDLL mmsys.cpl,,1`
Открывает диалог **Звуки и аудиоустройства** на закладке **Речь**. Доступна в Windows XP/2003	`RunDLL32.EXE shell32.dll,Control_ RunDLL mmsys.cpl,,3`
Открывает диалог **Звуки и аудиоустройства** на закладке **Громкость**. Доступна в Windows XP/200	`RunDLL32.EXE shell32.dll,Control_ RunDLL mmsys.cpl,,0`
Открывает диалог **Свойства системы** на закладке **Дополнительно**. Доступна в Windows XP/2003	`RunDLL32.EXE shell32.dll,Control_ RunDLL sysdm.cpl,,3`
Открывает диалог **Свойства системы** на закладке **Восстановление системы**. Доступна в Windows XP/2003	`RunDLL32.EXE shell32.dll,Control_ RunDLL sysdm.cpl,,4`
Открывает диалог **Свойства системы** на закладке **Автоматическое обновление**. Доступна в Windows XP	`RunDLL32.EXE shell32.dll,Control_ RunDLL sysdm.cpl,,5`
Открывает диалог **Свойства системы** на закладке **Имя компьютера**. Доступна в Windows XP/2003	`RunDLL32.EXE shell32.dll,Control_ RunDLL sysdm.cpl,,1`
Открывает диалог **Свойства системы** на закладке **Общие**. Доступна в Windows 95/98/ME/NT4/2000/XP/2003	`RunDLL32.EXE shell32.dll,Control_ RunDLL sysdm.cpl,,0`
Открывает диалог **Свойства системы** на закладке **Оборудование**. Доступна в Windows NT4/2000/XP/2003	`RunDLL32.EXE shell32.dll,Control_ RunDLL sysdm.cpl,,2`
Открывает диалог **Свойства системы** на закладке **Сетевая идентификация**. Доступна в Windows NT4/2000	`RunDLL32.EXE shell32.dll,Control_ RunDLL sysdm.cpl,,1`
Открывает диалог **Свойства системы** на закладке **Удаленное использование**. Доступна в Windows XP	`RunDLL32.EXE shell32.dll,Control_ RunDLL sysdm.cpl,,6`

Таблица 7.1 (окончание)

Описание команды	Команда
Открывает диалог **Свойства системы** на закладке **Удаленное использование**. Доступна в Windows 2003	`RunDLL32.EXE shell32.dll,Control_ RunDLL sysdm.cpl,,5`
Открывает диалог **Свойства системы** на закладке **Дополнительно**. Доступна в Windows 2000/XP	`RunDLL32.EXE shell32.dll,Control_ RunDLL sysdm.cpl,,4`
Открывает диалог **Свойства системы** на закладке **Профили пользователей**. Доступна в Windows 95/98/ME/NT4/2000/XP	`RunDLL32.EXE shell32.dll,Control_ RunDLL sysdm.cpl,,3`
Открывает диалог **Телефон и модем**. Доступна в Windows 95/98/ME/NT4/2000/XP/2003	`RunDLL32.EXE shell32.dll,Control_ RunDLL telephon.cpl`
Открывает диалог **Темы** в Windows 95/98 при установке Windows Plus. Доступна в Windows 95/98/ME	`RunDLL32.EXE shell32.dll,Control_ RunDLL themes.cpl`
Открывает диалог **Дата и время**. Доступна в Windows 95/98/ME/NT4/2000/XP/2003	`RunDLL32.EXE shell32.dll,Control_ RunDLL timedate.cpl`
Открывает диалог **Дата и время** на закладке **Часовой пояс**. Доступна в Windows 95/98/ME/NT4/2000/XP/2003	`RunDLL32.EXE shell32.dll,Control_ RunDLL timedate.cpl,,/f`
Открытие папки **Сеть** и **Удаленный доступ к сети**. Доступна в Windows NT4/2000/XP/2003	`RunDLL32.EXE shell32.dll,Control_ RunDLL ncpa.cpl`

Как видно из табл. 7.1, многие команды имеют два параметра, разделенные запятой. Первый параметр обычно указывает на внутренний параметр модуля `Control_RunDLL`, второй — номер закладки, которую необходимо открыть при открытии соответствующего элемента панели управления. Последний параметр можно опустить, при этом будет открыта закладка, использующаяся в диалоге по умолчанию.

В листинге 7.1 приведено несколько примеров запуска элементов панели управления.

Листинг 7.1. Запуск элементов панели управления

```
// Открывает диалог "Свойства системы" на закладке "Дополнительно"
ЗапуститьПриложение("RunDLL32.EXE shell32.dll,Control_RunDLL
sysdm.cpl,,4");
// Открывает диалог "Свойства системы" на закладке по умолчанию
ЗапуститьПриложение("RunDLL32.EXE shell32.dll,Control_RunDLL sysdm.cpl");
// Открывает диалог "Дата и время"
ЗапуститьПриложение("RunDLL32.EXE shell32.dll,Control_RunDLL
timedate.cpl");
// Открывает диалог "Дата и время" на закладке "Часовой пояс"
ЗапуститьПриложение("RunDLL32.EXE shell32.dll,Control_
RunDLL timedate.cpl,,/f");
// Открывает диалог "Свойства мыши"
ЗапуститьПриложение("RunDLL32.EXE shell32.dll,Control_RunDLL main.cpl @0");
// Открывает диалог "Клавиатура"
ЗапуститьПриложение("RunDLL32.EXE shell32.dll,Control_RunDLL main.cpl @1");
```

Запуск мастеров

При работе с операционной системой Microsoft Windows доступно множество мастеров, позволяющих облегчить выполнение различных действий, например, создание нового Dial-Up соединения, подключение сетевых дисков или настройка сети. Все доступные мастера находятся в различных библиотеках, поэтому общего правила их вызовов не существует.

В табл. 7.2 приведены команды запуска основных мастеров.

***Таблица 7.2.** Команды rundll32, запускающие мастеров выполнения различных действий*

Описание команды	Команда
Запуск мастера очистки рабочего стола. Доступна в Windows XP	`RunDLL32.EXE FLDRCLNR.DLL,Wizard_RunDLL`
Запуск мастера создания нового Dial-Up-соединения. Доступна в Windows 95/98/ME/NT4/2000XP	`RunDLL32.EXE RNAUI.DLL,RnaWizard`
Запуск мастера установки сканера или цифровой камеры. Доступна в Windows 95/98/ME/XP	`RunDLL32.EXE SysDM.cpl,InstallDevice_RunDLL ImageRunDLL32.EXE wiashext.dll,AddDeviceWasChosen`

Таблица 7.2 (окончание)

Описание команды	Команда
Запуск мастера добавления в сетевое окружение. Доступна в Windows 2000/XP/2003	`RunDLL32.EXE netplwiz.dll,AddNetPlaceRunDll`
Запуск мастера подключения сетевых дисков. Доступна в Windows 95/98/ME	`RunDLL32.EXE USER.DLL,wnetconnectdialog`
Запуск мастера подключения сетевых дисков. Доступна в Windows NT4/2000/XP/2003	`RunDLL32.EXE shell32.dll,SHHelpShortcuts_ RunDLL Connect`
Запуск мастера сетевой идентификации. Доступна в Windows 2000	`RunDLL32.EXE netplwiz.dll,NetAccWizRunDll`
Запуск мастера настройки сети. Доступна в Windows XP	`RunDLL32.EXE hnet-wiz.dll,HomeNetWizardRunDll`
Запуск мастера новых подключений. Доступна в Windows XP/2003	`RunDLL32.EXE net-shell.dll,StartNCW`
Запуск мастера установки нового принтера. Доступна в Windows 95/98/ME/NT4/2000/XP	`RunDLL32.EXE SHELL32.DLL,SHHelpShortcuts_ RunDLL AddPrinter` `RunDLL32.EXE RunDLL32 SysDM.cpl,InstallDevice_ RunDLL Printer,,0`
Запуск мастера добавления стандартного TCP/IP-порта принтеров. Доступна в Windows 2000	`RunDLL32.EXE tcpmo-nui.dll,LocalAddPortUI`

В листинге 7.2 приведено несколько примеров запуска различных ма-стеров.

Листинг 7.2. Запуск мастеров

```
// Запуск мастера создания нового Dial-Up-соединения
ЗапуститьПриложение("RunDLL32.EXE RNAUI.DLL,RnaWizard");
// Запуск мастера новых подключений
ЗапуститьПриложение("RunDLL32.EXE netshell.dll,StartNCW");
// Запуск мастера установки нового принтера
ЗапуститьПриложение("RunDLL32.EXE SHELL32.DLL,SHHelpShortcuts_
RunDLL AddPrinter");
```

Работа с сетью Интернет

В операционной системе Microsoft Windows существуют множество возможностей работы с сетью Интернет, небольшую часть которых можно использовать через утилиту rundll32.

В табл. 7.3 приведены команды, работающие с интернет-технологиями.

Таблица 7.3. Команды rundll32, работающие с интернет-технологиями

Описание команды	Команда
Установка соединения с Dial-Up-соединением, с именем, указанным в параметре `ConnectionName`. Доступна в Windows 95/98/ME/NT4/2000/XP	`RunDLL32.EXE Rnaui.dll,RnaDial ConnectionName`
Открывает специальные интернет-каналы, где параметр `%1` — полный путь к файлу CDF. Доступна в Windows 95/98/ME/NT4/2000/XP/2003	`RunDLL32.EXE cdfview.dll,OpenChannel %1`
Открывает Internet Explorer на странице авторизации почтового ящика HotMail. Доступна в Windows 95/98/ME/NT4/2000/XP/2003	`RunDLL32.EXE "%ProgramFiles%\ Internet Explorer\ hmmapi.dll",OpenInboxHandler`
Открывает диалог **Восстановления настроек Internet Explorer**. Поддерживается в Internet Explorer версий 5 и 6. Доступна в Windows 95/98/ME/NT4/2000/XP	`RunDLL32.EXE setupwbv.dll, IE5Maintenance "C:\ Program Files\Internet Explorer\Setup\ SETUP.EXE" /g "%SystemRoot%\ IE Uninstall Log.Txt"`
Открывает подписку на специальный интернет-канал, путь к CBA-файлу которого определяется в параметре `%1`. Доступна в Windows 95/98/ME/NT4/2000/XP/2003	`RunDLL32.EXE cdfview.dll,Subscribe %1`
Создание нового письма адресату, указанному в параметре `%1`. При этом запускается почтовая программа, используемая в системе по умолчанию. Доступна в Windows 95/98/ME/NT4/2000/XP/2003	`RunDLL32.EXE url.dll, MailToProtocolHandler %1`
Открытие программы чтения групп новостей, имя сервера указывается в параметре `%1`. Доступна в Windows 95/98/ME/NT4/2000/XP/2003	`RunDLL32.EXE url.dll, NewsProtocolHandler %1`

Таблица 7.3 (окончание)

Описание команды	Команда
Запуск интернет-ресурса с адресом, указанном в параметре %1. Доступна в Windows 95/98/ME/NT4/2000/XP/2003	`RunDLL32.EXE url.dll, FileProtocolHandler %1`
Открытие диалога печати HTML-документа с именем HtmlFileNameAndPath. Доступна в Windows 95/98/ME/NT4/2000/XP/2003	`RunDLL32.EXE mshtml.dll, PrintHTML "HtmlFileNameAndPath"`
Установить соединение по Telnet-адресу, указанному в параметре %1. Доступна в Windows 95/98/ME/NT4/2000/XP/2003	`RunDLL32.EXE url.dll, TelnetProtocolHandler %1`

В листинге 7.3 приведено несколько примеров использования интернет-технологий.

Листинг 7.3. Работа с Интернетом

```
// Создание нового письма автору
ЗапуститьПриложение("RunDLL32.EXE url.dll,MailToProtocolHandler
mav@erpg.ru");
// Запуск интернет-страницы автора
ЗапуститьПриложение("RunDLL32.EXE url.dll,FileProtocolHandler
www.erpg.ru");
// Распечатать заглавную страницу форума автора
ЗапуститьПриложение("RunDLL32.EXE mshtml.dll, PrintHTML
""http://forum.erpg.ru""");
```

Установка и удаление принтера

С помощью rundll32 можно автоматически установить или удалить принтер. Для этих целей используется команда:

```
rundll32 printui.dll,PrintUIEntry [параметры] [@командный файл].
```

Примечание

Данная команда доступна только в операционных системах Windows 2000/XP/2003.

Рассмотрим возможные параметры данной команды.

- `/a[файл]` — имя двоичного файла;
- `/b[имя]` — основное имя принтера;
- `/c [имя]` — UNC-имя компьютера, где выполняется команда;
- `/dl` — удаление локального принтера;
- `/dn` — удаление подключения к сетевому принтеру;
- `/dd` — удаление драйвера принтера;
- `/e` — вывод параметров настройки печати;
- `/f[файл]` — информационный или выходной файл;
- `/ga` — добавление подключений компьютеров к принтеру;
- `/ge` — перечисление подключений компьютеров к принтеру;
- `/gd` — удаление подключений компьютеров к принтеру;
- `/h[арх]` — архитектура драйвера Alpha|Intel|Mips|PowerPC;
- `/ia` — установка драйвера принтера с помощью информационного файла;
- `/id` — установка драйвера принтера с помощью мастера установки драйверов принтеров;
- `/if` — установка принтера с помощью информационного файла;
- `/ii` — установка принтера с помощью мастера установки принтеров и информационного файла;
- `/il` — установка принтера с помощью мастера установки принтеров;
- `/in` — добавление подключения к принтеру;
- `/j[имя]` — имя системы печати;
- `/k` — печать пробной страницы на выбранном принтере (не может использоваться с другими командами установки принтера);
- `/l[путь]` — путь к исходному размещению драйвера принтера;
- `/m[модель]` — имя модели драйвера принтера;
- `/n[имя]` — имя принтера;
- `/o` — показать очередь заданий на печать;
- `/p` — отображение свойств принтера;
- `/q` — скрытый режим без вывода сообщений об ошибках;
- `/r[порт]` — имя порта;
- `/s` — показать свойства сервера;
- `/Ss` — запись параметров принтера в файл;

- `/Sr` — чтение параметров принтера из файла, при этом флаги, используемые при записи или чтении параметров принтера и помещаемые в конце команды, могут принимать значения:
  - `c` — профиль цвета;
  - `d` — данные принтера;
  - `s` — дескриптор безопасности;
  - `g` — глобальный режим;
  - `m` — минимальный набор параметров;
  - `u` — пользовательский режим;
  - `r` — разрешить конфликты имен;
  - `f` — принудительное использование имени;
  - `p` — сопоставить порт;
- `/u` — использовать существующий драйвер принтера, если он установлен;
- `/t[номер]` — номер начальной страницы (с нуля);
- `/v[версия]` — версия драйвера;
- `/w` — запрос драйвера у пользователя, если драйвер не найден в INF-файле;
- `/y` — назначить принтер используемым по умолчанию;
- `/Xg` — получить параметры принтера;
- `/Xs` — установить параметры принтера;
- `/z` — не разрешать автоматический общий доступ к этому принтеру;
- `/Z` — общий доступ к принтеру, используется только вместе с параметром `/if`.
- `@командный файл` — файл аргументов командной строки.

В листинге 7.4 приведено несколько примеров работы с принтером.

Листинг 7.4. Работа с принтером

```
// Запуск свойств сервера
ЗапуститьПриложение("rundll32 printui.dll,PrintUIEntry /s /t1
/n\\machine");
// Запуск свойств принтера
ЗапуститьПриложение("rundll32 printui.dll,PrintUIEntry /p
/n\\machine\printer");
// Локальный запуск мастера установки принтеров
ЗапуститьПриложение("rundll32 printui.dll,PrintUIEntry /il");
// Запуск мастера установки принтеров на компьютере \\machine
ЗапуститьПриложение("rundll32 printui.dll,PrintUIEntry /il /c\\machine");
```

```
// Запуск показа очереди
ЗапуститьПриложение("rundll32 printui.dll,PrintUIEntry /o
/n\\machine\printer");
// Запуск установки с помощью информационного файла
ЗапуститьПриложение("rundll32 printui.dll,PrintUIEntry /if /
b ""Test Printer"" /f %windir%\inf\ntprint.inf /r ""lpt1:"" /
m ""AGFA-AccuSet v52.3""");
// Запуск мастера установки принтеров с помощью информационного файла
ЗапуститьПриложение("rundll32 printui.dll,PrintUIEntry /ii
/f %windir%\inf\ntprint.inf");
// Добавление подключений компьютеров к принтеру
ЗапуститьПриложение("rundll32 printui.dll,PrintUIEntry /ga /c\\machine
/n\\machine\printer /j""LanMan Print Services""");
// Удаление подключений компьютеров к принтеру
ЗапуститьПриложение("rundll32 printui.dll,PrintUIEntry /gd /c\\machine
/n\\machine\printer");
// Перечисление подключений компьютеров к принтеру
ЗапуститьПриложение("rundll32 printui.dll,PrintUIEntry /ge /c\\machine");
// Добавление драйвера принтера с помощью информационного файла
ЗапуститьПриложение("rundll32 printui.dll,PrintUIEntry /ia /c\\machine /m
""AGFA-AccuSet v52.3"" /h ""Intel"" /v ""Windows 2000"" /f %win-
dir%\inf\ntprint.inf");
// Удаление драйвера принтера
ЗапуститьПриложение("rundll32 printui.dll,PrintUIEntry /dd /c\\machine /m
""AGFA-AccuSet v52.3"" /h ""Intel"" /v ""Windows 2000""");
// Назначение принтера используемым по умолчанию
ЗапуститьПриложение("rundll32 printui.dll,PrintUIEntry /y /n
""printer""");
// Указание описания принтера
ЗапуститьПриложение("rundll32 printui.dll,PrintUIEntry /Xs /n ""printer""
comment ""My Cool Printer""");
// Получение параметров принтера
ЗапуститьПриложение("rundll32 printui.dll,PrintUIEntry /Xg /n
""printer""");
// Получение параметров принтера и запись их в файл
ЗапуститьПриложение("rundll32 printui.dll,PrintUIEntry /f ""results.txt""
/Xg /n ""printer""");
// Справка об установке параметров принтера:
ЗапуститьПриложение("rundll32 printui.dll,PrintUIEntry /Xs /n ""printer""
?");
// Запись всех параметров принтера в файл
ЗапуститьПриложение("rundll32 printui.dll,PrintUIEntry /Ss /n ""printer""
/a ""file.dat""");
```

```
// Чтение всех параметров принтера из файла
ЗапуститьПриложение("rundll32 printui.dll,PrintUI /Sr /n ""printer"" /a
""file.dat""");
// Запись сведений принтера уровня 2 в файл
ЗапуститьПриложение("rundll32 printui.dll,PrintUIEntry /Ss /n ""printer""
/a ""file.dat"" 2");
// Восстановление дескриптора безопасности из файла
ЗапуститьПриложение("rundll32 printui.dll,PrintUIEntry /Sr /n ""printer""
/a ""file.dat"" s");
// Восстановление глобального devmode и данных принтера из файла
ЗапуститьПриложение("rundll32 printui.dll,PrintUIEntry /Sr /n ""printer""
/a ""file.dat"" g d");
// Восстановление набора параметров из файла и разрешение имени порта
ЗапуститьПриложение("rundll32 printui.dll,PrintUIEntry /Sr /n ""printer""
/a ""file.dat"" m p");
// Выполнение команды, описанной во внешнем файле command.txt
ЗапуститьПриложение("rundll32 printui.dll,PrintUIEntry @command.txt");
```

Прочие команды

Помимо перечисленных ранее команд в Windows существуют и другие, которые не поддаются классификации. Например, команды, отвечающие за работу с папкой **Портфель**, команды открытия файлов, блокировки компьютера и др. Команды этого типа перечислены в табл. 7.4.

***Таблица 7.4.** Прочие команды rundll32*

Описание команды	Команда
Установить каскадное расположение окон приложений на рабочем столе. Доступна в Windows 95/98/ME	`RunDLL32.EXE USER.DLL,cascadechild`
Установить расположение окон приложений на рабочем столе в виде заголовков. Доступна в Windows 95/98/ME	`RunDLL32.EXE USER.DLL,tilechild`
Создать новую папку **Портфель**, если эта функция установлена в системе. Доступна в Windows 95/98/ME/NT4/2000/XP	`RunDLL32.EXE SYNCUI.DLL,Briefcase_Create`
Запустить заглавную страницу папки **Портфель**. Доступна в Windows 95/98/ME/NT4/2000/XP	`RunDLL32.EXE syncui.dll,Briefcase_Intro`

Таблица 7.4 (продолжение)

Описание команды	Команда
Открыть окно **Порты** панели управления. Доступна в Windows NT4	`RunDLL32.EXE shell32.dll,Control_ RunDLL ports.cpl`
Переключить разрешение экрана. Параметры: `H` — горизонтальное разрешение, например, 640, 800 и т. п.; `V` — вертикальное разрешение, например, 480, 600 и т. п.; `R` — глубина цвета, например, 8 (256 цветов), 16, 32 и т. п. Доступна в Windows 95/98/ME	`RunDLL32.EXE DESKCP16.DLL,QUICKRES_ RUNDLLENTRY HxVxR`
Запуск программы копирования дискет. Доступна в Windows 95/98/ME/NT4/2000/XP	`RunDLL32.EXE DISKCOPY.DLL,DiskCopyRunDll`
Запуск программы форматирования дискеты. Доступна в Windows 95/98/ME/NT4/2000/XP	`RunDLL32.EXE SHELL32.DLL,SHFormatDrive`
Перерисовка (обновление) экрана. Доступна в Windows 95/98/ME	`RunDLL32.EXE USER.DLL,repaintscreen`
Открытие DUN-сервера. Доступна в Windows 95/98/ME	`RunDLL32.EXE rnaserv.dll,CallerAccess`
Запуск программы быстрого поиска (если установлена). Доступна в Windows 95/98/ME/NT4/2000/XP	`RunDLL32.EXE shell32.dll,Control_ RunDLL findfast.cpl`
Запуск диалога открытия файла с именем `FileName`. Доступна в Windows 95/98/ME/NT4/2000/XP/2003	`RunDLL32.EXE SHELL32.DLL,OpenAs_ RunDLL FileName`
Открывает диалог **Отключение и извлечение аппаратного устройства**. Доступна в Windows 2000/XP/2003	`RunDLL32.EXE shell32.dll,Control_ RunDLL hotplug.dll`
Открывает диалог **Упорядочить избранное**. Доступна в Windows 95/98/ME/NT4/2000/XP/2003	`RunDLL32.EXE shdocvw.dll,DoOrganizeFavDlg`

Таблица 7.4 (продолжение)

Описание команды	Команда
Отключение клавиатуры. Доступна в Windows 95/98/ME	`RunDLL32.EXE KEYBOARD,disable`
Открывает Microsoft Exchange профиль. Доступна в Windows 95/98/ME	`RunDLL32.EXE shell32.dll,Control_ RunDLL mlcfg32.cpl`
Открывает Microsoft Postoffice Workgroup Admin. Доступна в Windows 95/98/ME	`RunDLL32.EXE shell32.dll,Control_ RunDLL wgpocpl.cpl`
Установка времени двойного щелчка мыши. Значение `Value` указывается в миллисекундах. Доступна в Windows 95/98/ME	`RunDLL32.EXE USER.DLL, SetDoubleClickTime Value`
Переключает левую и правую кнопки мыши. Доступна в Windows 95/98/ME	`RunDLL32.EXE USER.DLL,SwapMouseButton`
Переключает левую и правую кнопки мыши. Доступна в Windows NT4	`RunDLL32.EXE USER32.DLL,SwapMouseButton`
Запуск диалога отключения от присоединенного сетевого диска. Доступна в Windows 95/98/ME	`RunDLL32.EXE USER.DLL, wnetdisconnectdialog`
Запуск диалога отключения от присоединенного сетевого диска. Доступна в Windows NT4/2000/XP/2003	`RunDLL32.EXE shell32.dll, SHHelpShortcuts_ RunDLL Disconnect`
Запустить программу "Управление сервером". Доступна в Windows NT4	`RunDLL32.EXE shell32.dll,Control_ RunDLL srvmgr.cpl`
Открыть диалог создания нового общего ресурса. Доступна в Windows NT4/2000/XP/2003	`RunDLL32.EXE NTLANUI.DLL,ShareCreate`
Открыть диалог общих папок. Доступна в Windows NT4/2000/XP/2003	`RunDLL32.EXE NTLANUI.DLL,ShareManage`
Открытие диалога создания ярлыка для файла, указанного в параметре `%1`. Доступна в Windows 95/98/ME/NT4/2000/XP/2003	`RunDLL32.EXE AppWiz.Cpl, NewLinkHere %1`
Запуск программы TweakUI, если она установлена в системе. Доступна в Windows 95/98/ME/NT4	`RunDLL32.EXE shell32.dll,Control_RunDLL tweakui.cpl`

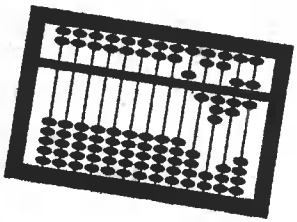
Таблица 7.4 (окончание)

Описание команды	Команда
Открыть файл с изображением, имя которого передано в параметре %1, в программе просмотра изображений и факсов. Доступна в Windows XP/2003	RunDLL32.EXE %SystemRoot%\System32\shimgvw.dll, ImageView_Fullscreen %1
Открытие диалога **О программе**. Доступна в Windows 2000/XP/2003	RunDLL32.EXE SHELL32.DLL, ShellAboutW RunDLL32.EXE SHELL32.DLL, ShellAboutA
Открытие диалога **Применить системные настройки** для перезагрузки или выключения компьютера. Доступна в Windows 95/98/ME	RunDLL32.EXE SHELL.DLL, RestartDialog
Открыть файл, переданный в параметре %1, в зарегистрированном для него приложении. Доступна в Windows 2000/XP/2003	RunDLL32.EXE shell32.dll, ShellExec_RunDLL %1
Открыть диалог **Заблокировать компьютер**, вызываемый при нажатии комбинации клавиш <Ctrl>+<Alt>+<Del>. Доступна в Windows 2000/XP/2003	RunDLL32.EXE user32.dll, LockWorkStation

В листинге 7.5 приведены примеры использования различных команд rundll32.

Листинг 7.5. Прочие команды rundll32

```
// Открытие диалога выбора программы для открытия файла boot.ini
ЗапуститьПриложение("RunDLL32.EXE SHELL32.DLL,OpenAs_RunDLL
c:\boot.ini");
// Открывает диалог "Отключение и извлечение аппаратного устройства"
ЗапуститьПриложение("RunDLL32.EXE shell32.dll,Control_RunDLL hot-
plug.dll");
// Открыть изображение в программе просмотра изображений и факсов
ЗапуститьПриложение("RunDLL32.EXE %SystemRoot%\System32\shimgvw.dll,
ImageView_Fullscreen c:\picture.jpg");
// Открыть файл book1c.doc в зарегистрированном для DOC-файлов приложении
ЗапуститьПриложение("RunDLL32.EXE shell32.dll,ShellExec_RunDLL
c:\book1c.doc");
// Заблокировать компьютер
ЗапуститьПриложение("RunDLL32.EXE user32.dll,LockWorkStation");
```



Глава 8

Интернет-технологии

Работа с электронной почтой

В связи с бурным развитием интернет-программирования поддержка сервиса электронной почты стала весьма распространенной задачей для программистов.

Чтобы эффективно решить задачу доступа к электронной почте, пользователю, в своей программе, необходимо понять основы ее устройства. Электронная почта представляет собой совокупность протоколов, регламентирующих обмен данными по сети, а также программных средств, реализующих эти протоколы и предоставляющих пользователю соответствующие интерфейсы. Основными протоколами являются протоколы передачи (отсылки) и чтения писем, которые работают поверх TCP/IP (Transmission Control Protocol/ Internet Protocol — протокол управления передачей данных Интернет). Для передачи сообщений по TCP-соединению подавляющее большинство программных средств пользуется протоколом SMTP (Simple Mail Transfer Protocol). Для приема почтовых сообщений в Интернете обычно используется протокол POP3 (Post Office Protocol 3).

Непосредственное использование в конфигурациях системы "1С:Предприятие" команд протоколов SMTP и POP3 не представляется возможным. В связи с этим для работы с электронной почтой будут использоваться средства более высокого уровня (внешние компоненты, приложения и стандартные объекты системы).

Использование команды *mailto*

Самый простой способ подготовить письмо для отправки — это просто открыть окно создания нового письма почтового клиента, используемого в системе по умолчанию, воспользовавшись при этом системной командой `mailto:` (листинг 8.1).

Листинг 8.1. Создание нового сообщения через почтового клиента

```
// Открывает окно нового сообщения с заполненными полями
// "Кому", "Тема" и "Текст сообщения".
ЗапуститьПриложение("mailto:mav@erpg.ru;boss@erpg.ru?subject=
Пример%20темы&body=Здавствуйте!%0dДоступ%20открыт.");
// Открывает окно нового сообщения с заполненным адресом получателя
ЗапуститьПриложение("mailto:mav@erpg.ru");
// Открывает окно нового сообщения с заполненным адресом получателя
и полями "Копия" и "Скрытая копия"
ЗапуститьПриложение("mailto:mav@erpg.ru?subject=Пример%20темы&cc=
erpg@erpg.ru&bcc=boss@erpg.ru");
```

Синтаксис команды `mailto`:

```
mailto:<Кому>?subject=<Тема>&cc=<Копия>&bcc=<Скрытаякопия>?&body= <Текст>.
```

Рассмотрим параметры команды:

- `subject` — тема (заголовок) письма;
- `cc` — строка с адресами электронной почты, перечисленными через точку с запятой, указывающая адресатов копии сообщения;
- `bcc` — строка с адресами электронной почты, перечисленными через точку с запятой, указывающая адресатов скрытой копии сообщения;
- `body` — текст сообщения без пробелов и знаков переноса строк.

Внимание

Обратите внимание, что текст, следующий после команды `mailto:`, должен быть без пробелов. Для указания пробела в тексте используется метка `%20`, а для указания символа переноса строки — `%0d`.

Использование интерфейса MAPI

Для совместимости и стандартизации программного доступа к приложениям, которые работают с электронной почтой, Microsoft разработала набор интерфейсов, называемых *MAPI* (Messaging Application Program Interface). Архитектура MAPI описывает так называемую подсистему MAPI, которая обеспечивает взаимодействие клиентских приложений с различными службами почтовой системы, такими как служба хранения информации, транспортная служба и т. д. Все почтовые клиенты, которые предназначены для функционирования в операционной системе Microsoft Windows, должны поддерживать интерфейсы MAPI. С другой стороны, MAPI — это прикладной интерфейс, который был создан для того, чтобы разработчики на раз-

личных языках программирования, в том числе и на встроенном языке системы "1С:Предприятие", имели бы возможность добавлять в свои приложения функцию для работы с электронной почтой. С точки зрения прикладной программы, подсистема MAPI — это набор динамических библиотек, содержащих функции и объектно-ориентированные интерфейсы, благодаря которым взаимодействуют клиентские и серверные части почтовых приложений. Это значит, что, если в системе функционирует почтовый клиент, совместимый с MAPI (а их подавляющее большинство), то можно обратиться к нему через MAPI-интерфейс посредством некоторых вызовов и выполнить нужные операции, например, сформировать и отправить письмо. При этом пользователь обращается к почтовому клиенту, используемому в системе по умолчанию, и всего лишь "поручает" ему выполнить необходимые действия. Как он это сделает, зависит лишь от его настроек. Отсюда возникают и минусы, и плюсы.

Основной минус в том, что MAPI ограничивает возможности программирования набором своих интерфейсов и функциональными возможностями почтового клиента. Более того, различные почтовые клиенты могут иметь различный набор команд.

Плюсы состоят в том, что всегда можно найти отправленное системой письмо в папке **Исходящие** или **Отправленные**, а полученное — в папке **Входящие** почтового клиента. При этом не нужно заботиться о подключении к Интернету (почтовый клиент сам все отправит при первом подключении).

Корпорация Microsoft поставляет элемент управления MSMAPI32.OCX, содержащий простые компоненты ActiveX, позволяющие работать с MAPI из среды любых языков программирования, поддерживающих OLE-автоматизацию. ActiveX содержит два объекта: `MAPISession` и `MAPIMessage`. `MAPISession` отвечает за установление и окончание сеанса связи с провайдером Интернета. Его функции и свойства позволяют задать параметры установления соединения, а затем, при необходимости, разорвать его. `MAPIMessages` предоставляет функции и свойства для работы с почтовыми сообщениями.

Оба объекта можно создать в конфигурации системы "1С:Предприятие". Для версии 7.7 — используется функция `СоздатьОбъект`, а для версии 8.0 — `Новый COMОбъект`.

Для инициализации сеанса связи (сессии) необходимо вызвать функцию `SignOn` объекта `MAPISession`, а для того чтобы использовать объект `MAPIMessages`, нужно связать его с объектом `MAPISession` (листинг 8.2). Для завершения сессии вызывается функция `SignOff` объекта `MAPISession`.

Листинг 8.2. Инициализация сеанса связи MAPI

```
// Элемент управления Session
SessionCtrl = СоздатьОбъект("MSMAPI.MAPISession");
// Элемент управления Messages
```

```
MessageCtrl = СоздатьОбъект("MSMAPI.MAPIMessages");
// Задаем имя пользователя
SessionCtrl.UserName = "MyName";
// Задаем пароль пользователя
SessionCtrl.Password = "MyPassword";
// Открываем сессию
SessionCtrl.SignOn();
// Связываем элементы управления, используя идентификатор сессии
MessageCtrl.SessionID = SessionCtrl.SessionID;
// ...
// Закрываем сессию
SessionCtrl.SignOff();
```

Свойства `UserName` и `Password` объекта `MAPISession` задают имя профиля и пароль доступа к нему, если в качестве почтовой системы используется Microsoft Mail. Если же применяется Microsoft Outlook, эти свойства ни на что не влияют, а в качестве активной учетной записи используется учетная запись, выставленная по умолчанию.

Для чтения входящих писем используется метод `Fetch`, после вызова которого заполняется коллекция сообщений объекта `MAPIMessages`. После заполнения коллекции переменная `MessageCtrl` содержит письмо с индексом, который определяется свойством `MsgIndex`.

В листинге 8.3 приведен пример вывода содержания писем, находящихся в папке **Входящие**.

Листинг 8.3. Чтение входящих писем через интерфейс MAPI

```
// Элемент управления Session
SessionCtrl = СоздатьОбъект("MSMAPI.MAPISession");
// Элемент управления Messages
MessageCtrl = СоздатьОбъект("MSMAPI.MAPIMessages");
// Открываем сессию
SessionCtrl.SignOn();
// Связываем элементы управления, используя идентификатор сессии
MessageCtrl.SessionID = SessionCtrl.SessionID;

// Выбирать для отображения все почтовые сообщения.
// Если вы хотите отображать только непрочитанные сообщения,
// нужно присвоить свойству FetchUnreadOnly значение -1 (Истина)
```

```
MessageCtrl.FetchUnreadOnly = 0; // Ложь
// Выбрать письма
MessageCtrl.Fetch();
Для Ном = 0 по MessageCtrl.MsgCount - 1 Цикл
  // Установить индекс для сообщения
  MessageCtrl.MsgIndex = Ном;
  // Сформировать текст из темы и тела почтового сообщения
  ТекстПисьма = "Тема: " + MessageCtrl.MsgSubject + "Текст: " +
  MessageCtrl.MsgNoteText;
  Сообщить(ТекстПисьма);
КонецЦикла;

// Закрываем сессию
SessionCtrl.SignOff();
```

С помощью данного кода будет получен список всех тем и текстов писем из папки **Входящие**. Свойства `MsgSubject` и `MsgNoteText` объекта `MAPIMessages` возвращают тему и содержание письма. Возможно получить доступ и к другой информации, заложенной в письмах. Например, для вывода имени и электронного адреса отправителя можно использовать свойства `MsgOrigDisplayName` и `MsgOrigAddress` соответственно, свойство `MsgDateReceived` будет содержать дату получения письма.

Для создания письма снова используется объект `MAPIMessages`. Перед отправкой нужно задать несколько необходимых параметров и, если нужно, присоединить файлы (листинг 8.4).

Листинг 8.4. Отправка письма через интерфейс MAPI

```
// Элемент управления Session
SessionCtrl = СоздатьОбъект("MSMAPI.MAPISession");
// Элемент управления Messages
MessageCtrl = СоздатьОбъект("MSMAPI.MAPIMessages");

// Открываем сессию
SessionCtrl.SignOn();
// Связываем элементы управления, используя идентификатор сессии
MessageCtrl.SessionID = SessionCtrl.SessionID;

// Подготовиться к отправке сообщения
MessageCtrl.Compose();
```

```
// Установить e-mail получателя
MessageCtrl.RecipAddress = "mav@erpg.ru";
// Установить тему сообщения
MessageCtrl.MsgSubject = "Заголовок тестового письма";
// Заполнить тело сообщения
MessageCtrl.MsgNoteText = "Это тестовое письмо";

// Присоединяем к письму два файла
MessageCtrl.AttachmentIndex = 0;
MessageCtrl.AttachmentPathName = ("C:\file1.txt");
MessageCtrl.AttachmentIndex = 1;
MessageCtrl.AttachmentPathName = ("C:\file2.txt");

// Отправить без отображения стандартного диалога отправки сообщения
MessageCtrl.Send(0);

// Закрываем сессию
SessionCtrl.SignOff();
```

Примечание

Указывая в качестве параметра метода `Send` значение "Истина" (`-1`), можно разрешить показ окна Создание нового письма.

Использование компоненты V7Plus.dll

Внешняя компонента V7Plus.dll состоит из четырех объектов: `V7Mail`, `V7SysInfo`, `V7HttpReader` и `XMLParser`.

Примечание

В данной главе будет рассматривается только объект работы с электронной почтой — `V7Mail`, который позволяет работать с любым почтовым клиентом, работающим через MAPI (например, Microsoft Outlook Express или Microsoft Outlook).

Инициализация внешней компоненты осуществляется с помощью метода `ЗагрузитьВнешнююКомпоненту("V7Plus.dll")`, а для доступа к свойствам и методам объекта необходимо вызвать функцию `СоздатьОбъект("AddIn.V7Mail")`, которая возвращает ссылку на объект `V7Mail`.

Рассмотрим общий алгоритм получения писем:

1. Установка соединения с помощью метода `Подключиться`.

2. Инициализация процесса приема сообщений с помощью метода ВыбратьСообщения.
3. Организация цикла получения сообщений с помощью метода ПолучитьСообщение.
4. С помощью методов ПрочитатьСообщение и ПолучитьФайл можно получить доступ к тексту сообщения и присоединенным файлам.
5. С помощью свойств Заголовок и Текст можно получить доступ к заголовку сообщения и его тексту.
6. Завершение работы с почтой происходит с помощью метода Отключиться.

Пример реализации данного алгоритма приведен в листинге 8.5.

Листинг 8.5. Чтение входящих писем через компоненту V7Plus.dll

```
Перем Путь, ИмяФайла;

// Загрузка компоненты V7Plus.dll
ЗагрузитьВнешнююКомпоненту("V7Plus.dll");
Почта = СоздатьОбъект("AddIn.V7Mail");

ТекстДок=СоздатьОбъект("Текст");
// Инициализируем сеанс работы с почтой в соответствии
// с выбранным на закладке свойств компоненты типом почты и параметрами
   подключения
Почта.Подключиться();
// Инициализируем процесс получения сообщений
Почта.ВыбратьСообщения();
Пока Почта.ПолучитьСообщение() = 1 Цикл
  // Получаем адрес, с которого было отправлено сообщение
  АдресОтправителя=Почта.АдресОтправителя();
  // Тема сообщения
  Заголовок=Почта.Заголовок;
  // Получаем содержимое сообщения (текст и присоединенные файлы)
  Почта.ПрочитатьСообщение(0);
  // Текст сообщения
  Текст=Почта.Текст;
  // Получить имя и каталог следующего присоединенного файла сообщения
  Пока Почта.ПолучитьФайл(Путь, ИмяФайла) = 1 Цикл
    ТекстДок.Открыть(Путь+ИмяФайла);
    ТекстДок.ТолькоПросмотр(1);
```

```
    ТекстДок.Показать(Заголовок);
  КонецЦикла;
КонецЦикла;

// Завершает сеанс работы с почтой
Почта.Отключиться();
```

Алгоритм отправки сообщения будет состоять из следующих шагов.

1. Установка соединения с помощью метода `Подключиться`.
2. Инициализация процесса создания нового сообщения с помощью метода `НовоеСообщение`.
3. С помощью метода `ДобавитьАдрес` можно задать один или несколько адресов получателей.
4. С помощью метода `ДобавитьФайл` можно прикрепить один или несколько файлов к сообщению.
5. С помощью свойств `Заголовок` и `Текст` можно задать заголовок сообщения и его текст.
6. Отправка сообщения происходит посредством вызова метода `Послать`.
7. Завершение работы с почтой происходит с помощью метода `Отключиться`.

В листинге 8.6 приведен пример отправки сообщения с помощью компоненты V7Plus.dll.

Листинг 8.6. Отправка письма через компоненту V7Plus.dll

```
// Загрузка компоненты V7Plus.dll
ЗагрузитьВнешнююКомпоненту("V7Plus.dll");
Почта = СоздатьОбъект("AddIn.V7Mail");

// Метод Подключиться инициализирует сеанс работы с почтой MAPI
// с соответствующими параметрами подключения и паролем.
Почта.Подключиться("MAPI","", "");
// Инициализируем создание нового сообщения
Почта.НовоеСообщение();
// Добавляем адрес в список адресов получателей
Почта.ДобавитьАдрес("mav@erpg.ru");
// Добавляем имя нового файла в список присоединенных файлов сообщения
Почта.ДобавитьФайл("C:\file.txt");
// Тема сообщения
```

```
Почта.Заголовок="Привет!";
// Текст сообщения
Почта.Текст="Добрый день! Жду результатов совещания.";
// Отправить сообщение
Почта.Послать();
// Завершает сеанс работы с почтой
Почта.Отключиться();
```

Примечание

Подробное описание всех методов и свойств объекта `V7Mail` можно найти в документации к компоненте.

Так как объект `V7Mail` компоненты V7Plus.dll работает через интерфейс MAPI, то при работе с ней сохраняются все особенности — плюсы и минусы использования интерфейса MAPI, которые были описаны ранее.

Использование компоненты Rom-Mail.dll

Все вышеописанные методы интеграции с электронной почтой используют интерфейс MAPI. Для работы напрямую с почтовыми серверами через протоколы POP3 и SMTP в системе "1С:Предприятие 7.7" лучше всего использовать компоненту Rom-Mail.dll, которая распространяется бесплатно автором Олегом Ризвановым.

Компонентой поддерживаются следующие функции:

- стандарты MIME, UU-ENCODE;
- кодирование Base64 и QuotedPrintable;
- основные таблицы кодировок кириллицы: DOS, Windows-1251, KOI8-R, MAC;
- сообщения, состоящие из нескольких частей (multipart);
- отправка, получение, сохранение вложенных файлов (attachment);
- отправка тела сообщения в форматах TEKCT(plain/text), HTML(plain/html), RTF(richable);
- доступ как к указанной строке тела сообщения (построчное чтение), так и получение всего тела сообщения сразу;
- использование полей CC (копия) и BCC (скрытая копия);
- настройка параметров соединения на странице свойств компоненты;
- сохранение соединения (keep alive);
- настройка аутентификации на странице свойств;

- возможность установки произвольного значения полей АдресОтправителя (From) и ИмяОтправителя (FromName), а также оставлять эти значения пустыми;
- выбор вида аутентификации при входе на SMTP-сервер;
- возможность выбора — забирать или оставлять сообщения на сервере;
- возможность установки каталога для сохранения вложений;
- пометка/снятие пометки на удаление;
- 20 направлений перекодировки в различные таблицы кодировок;
- поддержка функции контроля соединения (Ping).

Внешняя компонента ROM-Mail.dll содержит только один объект AddInMail. В листинге 8.7 приведен пример загрузки внешней компоненты.

Листинг 8.7. Загрузка внешней компоненты ROM-Mail.dll

```
// Загрузка внешней компоненты
Если ЗагрузитьВнешнююКомпоненту("ROM-Mail.dll") = 1 Тогда
  // Создание объекта AddInMail
  ВК = СоздатьОбъект("AddIn.AddInMail");
Иначе
  Сообщить("Не удалось загрузить ROM-Mail.dll");
  Возврат;
КонецЕсли;
```

Для чтения писем с POP3-сервера необходимо заполнить параметры подключения к нему и вызвать метод Подключиться с параметром "Прием" (листинг 8.8). После подключения необходимо в цикле выбрать все сообщения.

Листинг 8.8. Чтение писем с POP3-сервера

```
// Загрузка внешней компоненты
Если ЗагрузитьВнешнююКомпоненту("ROM-Mail.dll") = 1 Тогда
  // Создание объекта AddInMail
  ВК = СоздатьОбъект("AddIn.AddInMail");
Иначе
  Сообщить("Не удалось загрузить ROM-Mail.dll");
  Возврат;
КонецЕсли;
// NetBIOS-имя или IP-адрес сервера приема сообщений (POP3)
```

```
ВК.СерверПриема = "pop.server.ru";
// Порт сервера приема сообщений
ВК.ПортПриема = 110;
// Логин для аутентификации на POP3-сервере
ВК.Логин = "user";
// Пароль для аутентификации на POP3-сервере
ВК.Пароль = "password";
// 1 - оставлять письма на POP3-сервере, 0 - удалять письма с сервера
ВК.ОставлятьНаСервере = 1;
// Разрешить/запретить показ модальных окон предупреждений
ВК.ПоказыватьПредупреждения = 0;
// Выполняет подключение к серверу
Если ВК.Подключиться("Прием")=0 Тогда
  Предупреждение("Не удалось подключиться к серверу!");
  Возврат;
КонецЕсли;
// Выборка писем
Для Индекс = 1 По ВК.КоличествоСообщений() Цикл
  // Получает сообщение с заданным номером
  ВК.ПолучитьСообщение(Индекс);
  // Заполняем данные письма в таблицу значений
  ТЗ.НоваяСтрока();
  ТЗ.Отправитель = ВК.ИмяОтправителя;
  ТЗ.Адрес = ВК.АдресОтправителя;
  ТЗ.Тема = ВК.Тема;
  ТЗ.Получатель = ВК.АдресПолучателя;
  ТЗ.MessageID = ВК.IDMessage; // Идентификатор сообщения
  ТЗ.Копия = ВК.СС;
  ТЗ.Тело = ВК.Тело;
  // Имя файла-вложения или список файлов (строка, с разделителем ";")
  ТЗ.Вложения = ВК.Вложения;
  ТЗ.Дата = ВК.ДатаОтправки;
  ТЗ.Размер = ВК.Размер; // Размер сообщения в байтах
КонецЦикла;
// Разрывает установленные соединения
ВК.Отключиться();
```

В данном примере вся информация о письмах добавляется в таблицу значений с идентификатором ТЗ.

Примечание

Инициализация данной таблицы значений лежит за рамками данного примера.

При получении тела письма оно может быть в произвольной кодировке. Чтобы преобразовать его к удобочитаемому виду, можно воспользоваться методом `Перекодировать`, первым параметром которого указывается текст письма, а вторым — номер направления перекодировки:

1. Из ALT в ISO.
2. Из ALT в KOI.
3. Из ALT в MAC.
4. Из ALT в WIN.
5. Из ISO в ALT.
6. Из ISO в KOI.
7. Из ISO в MAC.
8. Из ISO в WIN.
9. Из KOI в ALT.
10. Из KOI в ISO.
11. Из KOI в MAC.
12. Из KOI в WIN.
13. Из MAC в ALT.
14. Из MAC в ISO.
15. Из MAC в KOI.
16. Из MAC в WIN.
17. Из WIN в ALT.
18. Из WIN в ISO.
19. Из WIN в KOI.
20. Из WIN в MAC.

Метод `Перекодировать` возвращает строку в кодировке, соответствующей номеру направления преобразования.

В цикле выборки писем, после получения очередного письма, при наличии вложенных файлов, можно сохранить их на жестком диске. Ниже приведен пример сохранения всех вложенных файлов в корневом каталоге диска (листинга 8.9).

Листинг 8.9. Сохранение вложенных файлов на диск

```
// ...
Для Индекс = 1 По ВК.КоличествоСообщений() Цикл
  ВК.ПолучитьСообщение(Индекс);
  // ...
  Если ПустоеЗначение(ВК.Вложения) = 0 Тогда
    // Получаем тело сообщения с заданным номером
    ВК.ПолучитьТело(Индекс);
    // Выбираем все вложенные файлы письма
    Для НомерФайла = 1 По ВК.КоличествоФайлов() Цикл
      // Получение строки с именем файла-вложения
      ИмяФайла = ВК.ПолучитьФайл(НомерФайла);
      // Сохранение файла
      ВК.СохранитьФайл(НомерФайла, "C:\" + ИмяФайла);
    КонецЦикла;
  КонецЕсли;
КонецЦикла;
```

Для отправки письма через SMTP-сервер необходимо подключиться к почтовому серверу, заполнить атрибуты письма и вызвать метод `Отправить`.

В листинге 8.10 приведен пример отправки одиночного письма.

Листинг 8.10. Отправка письма через SMTP-сервер

```
// Загрузка внешней компоненты
Если ЗагрузитьВнешнююКомпоненту("ROM-Mail.dll") = 1 Тогда
  // Создание объекта AddInMail
  ВК = СоздатьОбъект("AddIn.AddInMail");
Иначе
  Сообщить("Не удалось загрузить ROM-Mail.dll");
  Возврат;
КонецЕсли;
// NetBIOS-имя или IP-адрес сервера отправки сообщений (SMTP)
ВК.СерверОтправки = "smtp.server.ru";
// Порт сервера отправки сообщений (SMTP)
ВК.ПортОтправки = 25;
// Логин для аутентификации на SMTP-сервере
ВК.Логин = "user";
```

```
// Пароль для аутентификации на SMTP-сервере
ВК.Пароль = "password";
// SMTP-серверу требуется проверка подлинности
ВК.АутентификацияПриОтправке = 1;
// Выполняет подключение к серверу
Если ВК.Подключиться("Отправка") = 0 Тогда
  Предупреждение("Не удалось подключиться к серверу!");
  Возврат;
КонецЕсли;

ВК.АдресОтправителя = "mav@erpg.ru";
ВК.ИмяОтправителя = "Андрей Михайлов";
ВК.АдресПолучателя = "Получатель1 <user1@server.ru>; Получатель2
<user2@server.rua>";
ВК.Копия = "Получатель3 <user3@server.ru>";
ВК.СкрытаяКопия = "Получатель4 <user4@server.ru>";
ВК.Тема = "Тема письма";
ВК.Тело = "Пример отправки письма";
ВК.Вложения = "C:\Temp\example.ert; C:\Temp\Readme.doc";
ВК.Кодировка="windows-1251";
ВК.ТипПисьма = 0; // text/plain
ВК.Важность = 1; // Высокая
ВК.ПодтверждениеДоставки = 0;
ВК.ПодтверждениеПрочтения = 1;
// Выполнить отправку сообщений с SMTP-сервера
Если ВК.Отправить() = 1 Тогда
  Предупреждение("Сообщение отправлено!");
Иначе
  Предупреждение("Не удалось отправить сообщение!");
КонецЕсли;
// Разрывает установленные соединения
ВК.Отключиться();
```

Данный пример демонстрирует отправку письма текстового типа. Для его отправки в формате `application/rtf` или `text/html` необходимо в свойстве `ТипПисьма` указать соответственно значение `1` или `2` и выполнить метод `ЗагрузитьИзФайла`, который загружает тело письма из указанного файла.

В компоненте ROM-Mail.dll реализован дополнительный метод `Ping`, который возвращает процент потерь. После выполнения метода свойство `PingText` содержит результат выполнения метода `Ping` (листинг 8.11).

Листинг 8.11. Использование метода `Ping`

```
// NetBIOS-имя или IP-адрес хоста
Host = "erpg.ru";
// Загрузка внешней компоненты
Если ЗагрузитьВнешнююКомпоненту("ROM-Mail.dll") = 1 Тогда
  // Создание объекта AddInMail
  ВК = СоздатьОбъект("AddIn.AddInMail");
Иначе
  Сообщить("Не удалось загрузить ROM-Mail.dll");
  Возврат;
КонецЕсли;
// Отправляем пакет размером 72 байта по протоколу ICMP
ВК.Ping(Host);
// Вывести результат выполнения метода Ping
Сообщить(ВК.PingText);
```

Результатом выполнения данного кода будет сообщение вида:

"IP-адрес: 194.135.22.201; Байт отправлено: 72; Байт получено: 72; TTL: 128; Потерь 0%".

Компонента ROM-Mail.dll имеет страницу свойств, атрибуты которой устанавливаются из меню **Сервис | Параметры** системы "1С:Предприятие". При использовании страницы свойств создавать объект и инициализировать компоненту следует в глобальном модуле.

Использование объекта *Почта*

В системе "1С:Предприятие" (версия 8.0) для работы с электронной почтой существует объект `Почта`, который является аналогом объекта `V7Mail` в системе "1С:Предприятие" (версия 7.7) в том смысле, что он тоже работает по технологии MAPI.

Рассмотрим все объекты, которые необходимо знать при работе с электронной почтой.

- `Почта` — основной объект, предназначенный для подключения, отключения соединения, а также для отправки и получения писем.
- `ПочтовоеСообщение` — содержит информацию о письме, такую как: тема, текст письма, адресаты, вложения и т. п.
- `ПочтовыйАдрес` — содержит информацию о почтовом адресе (электронный адрес и имя пользователя).

- ПочтовыеАдрес — содержит коллекцию элементов ПочтовыйАдрес.
- ПочтовоеВложение — содержит информацию о вложенном файле (данные и имя файла).
- ПочтовыеВложения — содержит коллекцию элементов ПочтовоеВложение.

Для подключения к почтовому клиенту используется метод Подключиться объекта Почта. Метод Подключиться имеет два параметра. Первый — строковое выражение, обозначающее название профиля для подключения, если параметр не указан, то используется профиль пользователя, установленный по умолчанию. Второй параметр — пароль пользователя, если он требуется при подключении к серверу.

В листинге 8.12 приведен пример подключения к почтовому серверу.

Листинг 8.12. Подключение к почтовому серверу

```
// Создает объект Почта
Почта = Новый Почта;
// Подключение к почтовому серверу
Почта.Подключиться("MAV",);
//...
// Оключение от почтового сервера
Почта.Отключиться();
```

Для получения почты с сервера используется метод Выбрать объекта Почта. Метод имеет два параметра. Первый — признак выборки только непрочитанных сообщений. Второй — признак выборки только конвертов сообщений (т. е. без вложений).

В листинге 8.13 приведен пример выборки писем из профиля, используемого по умолчанию, и записи информации о письмах в табличное поле ТабличноеПолеИнтернетПочта.

Листинг 8.13. Чтение сообщений

```
// Создает объект Почта
Почта = Новый Почта;
// Подключение к почтовому серверу для получения почты
Почта.Подключиться(,);
МассивПисем = Почта.Выбрать(Ложь, Истина);
// Перебираем массив писем
Для каждого Письмо Из МассивПисем Цикл
```

```
    // Добавляем новую строку в табличное поле
    СтрокаТП = ТабличноеПолеИнтернетПочта.Добавить();
    Попытка
      СтрокаТП.ОтКого = Письмо.Отправитель.Адрес;
    Исключение
      СтрокаТП.ОтКого = Письмо.Отправитель;
    КонецПопытки;
    Получатели = "";
    // Перебираем всех получателей
    Для каждого Получатель Из Письмо.Получатели Цикл
      Получатели = Получатели + Получатель.Пользователь + " <" +
Получатель.Адрес + "> ";
    КонецЦикла;
    СтрокаТП.Кому = Получатели;
    СтрокаТП.Тема = Письмо.Тема;
    СтрокаТП.Текст = Письмо.Текст;
    // Перебираем вложения
    Для каждого Вложение Из Письмо.Вложения  Цикл
    // Сохраняем вложение в папке C:\Mail\
    Вложение.Данные.Записать("C:\Mail\" + Вложение.Имя);
    КонецЦикла;
КонецЦикла;
// Оключение от почтового сервера
Почта.Отключиться();
```

В приведенном примере есть несколько особенностей:

1. При обращении к свойству `Отправитель` объекта `Письмо` используется конструкция `Попытка...Исключение...КонецПопытки`. Это необходимо вследствие того, что свойство `Отправитель` может возвращать значение типа `Строка` адреса или `ПочтовыйАдрес`. Вместо конструкции `Попытка...Исключение...КонецПопытки` можно анализировать тип возвращаемого значения.
2. Все файлы вложений автоматически сохраняются в папке "C:\Mail\". Вместо этого можно сохранять данные вложений в реквизиты объектов системы "1С:Предприятие 8.0" типа `ХранилищеЗначения`.

Для отправки писем используется объект `ПочтовоеСообщение`. В листинге 8.14 приведен пример создания почтового сообщения и его отправки.

Листинг 8.14. Отправка почтового сообщения

```
// Создаем почтовое сообщение
Письмо = Новый ПочтовоеСообщение;
Письмо.Тема = Тема;
Письмо.Текст = Текст;
// Добавляем адресата
Письмо.Получатели.Добавить("mav@erpg.ru");
// Добавляем отправителя
Письмо.Отправитель = "boss@erpg.ru";
// Добавляем файл
Письмо.Вложения.Добавить("c:\plan.txt");
// Добавляем произвольные двоичные данные
Архив = Новый ДвоичныеДанные("c:\trade.zip");
Письмо.Вложения.Добавить(Архив,"trade.zip");
// Создаем объект Почта
Почта = Новый Почта;
// Отсылаем письмо
Почта.Послать(Письмо);
```

Примечание

Полный список свойств и методов каждого объекта приведен в документации.

Использование объекта *ИнтернетПочта*

Объект `ИнтернетПочта` присутствует только в системе "1С:Предприятие" (версия 8.0) и позволяет работать с электронной почтой напрямую через сервер электронной почты. Другими словами — работа данного объекта не зависит от установленного почтового клиента.

Объект предназначен для доступа к почтовым серверам с целью отправки и получения сообщений. В отличие от объекта `Почта`, который работает по технологии MAPI и требует от пользователя установленного почтового клиента, объект `ИнтернетПочта` использует интернет-протоколы SMTP и POP3.

Рассмотрим используемые в данном режиме объекты.

- `ИнтернетПочта` — основной объект, предназначенный для подключения и отключения от почтового сервера, а также для выборки писем с почтового сервера и отправки сообщений.
- `ИнтернетПочтовыйПрофиль` — содержит информацию об учетной записи пользователя (имя POP3, SMTP-серверов, имя пользователя, пароль и т. п.).

- ИнтернетПочтовоеСообщение — содержит информацию о письме (тема, дата получения, дата отправления, вложения, почтовые адреса отправителя и получателей и т. д.).
- ИнтернетПочтовыйАдрес — содержит информацию о почтовом адресе (электронный адрес и имя пользователя).
- ИнтернетПочтовыеАдреса — содержит коллекцию элементов ИнтернетПочтовыйАдрес.
- ИнтернетТекстПочтовогоСообщения — содержит текст почтового сообщения и его тип (HTML, простой или размеченный текст).
- ИнтернетТекстыПочтовогоСообщения — содержит коллекцию элементов ИнтернетТекстПочтовогоСообщения.
- ИнтернетПочтовоеВложение — содержит информацию о вложенном файле (данные и имя файла).
- ИнтернетПочтовыеВложения — содержит коллекцию элементов ИнтернетПочтовоеВложение.

Для подключения к почтовому серверу используется метод Подключиться, в параметре которого передается значение типа ИнтернетПочтовыйПрофиль.

В листинге 8.15 приведен пример создания профиля пользователя и подключения к почтовому серверу.

Листинг 8.15. Подключение к почтовому серверу

```
Профиль = Новый ИнтернетПочтовыйПрофиль;
// Адрес SMTP-сервера
Профиль.АдресСервераSMTP = "smtp.erpg.ru";
// Адрес POP3-сервера
Профиль.АдресСервераPOP3 = "pop.erpg.ru";
// Порт протокола SMTP
Профиль.ПортSMTP = 25;
// Порт протокола POP3
Профиль.ПортPOP3 = 110;
// Логин пользователя
Профиль.Пользователь = "boss";
// Логин пользователя для аутентификации на SMTP-сервере
Профиль.ПользовательSMTP = "boss";
// Пароль доступа к почтовому ящику
Профиль.Пароль = "neskaju";
// Пароль пользователя для аутентификации на SMTP-сервере
```

```
Профиль.ПарольSMTP = "toje_neskaju";
// Время ожидания удачного исполнения операции в секундах
Профиль.ВремяОжидания = 30;

Почта = Новый ИнтернетПочта;
// Подключение к почтовому серверу
Попытка
   Почта.Подключиться(ПолучитьПрофиль());
Исключение
   Сообщить(ОписаниеОшибки());
   Возврат;
КонецПопытки;

//...
// Отключение от почтового сервера
Почта.Отключиться();
```

Примечание

При подключении к почтовому серверу используется конструкция `Попытка...Исключение...КонецПопытки`. **Это необходимо сделать для подстраховки от неудачного подключения.**

Для получения писем с почтового сервера используется метод `Выбрать` объекта `ИнтернетПочта`, который возвращает массив объектов типа `ИнтернетПочтовоеСообщение`.

В листинге 8.16 приведен пример получения писем с почтового сервера, профиль для которого был создан ранее.

Листинг 8.16. Получение писем с почтового сервера

```
Почта = Новый ИнтернетПочта;
// Подключение к почтовому серверу
Попытка
   Почта.Подключиться(Профиль);
Исключение
   Сообщить(ОписаниеОшибки());
   Возврат;
КонецПопытки;

ТабличноеПолеИнтернетПочта.Очистить();
```

```
// Массив объектов типа ИнтернетПочтовоеСообщение
Сообщения = Почта.Выбрать(Ложь);
Для каждого Сообщение Из Сообщения Цикл
  // Добавляем строку в табличную часть
  СтрокаТП = ТабличноеПолеИнтернетПочта.Добавить();
  // Перебираем коллекцию получателей сообщения
  КомуВходящие = "";
  Для каждого Получатель Из Сообщение.Получатели Цикл
    КомуВходящие = КомуВходящие + Получатель.Пользователь + " <" +
    Получатель.Адрес + "> ";
  КонецЦикла;
  СтрокаТП.Кому = КомуВходящие;
  СтрокаТП.ОтКого = Сообщение.Отправитель.Пользователь + " <" +
  Сообщение.Отправитель.Адрес + "> ";
  СтрокаТП.Тема = Сообщение.Тема;
  // Перебираем коллекцию текстов сообщения
  ТекстВходящие = "";
  Для каждого ТекстСообщения Из Сообщение.Тексты Цикл
    ТекстВходящие = ТекстВходящие + ТекстСообщения.Текст;
  КонецЦикла;
  СтрокаТП.Текст = ТекстВходящие;
  СтрокаТП.ДатаОтправления = Сообщение.ДатаОтправления;

  // Перебираем коллекцию вложений
  Для каждого Вложение Из Сообщение.Вложения Цикл
    // Сохраняем вложение в папке C:\Mail\
    Вложение.Данные.Записать("C:\Mail\" + Вложение.Имя);
  КонецЦикла;
КонецЦикла;

// Отключение от почтового сервера
Почта.Отключиться();
```

В листинге 8.16 информация о письмах заносится в табличное поле `ТабличноеПолеИнтернетПочта`, расположенное на форме.

Для отправки письма необходимо сначала создать объект типа `ИнтернетПочтовоеСообщение`, заполнить все необходимые его свойства, после чего вызвать метод `Послать` объекта `Почта`, в параметре которого указать ссылку на созданный объект (листинг 8.17).

Листинг 8.17. Подготовка и отправка почтового сообщения

```
Почта = Новый ИнтернетПочта;
// Подключение к почтовому серверу
Попытка
   Почта.Подключиться(Профиль);
Исключение
   Сообщить(ОписаниеОшибки());
   Возврат;
КонецПопытки;

Сообщение = Новый ИнтернетПочтовоеСообщение;

// Добавить вложение
Сообщение.Вложения.Добавить(Новый ДвоичныеДанные("c:\Temp\MAV.txt"),
"MAV.txt");
// Имя отправителя письма
Сообщение.ИмяОтправителя = "Андрей Михайлов aka MAV";
// Отправитель сообщения
Сообщение.Отправитель = "mav@erpg.ru";
// Добавление почтового адреса получателя
ПочтовыеАдреса = Сообщение.Получатели;
ПочтовыйАдрес = ПочтовыеАдреса.Добавить();
ПочтовыйАдрес.Адрес = "alx@erpg.ru";
ПочтовыйАдрес.Пользователь = "ALX";

// Тема сообщения
Сообщение.Тема = "Пришли учредительные документы";
// Добавление текста сообщения
ИнтернетТекстПочтовогоСообщения = Сообщение.Тексты.Добавить();
ИнтернетТекстПочтовогоСообщения.Текст =
"До субботы жду от тебя учредительный договор и прочие документы";
ИнтернетТекстПочтовогоСообщения.ТипТекста =
ТипТекстаПочтовогоСообщения.ПростойТекст;

// Посылает сообщение типа ИнтернетПочтовоеСообщение
Попытка
   Почта.Послать(Сообщение);
Исключение
```

```
    Сообщить(ОписаниеОшибки());
    Возврат;
КонецПопытки;

// Отключение от почтового сервера
Почта.Отключиться();
```

Примечание

При вызове метода `Послать` объекта `Почта` используется конструкция `Попытка...Исключение...КонецПопытки`, которая необходима для подстраховки от неудачной отправки письма. Полный список свойств и методов каждого объекта приведен в документации.

Работа с протоколом FTP

Аббревиатура *FTP* расшифровывается как File Transfer Protocol (протокол передачи файлов). Протокол FTP оптимизирован для передачи файлов, это и послужило причиной того, что программы, использующие его, стали частью отдельного сервиса Интернета. Сервис, обеспечивающий доступ к файлам других компьютеров сети по протоколу FTP, носит одноименное название.

Чтобы воспользоваться протоколом FTP, понадобится специальная программа, так называемый FTP-клиент. FTP-клиентом может служить браузер (Microsoft Internet Explorer, Netscape Navigator), программа управления файлами FAR, специализированная программа CuteFTP и др.

Для обеспечения программного доступа к протоколу FTP можно использовать либо вызов программы ftp.exe, либо — объект `FTP`, являющийся частью объектной модели системы "1С:Предприятие" (версия 8.0).

Использование программы ftp.exe

Программа ftp позволяет передавать файлы на удаленный компьютер или получать их оттуда. Она работает с файлами и каталогами удаленного компьютера.

При работе с функциями ftp необходимо запустить на своем компьютере программу ftp.exe, которая связывается с серверной программой, работающей на удаленном компьютере. В дальнейшем, используя возможности ftp, можно задавать выполняемые серверной программой команды, которые транслируются в инструкции.

Для запуска программы ftp в контексте конфигурации системы "1С:Предприятие" (версия 7.7) достаточно вызвать команду `КомандаСистемы`,

в параметрах которой указать команду ftp с необходимыми параметрами. Эти параметры помещаются после названия — ftp. Каждый параметр состоит из дефиса (-) и одной буквы и имеет соответствующую команду того же имени, которая может быть использована внутри ftp.

Примечание

Следует различать использование опций и соответствующих ftp-команд.

Рассмотрим возможные параметры программы ftp.

- `-v` — заставляет ftp работать в многословном режиме. В этом режиме сообщения ftp, посланные удаленной машиной, появляются на экране дисплея-приемника. Кроме того, если этот режим используется после завершения передачи каждого файла, то появляется статистическое сообщение, уведомляющее об этом. Данный режим устанавливается по умолчанию, если ftp выполняется интерактивно. Если ftp выполняется в командном режиме, многословный режим выключен. Для того чтобы его включить, надо установить опцию `-v`. Включить этот режим внутри ftp можно с помощью команды `verbose`.
- `-d` — заставляет ftp работать в режиме отладки. В этом режиме сообщения ftp, посланные ftp в удаленную машину, отображаются на экране дисплея-передатчика. Если не используется эта опция — информация не отображается. Вызвать этот режим в ftp можно также используя команду `debug`.
- `-i` — данная опция означает не интерактивную работу.
- `-n` — опция позволяет предотвратить использование ftp-авторегистрации, во время связи с удаленной машиной. Когда установлен режим авторегистрации, ftp идентифицирует вас автоматически на удаленной машине и регистрирует вас на ней. Если вы используете `-n`-опцию для отключения автоматической регистрации, вы должны будете использовать команду `user`, чтобы вручную зарегистрироваться на удаленной машине.
- `-g` — является причиной лишения имен файлов, созданных в среде операционной системы UNIX, их расширений, например, таких как универсальное — *. Если не используется эта опция, ftp расширяет имена файлов с помощью универсального расширения в списке файлов. Вместо этой опции можно использовать команду `glob`.
- `-s:ИмяФайла` — определяет выполнение команды ftp в пакетном режиме. В файле, имя которого указывается в параметре, описывается последовательность команд. Данный режим является основным для автоматизации работы с протоколом FTP посредством системы "1С:Предприятие".

В листинге 8.18 приведен пример работы с FTP-сервером посредством пакетного файла.

Листинг 8.18. Подключение к FTP-серверу через пакетный файл

```
// Соединение с сервером
ТекстКоманды = "open ftp.server.ru
// Имя пользователя
|user
// Пароль пользователя
|password
// Выполняем команды программы ftp...
//...
// Закрыть связь с сервером
|quit
|";

// Имя временного файла
ИмяФайла = КаталогИБ() + "ftp\transfer.ftp";
// Создаем временный файл для пакетного выполнения команды ftp
Текст = СоздатьОбъект("Текст");
Текст.ДобавитьСтроку(ТекстКоманды);
Текст.Записать(ИмяФайла);
// Выполняем команду ftp в пакетном режиме
КомандаСистемы("ftp -s:" + ИмяФайла);
// Удаляем временный файл
ФС.УдалитьФайл(ИмяФайла);
```

Из приведенного примера видно, что текст пакетного файла обязательно начинается с команды `open`, после которой указывается имя сервера. Следующими двумя строками следуют имя пользователя и пароль. Заканчивается текст пакетного файла командой `quit`.

В табл. 8.1 приведен полный список команд программы ftp.

***Таблица 8.1.** Команды программы ftp*

Команда	Описание команды
`append`	Добавляет содержимое локального файла в конец удаленного файла. Например: `append <имя локального файла> <имя файла удаленной машины>`
`ascii`	Установка режима передачи файлов в формате ASCII (По умолчанию код всегда ASCII)

Таблица 8.1 (продолжение)

Команда	Описание команды
`bell`	Выдача звукового сигнала по завершении команд. Чтобы прекратить подачу сигнала, нужно снова вызвать эту команду
`binary`	Установка режима передачи файлов в двоичном формате
`bye` или `quit`	Закрывает все открытые связи
`cd`	Изменение рабочего каталога на удаленном компьютере. Например: `cd /usr/bin`
`close`	Завершение сеанса работы ftp и закрытие текущей связи
`debug`	Включение и выключение режима отладки
`delete`	Удаление файла на удаленном компьютере. Например: `delete <имя файла для удаления>`
`dir`	Вывод детального списка каталога удаленного компьютера. Например: `dir /usr/bin`. Если имя каталога не указывать, то будет распечатан текущий каталог удаленного компьютера. Для того чтобы занести результаты выполнения команды в файл, необходимо указать имя выходного файла. Например: `dir /usr/bin printfile`. Если необходимо напечатать текущий каталог в файл, необходимо вызвать команду: `dir . printfile` где "." означает текущий каталог
`get` или `recv`	Копирует файл из удаленного компьютера на локальный. Например: `get <имя файла удаленной машины> <имя файла вашей машины>`. Если просто указать имя файла удаленного компьютера, который нужно скопировать, то скопированный файл на вашей машине будет иметь то же самое имя
`glob`	Переключение метасимвольного расширения имен локальных файлов. Эта команда запрещает такое универсальное расширение имен файлов, созданных в среде операционной системы UNIX, как — '*'
`hash`	Переключает вывод символа "#" на экране пользователя после каждого блока данных, который посылается удаленному компьютеру. Размер блоков данных может быть различным в зависимости от версии программного обеспечения. Для того чтобы увидеть текущее состояние данного режима, с командой ftp должен использоваться режим отладки. Эта команда включает и выключает выдачу символа "#" на экран пользователя

Таблица 8.1 (продолжение)

Команда	Описание команды
lcd	Изменяет рабочий каталог, используемый ftp. Например: `lcd /usr/mav.` Если вы не задали имя каталога, то будет использоваться "стартовый" каталог

ls	Распечатывает аббревиатурный список содержания каталога удаленного компьютера, с которого установлена связь. Например:
	`ls /usr/bin.`
	Если имя каталога не определено, будет распечатан текущий. Можно также задать команду записи этих данных в файл. Это делается ftp, если указано имя файла локальной машины, куда следует поместить листинг каталога. Например:
	`ls /usr/bin printfile.`
	Имя каталога нужно задать до вывода файла (printfile). Например, если вам необходимо распечатать текущий каталог в файл с именем printfile: `ls . printfile`
	где символ "." ставится для указания, что каталог текущий

mdelete	Удаляет список файлов на удаленном компьютере. Например: `mdelete <имя 1-го файла> < имя 2-го файла >...`
mdir	Выводит листинг директории удаленного компьютера, и результат помещается в файл локальной машины. Можно задать список файлов удаленной машины и имя файла локальной машины, куда поместить результат при вызове команды. Например: `mdir <имя 1-го файла> <имя 2-го файла>... printfile`
mget	Копирование одного или более файлов с удаленного компьютера. Файлы после копирования будут иметь те же имена. Можно указать список файлов для копирования, например: `mget <имя 1-го файла> <имя 2-го файла>...`
mkdir	Создание каталога на удаленном компьютере. Например: `mkdir /u/mydir`
mls	Вывод содержимого нескольких каталогов удаленного компьютера. В команде можно задать список файлов удаленного компьютера и имя файла локального компьютера, куда поместить результат при выполнении команды, например: `mls <имя 1-го файла> ... printfile`
mput	Отправка нескольких файлов. Эта команда копирует один или более файлов с локального компьютера на удаленный. На удаленном компьютере файлы будут иметь те же имена. Например: `mput <1-й файл> <2-й файл> машины...`

Таблица 8.1 (окончание)

Команда	Описание команды
`open`	Подключение к протоколу FTP удаленного компьютера. Эта команда устанавливает связь с удаленным компьютером, в который предполагается передача файлов. Например: `open ftp.server.ru`
`prompt`	Переключение интерактивной подсказки для составных команд. Эта команда переключает запросы к ftp между одноместными (обрабатывают один файл) и многоместными командами (например, такими как mget). Эта команда подключается и отключается при повторном наборе
`put` или `send`	Отправка одного файла. Эта команда перемещает файл с локального компьютера на удаленный. Например: `put <имя локального файла> <имя удаленного файла>` или `put <имя локального файла>`
`pwd`	Вывод рабочего каталога удаленного компьютера
`quote`	Отправка произвольных команд ftp, которые поддерживает удаленный сервер. Список доступных команд может быть отображен на экране с помощью команды `remotehelp`
`remotehelp`	Получение справочных сведений с удаленного сервера. Эта команда запрашивает помощь ftp на удаленном компьютере, с которым установлена связь на данный момент времени. Запрашиваемая информация сообщает о том, какие команды поддерживает удаленный компьютер
`rename`	Переименование файла на удаленном компьютере. Например: `rename <имя 1-го файла> <имя 2-го файла>`
`mdir`	Вывод содержимого нескольких каталогов удаленного компьютера. Например: `rmdir /usr/mydir`
`status`	Выводит текущее состояние режимов `bell`, `form`, `hash`, `glob`, `port`, `type`
`type`	Установка типа передачи файлов. Допустимы двоичный и коды ASCII. Эта команда аналогична командам `ascii` и `binary`. Если не указан тип при вызове команды, то устанавливается ASCII
`trace`	Переключение трассировки пакетов. Эта команда включается и отключается при ее повторным наборе
`user`	Отправка сведений о новом пользователе. Команда позволяет пользователю идентифицировать самого себя на удаленном компьютере при установлении связи. Например: `user mike cat myaccount`
`verbose`	Переключение режима вывода сообщений. Данная команда включается и выключается при повторном наборе. В режиме отладки протокольные сообщения, посланные удаленной машиной, появляются на терминале локальной машины. Кроме того, в этом режиме отображается статистика процесса вывода после передачи каждого файла

В листинге 8.19 приведен пример работы пакетного режима команды ftp при работе с файлами.

Листинг 8.19. Пример работы пакетного режима команды ftp

```
// Соединение с сервером
ТекстКоманды = "open ftp.server.ru
// Имя пользователя
|user
// Пароль пользователя
|password
// Получить файл с сервера на локальную машину
|get file.rtf c:\file.rtf
|get file.mp3 c:\mp3\file.mp3
// Записать файл на сервер с локальной машины
|put c:\file.htm file.htm
|put c:\docs\file.txt file.txt
// Удалить файл с сервера
|delete file.rtf
// Создать новый каталог на FTP-сервере
|mkdir newdir
// Закрыть связь с сервером
|quit
|";

// Имя временного файла
ИмяФайла = КаталогИБ() + "ftp\transfer.ftp";
// Создаем временный файл для пакетного выполнения команды ftp
Текст = СоздатьОбъект("Текст");
Текст.ДобавитьСтроку(ТекстКоманды);
Текст.Записать(ИмяФайла);
// Выполняем команду ftp в пакетном режиме
КомандаСистемы("ftp -s:" + ИмяФайла);
// Удаляем временный файл
ФС.УдалитьФайл(ИмяФайла);
```

Использование объекта *FTPСоединение*

В системе "1С:Предприятие" (версия 8.0) для работы с удаленными компьютерами через протокол FTP предназначен объект `FTPСоединение`. С помощью данного объекта можно выполнять типовые действия с файлами, например, такие как: скачивать и закачивать файлы, переименовывать их, обращаться к их свойствам, создавать каталоги и т. д.

Для установления соединения с FTP-сервером проще всего воспользоваться конструктором объекта `FTPСоединение`, который имеет следующие параметры:

- `Сервер` — сервер, с которым осуществляется соединение;
- `Порт` — порт сервера, с которым осуществляется соединение (по умолчанию 21);
- `Имя пользователя` — имя пользователя на указанном сервере;
- `Пароль пользователя` — пароль пользователя на указанном сервере;
- `Прокси` — значение типа `ИнтернетПрокси` (прокси, используемый для соединения с сервером);
- `Пассивное соединение` — булево значение, определяющее тип ftp-соединения (по умолчанию — `Ложь`).

Поле установки соединения с помощью метода `УстановитьТекущийКаталог` можно указать текущий каталог сервера, с которым будет происходить работа в дальнейшем. Далее можно работать с файлами и каталогами удаленного компьютера. В листинге 8.20 приведен пример вывода содержимого каталога в табличное поле.

Листинг 8.20. Вывод содержимого каталога ftp

```
// Соединение с удаленным сервером
FTP = Новый FTPСоединение("ftp.server.ru",21,"user","password",,);
// Установка текущего каталога
FTP.УстановитьТекущийКаталог("httpd/www/");
// Получить массив файлов, расположенных в текущем каталоге
МассивФайлов = FTP.НайтиФайлы("",);
// Выбрать файлы их массива
Для каждого Файл Из МассивФайлов Цикл
  // Добавить информацию о файле в табличное поле
  СтрокаТП = ТабличноеПолеФайлы.Добавить();
  СтрокаТП.Имя = Файл.Имя;
  СтрокаТП.ПолноеИмя = Файл.ПолноеИмя;
  // Определяем, соответствует ли файловый объект файлу
```

```
  Если Файл.ЭтоФайл() Тогда
    СтрокаТП.ВремяИзменения = Файл.ПолучитьВремяИзменения();
    СтрокаТП.Размер = Файл.Размер();
  КонецЕсли;
КонецЦикла;
```

Для удаления файлов на сервере предназначен метод `Удалить` объекта `FTPСоединение`.

В программном коде, приведенном в листинге 8.21, показаны примеры удаления одного файла и группы файлов.

Листинг 8.21. Удаление файлов на удаленном компьютере

```
// Соединение с удаленным сервером
FTP = Новый FTPСоединение("ftp.server.ru",21,"user","password",,);
// Установка текущего каталога
FTP.УстановитьТекущийКаталог("usr/mav/");
Попытка
  // Удаляет файл на сервере
  FTP.Удалить("data.txt");
  // Удаляет группу файлов на сервере
  FTP.Удалить("tmp/", "*.txt");
Исключение
  Сообщить(ОписаниеОшибки());
КонецПопытки;
```

Во втором вызове метода `Удалить`, в коде, который представлен в листинге 8.21, используется маска, по которой определяются файлы, подлежащие удалению.

Для копирования файла с удаленного компьютера на локальный предназначена команда `Получить`. В параметрах этой команды указывается имя файла источника и имя файла локального компьютера (листинг 8.22).

Листинг 8.22. Копирование файла с удаленного компьютера на локальный

```
// Соединение с удаленным сервером
FTP = Новый FTPСоединение("ftp.server.ru",21,"user","password",,);
// Установка текущего каталога
FTP.УстановитьТекущийКаталог("usr/mav/");
// Выбор папки для сохранения файла
Диалог = Новый ДиалогВыбораФайла(РежимДиалогаВыбораФайла.ВыборКаталога);
```

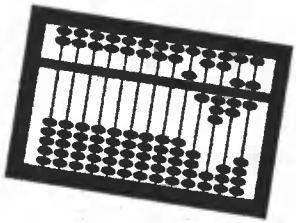
```
Диалог.Каталог = "c:\";
Диалог.Заголовок = "Выберите папку для сохранения...";
Диалог.МножественныйВыбор = Ложь;
Если Диалог.Выбрать() Тогда
  Попытка
    // Сохраняет файл из указанного адреса на локальный компьютер
    // Имя файла удаленного компьютера содержится в переменной <ИмяФайла>
    FTP.Получить(ИмяФайла, Диалог.Каталог + "\" + ИмяФайла);
  Исключение
    Сообщить(ОписаниеОшибки());
  КонецПопытки;
КонецЕсли;
```

В приведенном примере для определения локального каталога, в котором сохраняется файл, используется объект ДиалогВыбораФайла, описание методов и свойств которого можно найти в документации.

Для обратного действия, т. е. для копирования файла с локального компьютера на удаленный, используется метод Записать. Пример использования этого метода приведен в листинге 8.23.

Листинг 8.23. Копирование файла с локального компьютера на удаленный

```
// Соединение с удаленным сервером
FTP = Новый FTPСоединение("ftp.server.ru",21,"user","password",,);
// Установка текущего каталога
FTP.УстановитьТекущийКаталог("usr/mav/");
// Выбор файла для загрузки файла на ftp
Диалог = Новый ДиалогВыбораФайла(РежимДиалогаВыбораФайла.Открытие);
Диалог.Каталог = "c:\";
Диалог.Заголовок = "Выберите файл для загрузки на ftp...";
Диалог.МножественныйВыбор = Ложь;
Если Диалог.Выбрать() Тогда
  Попытка
    // Из полного имени выделяем только имя файла
    ИмяФайла = Сред(Диалог.ПолноеИмяФайла, СтрДлина(Диалог.Каталог)+1);
    // Записывает файл с локального компьютера на ftp
    FTP.Записать(Диалог.ПолноеИмяФайла, ИмяФайла);
  Исключение
    Сообщить(ОписаниеОшибки());
  КонецПопытки;
КонецЕсли;
```



Глава 9

Использование Active Directory Service Interfaces

Понятие ADSI

Технология *Active Directory Service Interfaces* (ADSI) предоставляет объектную модель, которая обеспечивает единообразный, не зависящий от конкретного сетевого протокола, доступ из программ или сценариев WSH к функциям различных каталогов (Active Directory для Windows 2000/Windows Server 2003, Windows Directory Service для Windows NT 4.0, NDS для Novell NetWare 4.*x*/5.*x*).

Объекты ADSI включены в операционные системы Windows XP/2000/2003, а также могут быть установлены в более ранних версиях, для чего их нужно загрузить с сервера компании Microsoft.

Для пользователей локальных или глобальных сетей важно уметь находить и использовать объекты разных типов (зарегистрированные пользователи, доступные сетевые принтеры и очереди печати и т. д.), которых в крупной сети может быть огромное количество. Администраторы сети должны поддерживать эти объекты в работоспособном состоянии. Под службой каталога (Directory Service) понимается та часть распределенной компьютерной системы (компьютерной сети), которая предоставляет средства для поиска и использования имеющихся сетевых ресурсов. Другими словами, служба каталога — это единое образование, объединяющее данные об объектах сети и совокупность служб, осуществляющих манипуляцию этими данными.

В гетерогенной (неоднородной) компьютерной сети могут одновременно функционировать несколько различных служб каталогов, например, NetWare Bindery для Novell Netware 3.*x*, NDS для Novell NetWare 4.*x*/5.*x*, Windows Directory Service для Windows NT 4.0 или Active Directory для Windows 2000. Естественно, для прямого доступа к разным службам каталогов приходится использовать различные инструментальные средства, что усложняет процесс администрирования сети в целом.

Для решения этой проблемы фирмой Microsoft была разработана технология Active Directory Service Interfaces (ADSI), которая предоставляет набор объектов ActiveX, обеспечивающих единообразный, не зависящий от конкретного сетевого протокола, доступ к функциям различных каталогов. Немаловажным фактом является то, что объекты ADSI включены в операционные системы Windows XP/2000. Также они могут быть установлены в более ранних версиях операционной системы, для чего их нужно скачать с сервера Microsoft.

Служба Active Directory состоит из четырех основных частей:

- *Служба DNS* (Domain Name Service);
- *WinNT Provider Service*;
- *NWCOMPAT Provider Service*;
- *Протокол LDAP* (Lightweight Directory Access Protocol).

Традиционная в сетях TCP/IP служба DNS используется, в частности, для поиска контроллеров домена, а благодаря протоколу LDAP клиенты могут по имени находить в каталоге Active Directory нужные объекты и получать доступ к их атрибутам. С помощью службы WinNT Provider производится доступ к доменам сети в операционных системах Windows NT, а посредством службы NWCOMPAT — в Novell NetWare 3.*x*.

В данной главе будет рассмотрена только служба WinNT Provider, как самая простая и наиболее используемая.

Применяя объекты ADSI и службы WinNT Provider можно, например:

- получить список имеющихся в локальной сети доменов;
- получить список имеющихся в домене рабочих станций;
- получить список всех групп, определенных в домене или на компьютере;
- добавить и удалить пользователя домена или компьютера;
- определить всех пользователей заданной группы или все группы, в которые входит определенный пользователь;
- просмотреть атрибуты пользователя и изменить его пароль.

Далее будут рассмотрены практические примеры работы с ADSI.

Формирование дерева объектов WinNT Provider

Все объекты службы WinNT Provider представляются в виде дерева, структура которого определена в табл. 9.1.

Таблица 9.1. *Иерархия классов WinNT Provider*

Имя класса	Описание
Namespace	Самый верхний уровень пространства имен. Пример: `Obj = ПолучитьCOMОбъект("WinNT://")`
Domain	Домен Windows NT. Пример: `Obj = ПолучитьCOMОбъект("WinNT://MyDomain")`
User	Учетная запись пользователя. Пример: `Obj=ПолучитьCOMОбъект("WinNT://MyDomain/User")`
Group	Группа учетных записей, управляющая правами доступа. Пример: `Obj=ПолучитьCOMОбъект("WinNT://MyDomain/Group")`
UserGroupCollection	Пользователи группы
GroupCollection	Прочие группы
Computer	Windows NT-сервер или рабочая станция. Пример: `Obj=ПолучитьCOMОбъект("WinNT://MyServer")`
PrintJob	Очередь задач принтера. Пример: `Obj=ПолучитьCOMОбъект("WinNT://MyServer/aPrinter")`
PrintJobsCollection	Коллекция задач принтера
PrintQueue	Очередь задач программы-планировщика принтера
Service	Запущенные сервисы
FileService	Сервис доступа к файлам
FileShare	Общедоступные файлы
Resource	Ресурсы сервиса
Session	Активные файловые подключения
User	Локальные пользователи. Пример: `Obj = Получить COMОбъект("WinNT://MyServer/User")`
Group	Локальные группы. Пример: `Obj = ПолучитьCOMОбъект("WinNT://MyServer/Group")`

Таблица 9.1 (окончание)

Имя класса	Описание
UserCollection	Коллекция локальных пользователей группы
GroupCollection	Коллекция локальных групп
Schema	Схема WinNT-контейнера
Class	Схема определения классов
Property	Схема определения свойств
Syntax	Синтаксис свойств

В качестве имени домена необходимо указывать имя NETBIOS, т. к. WinNT Provider не поддерживает работу с DNS-именами.

Для создания определенного объекта необходимо подсоединиться к целевому контейнеру используя функцию ПолучитьCOMОбъект. Для точной идентификации класса создаваемого объекта можно указать его имя через запятую (листинг 9.1).

Листинг 9.1. Создание объектов WinNT Provider

```
// Создание объекта очереди задач принтера на компьютере "Comp"
Queue = ПолучитьCOMОбъект("WinNT:// Domain/Comp/aPrinter,printQueue");
// Создание объекта пользователя "MAV"
User = ПолучитьCOMОбъект("WinNT:// Domain/MAV,user");
// Создание объекта компьютера "Comp"
Computer = ПолучитьCOMОбъект("WinNT://Comp");
```

Для формирования дерева объектов WinNT Provider используется рекурсивная процедура ВывестиУровеньПространства, показанная в листинге 9.2.

Листинг 9.2. Формирование дерева объектов WinNT Provider

```
// Вызов процедуры и инициализация рекурсии
ВывестиУровеньПространства("WinNT:", ТабличноеПолеПространствоИмен.Строки);

// Рекурсивная процедура вывода очередного уровня иерархии
Процедура ВывестиУровеньПространства(Имя, ТП)
  // Связываемся с корневым объектом Namespace
  NameSpace = ПолучитьCOMОбъект(Имя);
```

```
    // Перебираем элементы коллекции
    Попытка
      Для каждого Domain Из NameSpace Цикл
        СтрокаТП = ТП.Добавить();
        СтрокаТП.КолонкаИмя = Domain.Name;
        // Ограничиваем иерархию 4-ю уровнями
        Если СтрокаТП.Уровень() < 4 Тогда
          ВывестиУровеньПространства(Имя + ?(Имя="WinNT:","//","/") +
          Domain.Name, СтрокаТП.Строки);
        КонецЕсли;
      КонецЦикла;
    Исключение
    КонецПопытки;
КонецПроцедуры // ВывестиУровеньПространства()
```

В данном примере для простоты вся выборка объектов определенного уровня взята в программные скобки `Попытка...Исключение...КонецПопытки`. Это сделано для того, чтобы процедура не отработала с критической ошибкой, при отсутствии объектов в коллекции.

Вся иерархия объектов выводится в дерево значений, которое расположено на форме обработки с именем `ТабличноеПолеПространствоИмен`.

Приведенный в листинге 9.2 пример покажет дерево объектов локальной сети в соответствии с иерархией, приведенной в табл. 9.1.

Вывод списка всех доменов локальной сети

Для вывода всех доменов локальной сети необходимо выполнить следующую простую последовательность действий.

1. Используя функцию `ПолучитьCOMОбъект`, подсоединимся к целевому контейнеру `"WinNT:"`, который будет содержать коллекцию доменов.
2. С помощью конструкции `Для каждого...Из...Цикл...КонецЦикла` перебираем полученную коллекцию.

Пример вывода списка доменов показан в листинге 9.3.

Листинг 9.3. Вывод списка всех доменов локальной сети

```
// Вывод списка всех доменов локальной сети
Домены = ПолучитьCOMОбъект("WinNT:");
```

```
Для каждого Домен Из Домены Цикл
  Сообщить(Домен.Name);
КонецЦикла;
```

Вывод списка пользователей группы

Для вывода списка пользователей группы необходимо выполнить следующую простую последовательность действий.

1. Используя функцию `ПолучитьCOMОбъект`, подсоединиться к целевому контейнеру `"WinNT://<ИмяКомпьютера>/<ИмяГруппы>,Group"`, который будет содержать коллекцию пользователей. С помощью дополнения `Group` указывается, что создается элемент группы.
2. С помощью конструкции `Для каждого...Из...Цикл...КонецЦикла` перебираем полученную коллекцию.

Пример вывода списка пользователей группы `"Пользователи"` приведен в листинге 9.4.

Листинг 9.4. Вывод списка пользователей группы

```
// Связываемся с компьютером
Obj = ПолучитьCOMОбъект("WinNT://" + ИмяКомпьютера +
"/Пользователи,group");
Сообщить("Список пользователей компьютера " + ИмяКомпьютера);
// Выборка всех пользователей из коллекции
Для каждого Пользователь Из Obj.Members() Цикл
  Сообщить(Пользователь.Name);
КонецЦикла;
```

Создание и удаление пользователей

Для создания нового пользователя необходимо выполнить следующую последовательность действий.

1. Используя функцию `ПолучитьCOMОбъект`, подсоединиться к целевому контейнеру `"WinNT://<ИмяКомпьютера> "`.
2. Используя метод `Create` из ADSI, создадим новый объект `User` в локальном кэше свойств. Метод `Create` содержит два параметра: имя класса создаваемого объекта и имя объекта.
3. Используя свойство `Description`, можно задать текстовое описание пользователя.

4. С помощью метода `SetPassword` можно задать пароль пользователя.
5. С помощью метода `SetInfo` из ADSI новый объект записывается в каталог.

Пример создания нового пользователя приведен в листинге 9.5.

Листинг 9.5. Создание нового пользователя

```
// Связываемся с компьютером
Computer = ПолучитьCOMОбъект("WinNT://" + ИмяКомпьютера);
// Создаем объект класса User
User = Computer.Create("user", ИмяПользователя);
// Добавляем описание созданного пользователя
User.Description = Описание;
// Задаем пароль пользователя
User.SetPassword(ПарольПользователя);
// Сохраняем информацию на компьютере
User.SetInfo();
```

Для удаления локального пользователя достаточно соединиться с компьютером и вызвать метод `Delete`, указав имя класса удаляемого объекта и имя самого объекта (листинг 9.6).

Листинг 9.6. Удаление пользователя

```
// Связываемся с компьютером
Computer = ПолучитьCOMОбъект("WinNT://" + ИмяКомпьютера);
// Удаление пользователя
Tmp = Computer.Delete("user", ИмяПользователя);
```

Создание и удаление групп пользователей

Создание и удаление групп пользователей происходит аналогичным образом, что и создание и удаление пользователей. Исключением является то, что в первом параметре метода `Create` указывается имя класса `"Group"` (листинг 9.7).

Листинг 9.7. Создание новой группы пользователей

```
// Связываемся с компьютером
Computer = ПолучитьCOMОбъект("WinNT://" + ИмяКомпьютера);
// Создаем объект класса Group
```

```
User = Computer.Create("group", ИмяГруппы);
// Добавляем описание созданной группы
User.Description = Описание;
// Сохраняем информацию на компьютере
User.SetInfo();
```

Удаление группы происходит аналогично удалению пользователя — при помощи метода `Delete`.

Манипулирование с учетными записями

С помощью службы WinNT Provider можно управлять учетными записями: создавать, изменять пароль, отключать учетные записи. Примеры создания и удаления были показаны в программных кодах, приведенных в листингах 9.5 и 9.6.

В листинге 9.8 приведен пример инициализации выдачи запроса на смену пароля, который должен появиться при следующем подключении пользователя.

Листинг 9.8. Смена пароля пользователя при следующем подключении

```
// Получаем объект пользователя
User = ПолучитьCOMОбъект("WinNT://" + ИмяКомпьютера + "/" +
ИмяПользователя + ",user");
// Указываем, что пользователь должен сменить пароль
// при следующем подключении
User.Put("PasswordExpired", 1);
// Сохраняем информацию на компьютере
User.SetInfo();
Предупреждение("При следующем подключении пользователя " +
ИмяПользователя + " к компьютеру " + ИмяКомпьютера +
" будет запрошен ввод нового пароля.");
```

В данном примере в свойство пользователя `PasswordExpired` записывается значение `1`, указывающее на то, что при подключении пользователя к компьютеру появится запрос на смену пароля. Метод `SetInfo` производит сохранение информации на компьютере.

С помощью свойства пользователя `AccountDisabled` можно отключить или включить учетную запись, записав в него соответственно значение `Истина` или `Ложь`.

В листинге 9.9 приведен пример отключения учетной записи.

Листинг 9.9. Отключение учетной записи

```
// Получаем объект пользователя
User = ПолучитьCOMОбъект("WinNT://" + ИмяКомпьютера + "/" +
ИмяПользователя);
// Отключить учетную запись
User.AccountDisabled = Истина;
// Сохраняем информацию на компьютере
User.SetInfo();
// Проверка отключения учетной записи
Сообщить("Проверка отключения учетной записи " + ИмяПользователя + ": " +
User.AccountDisabled);
```

С помощью свойства `AccountExpirationDate` можно ограничить срок действия учетной записи. В листинге 9.10 приведен пример, в котором учетная запись действительна до 5 июня 2007 года.

Листинг 9.10. Установить дату истечения действия учетной записи

```
// Получаем объект пользователя
User = ПолучитьCOMОбъект("WinNT://" + ИмяКомпьютера + "/" +
ИмяПользователя);
// Установить дату истечения действия учетной записи
User.AccountExpirationDate = "05/06/2007";
// Сохраняем информацию на компьютере
User.SetInfo();
```

Примечание

При разборе данного примера обратите внимание на формат даты, передаваемой в свойство `AccountExpirationDate`.

Вывод информации о компьютере и пользователе

Как видно из табл. 9.1, служба WinNT Provider позволяет работать с множеством объектов, определенных в сети. Каждый из объектов имеет свой уникальный набор свойств, которые приведены в табл. 9.2.

Примечание

Назначение данных свойств расписываться не будет.

Таблица 9.2. Свойства классов WinNT Provider

Имя класса	Поддерживаемые свойства
Computer	Division Owner OperatingSystem OperatingSystemVersion Processor ProcessorCount
Domain	MinPasswordLength MinPasswordAge MaxPasswordAge MaxBadPasswordsAllowed PasswordHistoryLength AutoUnlockInterval LockoutObservationInterval
FileService	HostComputer DisplayName Version ServiceType StartType Path ErrorControl LoadOrderGroup Description MaxUserCount ServiceAccountName Dependencies
FileShare	CurrentUserCount Description HostComputer Path MaxUserCount
FPNWFileService	HostComputer DisplayName Version ServiceType StartType Path ErrorControl LoadOrderGroup

Таблица 9.2 (продолжение)

Имя класса	Поддерживаемые свойства
FPNWFileService	ServiceAccountName Dependencies Description MaxUserCount
FPNWFileShare	CurrentUserCount HostComputer Path MaxUserCount
FPNWResource	User Path LockCount
FPNWSession	User Computer ConnectTime
Group	Description objectSid
Namespace	Все перечисленные в таблице свойства
PrintJob	HostPrintQueue User TimeSubmitted TotalPages Size Description Priority StartTime UntilTime Notify TimeElapsed PagesPrinted Position Action ObjectGUID
PrintQueue	PrinterPath PrinterName Model Datatype PrintProcessor PrintDevices

Таблица 9.2 (продолжение)

Имя класса	Поддерживаемые свойства
PrintQueue	Description HostComputer Location StartTime UntilTime DefaultJobPriority JobCount Priority Attributes BannerPage ObjectGuid Action
Resource	User Path LockCount
Service	HostComputer LoadOrderGroup ServiceAccountName Dependencies StartType ServiceType DisplayName Path ErrorControl
Session	Computer ConnectTime IdleTime User
User	AccountExpirationDate AutoUnlockInterval BadPasswordAttempts Description FullName HomeDirDrive HomeDirectory UserFlags LockoutObservationInterval LoginHours

Таблица 9.2 (окончание)

Имя класса	Поддерживаемые свойства
User	LastLogin
	LastLogoff
	LoginScript
	LoginWorkstations
	MinPasswordAge
	MinPasswordLength
	MaxBadPasswordsAllowed
	MaxLogins
	MaxPasswordAge
	MaxStorage
	ObjectSid
	Parameters
	PasswordAge
	PasswordExpirationDate
	PasswordExpired
	PasswordHistoryLength
	PrimaryGroupID
	Profile

Важно отметить, что перечисленные в табл. 9.2 свойства не всегда будут доступны. Наличие тех или иных свойств зависит от версии операционной системы и версии установленной компоненты ADSI.

В листинге 9.11 приведен пример вывода свойств компьютера (установленная операционная система, имя процессора и т. п.).

Листинг 9.11. Вывод информации о компьютере

```
// Получаем объект компьютера
Computer = ПолучитьCOMОбъект("WinNT://" + ИмяКомпьютера);
// Выводим свойства компьютера
Сообщить("Division = " + Computer.Division);
Сообщить("Owner = " + Computer.Owner);
Сообщить("OperatingSystem = " + Computer.OperatingSystem);
Сообщить("OperatingSystemVersion = " + Computer.OperatingSystemVersion);
Сообщить("Processor = " + Computer.Processor);
Сообщить("ProcessorCount = " + Computer.ProcessorCount);
```

Создав объект User, можно выводить свойства пользователя (имя пользователя, описание, время подключения, имя рабочей папки и т. п.). Пример вывода свойств пользователя приведен в листинге 9.12.

Листинг 9.12. Вывод информации о пользователе

```
// Получаем объект пользователя
User = ПолучитьCOMОбъект("WinNT://" + ИмяКомпьютера + "/" +
ИмяПользователя);
// Выводим свойства пользователя
Сообщить("FullName = " + User.FullName);
Сообщить("AccountExpirationDate = " + User.AccountExpirationDate);
Сообщить("AutoUnlockInterval = " + User.AutoUnlockInterval);
Сообщить("BadPasswordAttempts = " + User.BadPasswordAttempts);
Сообщить("Description = " + User.Description);
Сообщить("HomeDirDrive = " + User.HomeDirDrive);
Сообщить("HomeDirectory = " + User.HomeDirectory);
Сообщить("UserFlags = " + User.UserFlags);
Сообщить("LockoutObservationInterval = " +
User.LockoutObservationInterval);
Сообщить("LoginHours = " + User.LoginHours);
Сообщить("LastLogin = " + User.LastLogin);
Сообщить("LoginScript = " + User.LoginScript);
Сообщить("MinPasswordAge = " + User.MinPasswordAge);
Сообщить("MinPasswordLength = " + User.MinPasswordLength);
Сообщить("MaxBadPasswordsAllowed = " + User.MaxBadPasswordsAllowed);
Сообщить("MaxPasswordAge = " + User.MaxPasswordAge);
Сообщить("MaxStorage = " + User.MaxStorage);
Сообщить("ObjectSid = " + User.ObjectSid);
Сообщить("Parameters = " + User.Parameters);
Сообщить("PasswordAge = " + User.PasswordAge);
Сообщить("PasswordExpirationDate = " + User.PasswordExpirationDate);
Сообщить("PasswordExpired = " + User.PasswordExpired);
Сообщить("PasswordHistoryLength = " + User.PasswordHistoryLength);
Сообщить("PrimaryGroupID = " + User.PrimaryGroupID);
Сообщить("Profile = " + User.Profile);
```

Аналогичным образом можно выводить свойства и других объектов сети, создавая экземпляры этих объектов так, как это было показано в листинге 9.1.

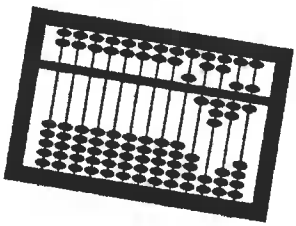
Помимо перечисленных свойств каждый пользователь имеет уникальный идентификатор SID (user security identifier), который можно получить с помощью метода `Get`, в параметрах которого передать название свойства — `objectSID`. Данный метод возвращает значение типа `COMSafeArray`, которое посредством метода `Выгрузить` можно конвертировать в обычный массив. Далее, с помощью конструкции `Для каждого...из...Цикл...КонецЦикла` можно обойти все элементы массива и собрать полный идентификатор.

Пример получения идентификатора пользователя приведен в листинге 9.13.

Листинг 9.13. Вывод SID пользователя (user security identifier)

```
// Получаем объект пользователя
User = ПолучитьCOMОбъект("WinNT://" + ИмяКомпьютера + "/" +
ИмяПользователя);
// Получаем SID (user security identifier)
Sid = User.Get("objectSID"); // получили тип COMSafeArray
Масс = Новый Массив();
// Выгружаем COMSafeArray в массив
Масс = Sid.Выгрузить();
SidСтрока = "";
// Складываем все строки массива
Для каждого Значение из Масс Цикл
  SidСтрока = SidСтрока + Значение + " ";
КонецЦикла;
Сообщить("SID пользователя " + ИмяПользователя + ":" + SidСтрока);
```

Приложение



Описание прилагаемого компакт-диска

В прилагаемом к книге CD-ROM содержатся исходные тексты программ всех примеров, которые в ней рассмотрены. Помимо примеров на диске содержится Offline-версия интернет-проекта ERP Group (**www.erpg.ru**), основателем которого является автор книги.

Структура компакт-диска представлена в табл. П.1.

Таблица П.1. *Содержимое компакт-диска*

Каталог\Файл	Описание
Chapter 01\	Примеры, рассмотренные в *главе 1.*
ActiveBarCode.epf	Формирование штрихкода средствами `ActiveBarcode` для "1С:Предприятие 8.0"
ActiveBarCode.ert	Формирование штрихкода средствами `ActiveBarcode` для "1С:Предприятие 7.7"
HTML Editor.epf	Пример создания визуального редактора HTML-страниц для "1С:Предприятие 8.0"
InternetExplorer.epf	Примеры работы с объектом `InternetExplorer` для "1С:Предприятие 8.0"
InternetExplorer.ert	Примеры работы с объектом `InternetExplorer` для "1С:Предприятие 7.7"
IP.epf	Определение IP-адреса локального и удаленного компьютера для "1С:Предприятие 8.0"
MSAgent.epf	Примеры использования помощников MS Agent для "1С:Предприятие 8.0
MSAgent.ert	Примеры использования помощников MS Agent для "1С:Предприятие 7.7"
MSAgentBalloonDialog.epf	Примеры интерактивного взаимодействия с MS Agent для "1С:Предприятие 8.0"

Таблица П.1 (продолжение)

Каталог\Файл	Описание
Chapter 01\	Примеры, рассмотренные в *главе 1*.
MSAgentBalloonDialog.ert	Примеры интерактивного взаимодействия с MS Agent для "1С:Предприятие 7.7"
RegExp.epf	Примеры работы с регулярными выражениями для "1С:Предприятие 8.0"
WindowsMediaPlayer.epf	Пример использования проигрывателя Windows Media Player для "1С:Предприятие 8.0"
Chapter 02\	Примеры, рассмотренные в *главе 2*.
ComputerSystemInformation.epf	Вывод подробной информации об аппаратном и программном составе локального компьютера для "1С:Предприятие 8.0"
WMI.epf	Примеры использования объектов WMI для "1С:Предприятие 8.0"
WMI.ert	Примеры использования объектов WMI для "1С:Предприятие 7.7"
WMIClassViewer.epf	Инструмент просмотра всех классов WMI, их свойств и значений для "1С:Предприятие 8.0"
WmiServices.classes	Файл со списком WMI-классов, который используется для загрузки классов в обработке WmiServices.ert
WmiServices.ert	Инструмент просмотра всех классов WMI, их свойств и значений для "1С:Предприятие 7.7"
Chapter 03\	Примеры, рассмотренные в *главе 3*.
FileSystemObject.epf	Пример работы с файловой системой для "1С:Предприятие 8.0"
MSScriptControl.epf	Пример запуска произвольного VB-скрипта (VBS) для "1С:Предприятие 8.0"
ReadMP3Tags.epf	Пример чтения тегов (`IDv1`) MP3-файлов для "1С:Предприятие 8.0
WSH.epf	Примеры использования объектов `WSH` для "1С:Предприятие 8.0"
Chapter 04\	Примеры, рассмотренные в *главе 4*.
Photoshop.epf	Примеры использования Adobe Photoshop в качестве OLE-сервера для версии "1С:Предприятие 8.0"
V8Application.vbs	Пример использования OLE-соединения к БД "1С:Предприятие 8.0" из VBS

Таблица П.1 (продолжение)

Каталог\Файл	Описание
Chapter 04\	Примеры, рассмотренные в *главе 4*.
V8ApplicationCOMConnector.epf	Пример использования COM-соединения для "1С:Предприятие 8.0"
V8COMConnector.vbs	Пример использования COM-соединения к БД "1С:Предприятие 8.0" из VBS
WordCheckSpelling.epf	Пример реализации проверки орфографии текстов с помощью MS Word для "1С:Предприятие 8.0"
WordExcel.epf	Примеры использования MS Word и Excel в качестве OLE-сервера для "1С:Предприятие 8.0"
WordMacros.epf	Динамическое создание и выполнение макросов MS Word для "1С:Предприятие 8.0"
Chapter 05\	Примеры, рассмотренные в *главе 5*.
ADO Demo.epf	Пример подключения к произвольной БД средствами ADO для "1С:Предприятие 8.0"
ADO Demo.ert	Пример подключения к произвольной БД средствами ADO для "1С:Предприятие 7.7"
ExcelADO.epf	Пример подключения к книге MS Excel средствами ADO для "1С:Предприятие 8.0"
HexDumper.epf	Пример реализации шестнадцатеричного редактора средствами ADO для "1С:Предприятие 8.0"
ProjectADO.epf	Пример подключения к проекту MS Project средствами ADO для "1С:Предприятие 8.0"
example.mpp	Пример файла проекта, используемого в обработке ProjectADO.epf
Chapter 06\	Примеры обработок, рассмотренные в *главе 6*.
SQL-DMO Demo.ert	Пример подключения MS SQL Server средствами SQL-DMO для "1С:Предприятие 7.7"
Chapter 07\	Примеры обработок, рассмотренные в *главе 7*.
ControlPanel.epf	Примеры запуска команд управления Windows, использующие rundll32 для "1С:Предприятие 8.0"
Chapter 08\	Примеры обработок, рассмотренные в *главе 8*.
FTP.epf	Пример работы с FTP с помощью объекта `FTPСоединение` для "1С:Предприятие 8.0"
FTP.ert	Пример работы с FTP с помощью команды ftp для "1С:Предприятие 7.7"

Таблица П.1 (окончание)

Каталог\Файл	Описание
Chapter 08\	Примеры обработок, рассмотренные в *главе 8*.
Mail.epf	Примеры работы с электронной почтой для "1С:Предприятие 8.0"
Mail.ert	Примеры работы с электронной почтой для "1С:Предприятие 7.7"
RomPop3.ert	Пример работы с протоколом POP3 для "1С:Предприятие 7.7"
RomSmtp.ert	Пример работы с протоколом SMTP для "1С:Предприятие 7.7"
Chapter 09\	Примеры обработок, рассмотренные в *главе 9*.
ADSI.epf	Примеры использования Active Directory Service Interfaces "1С:Предприятие 8.0"
AddOn\	Дополнительные файлы и COM-объекты, необходимые для запуска некоторых примеров.
ActiveBarcode	Установочный комплект ActiveBarcode 3.50
ADSI	Microsoft Active Directory Service Interfaces version 2.5 (ADSI)
COMServices	COM-объект (с исходными кодами на VB), предназначенный для работы с COM-объектами в системе "1С:Предприятие 7.7"
COMViewer	OLE/COM Object Viewer
IE60	Internet Explorer 6.0 — русская и английская версии
MDAC	Microsoft Data Access Components 2.8
MSAgent	Установочный комплект Microsoft Agent 2.0, включая персонажи и SDK
ROM-Mail	Внешняя компонента ROM-Mail
Wmi	Windows Management Instrumentation (WMI) Core 1.5 (Windows 95/98) Distributed Component Object Model (DCOM) for Windows 95/98 1.3
Wmp	Установочный комплект Windows Media Player 9.0
Wsh	Windows Script 5.6 включая SDK
XMLCore	MS XML Core 4.0
ERPG.ru\	Offline-версия Web-проекта "ERP Group" Андрея Михайлова (**www.erpg.ru**)

Предметный указатель

К

М

О

П

Р

С

(окончание рубрики см. на стр. 336)

Т

У

Ф

Ч

1С: ПРЕДПРИЯТИЕ 7.7/8.0

СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ

ИСПОЛЬЗУЙ ВОЗМОЖНОСТИ СИСТЕМЫ НА ВСЕ 100

МИХАЙЛОВ АНДРЕЙ ВИТАЛЬЕВИЧ

Автор имеет большой опыт коммерческой разработки информационных систем на базе платформы "1С:Предприятие" версий 7.7 и 8.0, в том числе спроектированных и успешно внедренных "с нуля". Несколько его разработок получили сертификат "1С:Совместимо! Система программ 1С:Предприятие". Принимал участие в разработке конфигураций "1С:Управление производственным предприятием" на платформе "1С:Предприятие 8.0", "1С-Рарус:CRM Управление продажами", "Астрософт. Деловое досье: Учет оборудования" и многих других. Имеет ряд сертификатов фирмы "1С", в том числе самые разнообразные сертификаты по версии 8.0.

В настоящее время – вице-президент компании "Технологии успеха", которая занимается разработкой и внедрением информационных систем на базе платформы "1С:Предприятие 8.0".

Книга не является традиционным учебником по программированию в системе "1С:Предприятие". Основное внимание в ней уделено практическому использованию неявных, скрытых, но не менее важных, возможностей этой платформы, а также операционной системы и разнообразных COM-объектов.

Книга представляет собой практическое руководство, предназначенное для программистов, разработчиков и администраторов системы "1С:Предприятие" при работе с технологиями ActiveX, COM, WMI и WSH, при интеграции с базами данных через интерфейсы ADO и SQL-DMO, при работе с Интернетом, электронной почтой и FTP, взаимосвязи с внешними приложениями и многим другим.

Главная цель книги – показать читателю, что возможности системы "1С:Предприятие" не ограничиваются только решениями задач для автоматизации документооборота. В системе можно применять различные технологии программирования, которые активно используются при разработке приложений на "стандартных" языках и в других средах разработки.

Материал сопровождается большим количеством наглядных примеров. Книга ориентирована на программистов, разработчиков и администраторов различного уровня.

КАТЕГОРИЯ:
ПРОГРАММИРОВАНИЕ/1С:ПРЕДПРИЯТИЕ

+CD Компакт-диск содержит все исходные тексты примеров, приведенных в книге, а также COM-объекты.

ISBN 5-94157-536-X

9 785941 575367

БХВ-ПЕТЕРБУРГ 190005, Измайловский пр., 29
E-mail: mail@bhv.ru Internet: www.bhv.ru
Тел.:(812) 251-4244 Факс:(812) 251-1295