МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение   
высшего профессионального образования

«Ижевский государственный технический университет имени М.Т. Калашникова»

(ФГБОУ ВПО «ИжГТУ имени М.Т. Калашникова»)

Факультет «Информатика и вычислительная техника»

Кафедра «Защита информации в компьютеризированных системах»

Стукалина Е.Ф.

Методические указания для выполнения лабораторных работы по дисциплине «Автоматизация процессов на основе информационных технологий»

**Работа с объектными моделями в операционной системе Windows**



Ижевск – 2017

**Рег. номер \_\_\_/Ф\_\_\_/ГОС2**

Рецензент:

Составитель: к.т.н., доцент каф. ВТ Стукалина Е.Ф.

Методические указания предназначены для изучения теоретического материала и выполнения лабораторных работ в области современных объектных моделей и форматов обмена данными между приложениями по дисциплине «Информационно – аналитическое обеспечение комплексных систем информационной безопасности» для студентов специальности 090105 «Комплексное обеспечение информационной безопасности автоматизированных систем».

Рекомендовано Советом факультета «Информатики и вычислительной техники» к использованию в качестве учебно-методических материалов для использования в учебном процессе для специальности 090105.65 «Комплексное обеспечение информационной безопасности автоматизированных систем»

(протокол № \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ от \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_г)

Оглавление

[Введение 4](#_Toc372834069)

[Объектные модели ОС Windows 4](#_Toc372834070)

[Лабораторная работа. Применение объектных моделей Windows для анализа и поиска информации 6](#_Toc372834071)

[Работа с базами данных и применение объектной модели ADO 7](#_Toc372834072)

[Лабораторная работа. Применение объектной модели ADO для получения данных из СУБД 12](#_Toc372834073)

[Интеграция офисных приложений. Приложения Outlook и PowerPoint 14](#_Toc372834074)

[Лабораторная работа. Механизмы интеграции офисных приложений 24](#_Toc372834075)

[Основы разметки XML 25](#_Toc372834076)

[Лабораторная работа. Создание и обработка XML-документов 33](#_Toc372834077)

[Контрольные вопросы 34](#_Toc372834078)

# 

# Введение

В операционную систему Windows встроено множество объектных моделей, применение которых может очень сильно расширить возможности пользовательских приложений. Кроме того, при администрировании систем и пользователей объектные модели могут быть очень полезны. Далее в лабораторных работах изучаются возможности объектных моделей.

# Объектные модели ОС Windows

Чтобы использовать возможности объектных моделей в своей программе, необходимо добавить ссылку на них в ваш проект. Делается это очень просто: в окне редактора **Visual Basic выберите Tools / References** и в списке установите флажок около нужной библиотеки.

Вот перечень наиболее интересных с точки зрения применения в своих приложениях объектных моделей:

**Windows Script Host Object Model (wshom.exe)** — эта библиотека предназначена для автоматизации работы администраторов. Она предоставляет возможность программно работать с сетью, принтерами, реестром, ярлыками, журналом событий, позволяет запускать внешние приложения и передавать в них нажатия клавиш и консольные строки и т. п. Есть на всех компьютерах с Windows 2000, ХР, 2003 (в качестве необязательного компонента имеется и в Windows 98 Second Edition и Windows NT 4.0);

**Microsoft Scripting Runtime (scrrun.dll)** — еще одна библиотека для администраторов. Главное ее богатство — очень удобный (и при этом простой) набор классов для работы с файловой системой — дисками, каталогами, файлами, содержимым текстовых файлов и т. п. Поставляется в одном наборе с Windows Script Host Object Model;

**Microsoft ADO** (набор файлов, начинающихся с "msado")— классы для работы с базами данных.

**Microsoft SQLDMO Object Library (файл sqldmo.dll)** — набор классов для получения полного контроля над Microsoft SQL Server (возможность производить любые административные операции, выполнять запросы и т. п.), Имеется только на тех компьютерах, на которых установлен SQL Server версий 7.0, 2000 или 2005. В SQL Server 2005 она разбита на несколько частей — SMO (SQL Server Management Objects), RMO (Replication Management Objects) и AMO (Analysis Management Objects);

**Microsoft CDO (версия 1.21, for NTS версия 1.2, for Windows 2000 версия 1.0; файлы olemsg.dll, cdonts.dll, cdosys.dll)**— наборы классов для работы с электронной почтой. Можно использовать для создания и отправки своих почтовых сообщений, просмотра новых сообщений в почтовом ящике и т. п. Есть на всех компьютерах под управлением Windows 2000, ХР и Windows 2003 Server. На компьютерах с более старыми операционными системами обычно также имеется, поскольку эта библиотека устанавливается вместе с Microsoft Office;

**Microsoft WMI Scripting V 1.1 (wbemdisp.tlb)—** расширение возможностей программ через программный интерфейс WM1 (Windows Management Instrumentarium). Возможности совершенно невероятные: от управления скоростью вращения вентилятора (и вообще работы с любыми устройствами, про которые знает операционная система) до установки программного обеспечения, запуска процессов на удаленном компьютере, управления службами и т. п. В этой модели реализованы очень мощные возможности работы с событиями. Например, можно выполнять какой-либо код ответа на запуск или завершение работы программы с определенным именем на удаленном компьютере, в ответ на создание или удаление файла в каталоге, появления записи в журнале событий и т. п. Эта объектная модель (вместе со службой WMI) встроена во все компьютеры под управлением Windows 2000, ХР и Windows 2003 Server.

**Active Directory Scripting Interface (adsldp.dll, wldap32.dll, adsnt.dll, adsnds.dll, adsnw.dll)** — взаимодействие с объектами в каталогах Active Derectory, NT, NetWare, т. e. работа с учетными записями пользователей группами, объектами компьютеров, принтеров и т. п. Также встроена в операционные системы Windows 2000, ХР и Windows 2003 Server (как в серверные, так и в пользовательские версии);

Основной инструмент для поиска методов, свойств, событий — **Object Browser**, утилита, которая интегрирована в редактор кода VBA. Чтобы ей воспользоваться, необходимо вокне редактора кода нажать клавишу <F2> и выбрать нужную библиотеку классов. Классы показываются в левом списке как прямоугольники с разно-цветными "кирпичиками", методы — как летящий зеленый предмет, свойства — как надпись, на которую указывает рука, события— значок молнии.

Свои значки предусмотрены для модулей и перечислений. Если нужно просмотреть библиотеку типов, которой еще нет в списке **Object Browser**, необходимо добавить ссылку на нее через меню **Tools|References** или пункт References в контекстном меню самого Object Browser.

# Лабораторная работа №1. Применение объектных моделей Windows для анализа и поиска информации

**Цель лабораторной работы:** Изучить возможности использования внешних библиотек для анализа информации

**Задание:** Использовать внешнюю объектную модель Windows Script Host для сбора информации о пользователе

**Ход работы:**

1. Откройте Word и создайте новый документ (обычным, не программным способом). Затем нажмите клавиши <Alt>+<Fll>, чтобы открыть редактор Visual Basic, в окне Project Explorer щелкните правой кнопкой мыши по контейнеру вашего документа в контекстном меню выберите Insert | Module. Будет создан новый стандартный модуль. При помощи меню Insert | Procedure создайте в нем новую процедуру с именем wsh( ).
2. Добавьте при помощи меню Tools | References в проект этого документа ссылку на библиотеку Windows Script Host Object Model.
3. В процедуре wsh( ) создайте программные объекты Wscript.Network, Wscript.Shell и просмотрите свойства и методы этих объектов.
4. Добавьте в процедуру wsh () код, который бы принимал в текстовые переменные и печатал бы в документе Word или книге Excel значения свойств ComputerName, UserName и UserDomain.
5. Проанализируйте программный код и результат его работы

Sub WSH()

'Dim oNetwork As WshNetWork

'Dim oShell As WshShell

Dim sComputer As String

Dim sDomain As String

Dim sUser As String

Dim oColl As Variant

Dim sEnv As Variant

Set oNetwork = CreateObject("WScript.Network")

Set oShell = CreateObject("WScript.Shell")

sComputer = oNetwork.ComputerName

sUser = oNetwork.UserName

sDomain = oNetwork.UserDomain

Worksheets(1).Range("C7") = sComputer

Worksheets(1).Range("C8") = sUser

Worksheets(1).Range("C9") = sDomain

End Sub

# Работа с базами данных и применение объектной модели ADO

ADO расшифровывается как *ActiveX Data Objects* — набор программных объектов, построенных по технологии ActiveX(СОМ), которые позволяют получать  
данные из самых разных источников и управлять ими. Другие наборы  
программных объектов для доступа к источникам данных, которые часто используются  
в приложениях Office — это DAO и RDO, но эти программные  
продукты устарели и к использованию в современных приложениях не рекомендуются. В настоящее время появилась новая версия ADO — ADO.NET,  
которая сильно отличается от обычной ADO и предназначена для работы в  
.NET Framework. Однако по причине того, что [ADO.NET](http://ADO.NET):

обязательно требует установленной .NET Framework (чего на многих компьютерах нет);

обычными средствами с [ADO.NET](http://ADO.NET) из редактора Visual Basic работать  
нельзя, требуется Visual Studio;

отличается повышенной ресурсоемкостью.

ADO.NET рассматриваться не будет.

ADO умеет работать с самыми разными драйверами для подключения к базам  
данных, например, с драйверами OLE DB и ODBC. Поскольку ADO построен по технологии СОМ, его можно использовать в любых СОМ-  
совместимых языках программирования (VC++, Visual Basic, Delphi, VBA,  
VBScript, JScript, ActivePerl и т. п.).

К справке по ADO проще всего обращаться из Microsoft Access. Для этого нужно открыть Microsoft Access, создать в нем новую пустую базу данных, переключиться в окно редактора кода (при помощи клавиш <Ait>+<Fll>) и нажать клавишу <F1>. Вторая строка в оглавлении справки — это справка по ADO. В нее входят **ADO Programmer's Guide**, и **ADO Programmer's Reference** (справка по объектам ADO, их свойствам, методам и событиям).

Сама по себе объектная модель ADO очень проста и понятна. В нее включены всего три главных объекта.

* ***Connection*** — позволяет установить соединение с источником данных и управлять им. Все ошибки, которые возникают в ходе работы соединения, помещаются в сопутствующую коллекцию Errors.
* ***Command*** — представляет собой команду, при помощи которой производится выполнение определенной операции на источнике данных (выполнение запроса, хранимой процедуры, создание или изменение объекта, изменение данных и т. п.). Если источник данных SQL-совместимый, то объект Command, скорее всего, будет представлять команду SQL. Объекту command сопутствует коллекция Parameters — параметры, которые передаются запросу или хранимой процедуре.
* ***Recordset*** — является набором записей, полученных с источника или сгенерированных другим способом. Этому объекту сопутствует коллекция Fields, представляющая информацию о столбцах в этом наборе записей(имя, тип, размерность данных,…), а также сами данные.

**Объект *Connection* и коллекция *Errors***

Создание объекта *Connection* выполняется очень просто. Например, чтобы  
подключиться к базе данных *Northwind* на сервере SQL Server с именем  
LONDON, можно использовать код типа:

*Dim cn As New ADODB.Connection*

*cn.ConnectionString = "Provider=SQLOLEDB.1;Integrated*

*Security=SSPI; Initial Catalog=Northwind;Data Source=LONDON"*

*cn.Open*

В принципе, этого вполне достаточно, чтобы подключиться к базе данных и  
создать объект соединения, который можно будет потом использовать. Однако у   
большинства пользователей возникает вопрос: «а что написано в свойстве  
Connectionstring и как значение этого свойства можно написать самому?»

Самый простой вариант — вообще ничего самому не писать. Значение для этого свойства можно сгенерировать в автоматическом режиме. Выглядит это так:

Создаем любой пустой файл (например, текстовый). Для этого нужно  
щелкнуть правой кнопкой мыщи на пустом месте в окне проводника  
Windows и в контекстном меню выбрать **New** | **Text Document –** рис.1.И затем задать создавшемуся файлу любое название.

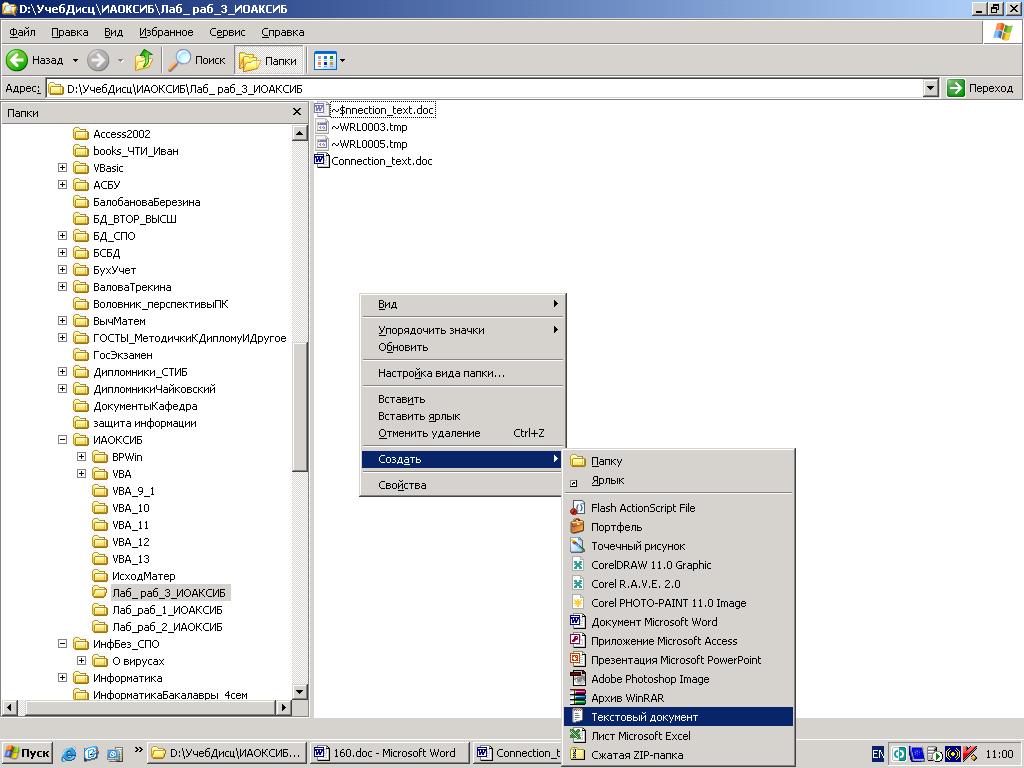


Рис.1 Создание пустого текстового документа

2. Переименовываем файл так, чтобы у него было расширение udl (от *User Data Link*— пользовательское подключение к данным). После переименования убедитесь, что иконка для него изменилась.

3. После того, как UDL-файл будет создан, просто щелкните по нему два раза левой кнопкой мыши. Откроется окно **Data Link Properties** с четырьмя вкладками.

4. На первой вкладке **Provider** выберите нужный тип базы данных (например, для базы данных SQL Server выберите **Microsoft OLE DB Provider for SQL Server,** для подключения к базе Oracle— **Microsoft OLE DB Provider for Oracle,** для подключения к базе Access — **Microsoft JET 4.0 OLE DB Provider).**

5. Далее перейдите на вкладку **Connection.** Для каждого типа базы данных эта вкладка выглядит по-своему. Например, для SQL Server она выглядит так, как представлено на рис. 2, а для Access — на рис. 3.

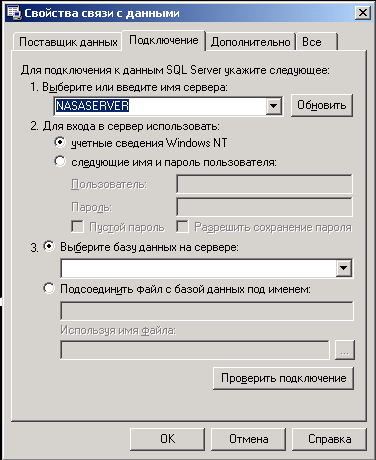


Рис.2- Окно свойств соединения с SQL-сервером

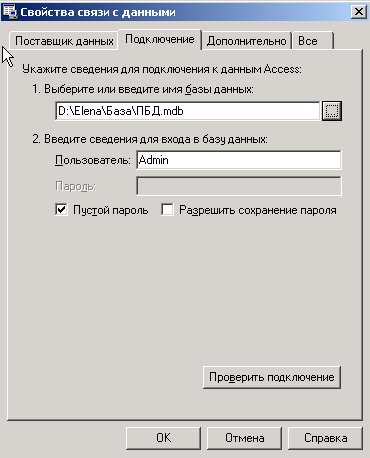


Рис.3 Окно свойств для подключения к базе данных Access

После того, как все параметры введены, рекомендуется нажать на кнопку **Test Connection,** чтобы проверить возможность подключения к базе данных. Если возникла ошибка то нужно исправить введенные вами параметры свойств соединения. После этого нужно нажать кнопку **ОК**, чтобы закрыть окно свойств соединения.

6. Последнее действие, которое нам потребуется сделать — щелкнуть правой кнопкой мыши по созданному UDL-файлу, в контекстном меню выбрать **Open With** | **Choose Program** и в появившемся списке выбрать **Notepad** (Блокнот) и нажать кнопку **ОК**. Созданный вами файл откроется в блокноте. В меню **Format** в блокноте снимите флажок **Word wrap** и скопируйте в буфер обмена последнюю строку этого файла (рис.4).

.

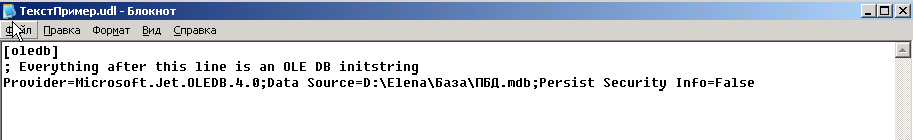


Рис 4. Сгенерированная строка подключения

7. Осталось вставить скопированное значение в окно редактора кода как значение свойства connectionstring и взять его в кавычки.

Конечно, можно написать строку подключения *(connection string)*'и вручную. Это просто набор параметров вида *"свойство=значение",* разделенных точкой с запятой (для значений в строке подключения кавычки не используются). Каждый из параметров означает следующее.

* *Provider* — драйвер для подключения к источнику данных. Для каждого типа источника данных (SQL Server, Access, Oracle) он свой. Мы с вами использовали драйвер OLE DB. Альтернатива ему— драйверы ODBC. Они работают медленнее и в основном используются для обеспечения обратной совместимости, но иногда без них не обойтись (например, когда подходящего драйвера OLE DB просто нет). Например, можно подключение по ODBC к таблице на листе Excel.

В принципе, в строке подключения значение для Provider можно не указывать вообще. Но тогда придется определить его как значение отдельного свойства объекта connection.

* *Integrated Security*— в данном случае это свойство используется, поскольку мы применяем для подключения к SQL Server аутентификацию Windows. Если бы мы использовали аутентификацию SQL Server, то это свойство нам бы было не нужно, но вместо него нам потребовалось бы использовать два других свойства: user id — идентификатор пользователя и Password— пароль.
* *Data source* — имя источника данных. В нашем случае это имя компьютера, на котором работает SQL Server. В других случаях оно могло бы быть именем экземпляра Oracle или файла базы данных Microsoft Access, все зависит от того, к какой базе данных вы подключаетесь.
* Initial Catalog — имя базы данных на этом сервере. В нашем случае это Northwind.

Когда вы генерируете строку подключения автоматически при помощи UDL-файла, в первый раз это может показаться долгим процессом. На самом деле это занимает не более минуты. Кроме того, при этом вы гарантируете, что в строке подключения не будет ошибок, и у вас есть возможность проверить подключение к базе данных прямо из свойств UDL-файла (кнопка Test Connection).

Фактически все, что нужно для открытия соединения с базой данных, мы сделали: создалиобъект connection, настроили для него свойство Сonnectionstring и вызвали метод Ореn( ). Однако для справки приведем информацию об основных свойствах и методах этого объекта.

* Provider— это свойство позволяет определить драйвер, который будет использован для подключения к базе данных. Мы с вами определили такой драйвер внутри значения connectionstring, но можно для этой цели использовать и отдельное свойство. Значения свойств Provider для подключения к разным источникам данных могут выглядеть так:
* "Microsoft. Jet .OLEDB. 4.0" — для подключений к файлам Access;
* "SQLOLEDB.1" — для подключений к SQL Server (как в примере);
* "MSDAORA.1" — для подключений к серверу Oracle;
* "ADsDSOObject" — для подключения к базе данных службы каталогов Windows.
* *Connectionstring* — это главное свойство объекта connection. Оно определяет параметры подключения к источнику.
* *Open( )* — этот метод позволяет открыть соединение с базой данных. Строку подключения можно не настраивать отдельно как свойство объекта connection, а просто передавать ее этому методу как параметр.
* *Close( )*— позволяет закрыть соединение. Учтите, что объект соединения при этом из памяти не удаляется. Чтобы полностью избавиться от этого объекта, можно использовать код:

Cn.Close

Set cn = Nothing

или просто:

Set cn = Nothing

а разрыв соединения произойдет автоматически.

Для объекта connection предусмотрено множество других свойств и методов. Опустим их рассмотрение. Единственное свойство, которое обязательно нужно знать, — это свойство Errors, которое возвращает коллекцию объектов Error — ошибок. Ошибки при установке или работе соединения встречаются очень часто (неверно введен пароль или имя пользователя, у пользователя недостаточно прав на подключение, невозможно обратиться к компьютеру по сети и т. п.), поэтому настоятельно рекомендуется реализовывать в программе обработку ошибок. Самый простой вариант реализации обработчика ошибок может выглядеть так:

*Dim en As ADODB.Connection*

*Set сn =CreateObject( "ADODB.Connection")*

*сn.Provider = "SQLOLEDB"*

*cn.ConnectionString = "User ID =SA; Password=password;" ,\_*

*& ''Data Source = LONPON1;Initial Catalog = Northwind"*

*On Error GOTO CnErrorHandler*

*cn.Open*

*Exit Sub,*

*CnErrorHandler:*

*For Each ADOErr In cn.Errors*

*Debug.Print ADOErr.Number*

*Debug.Print ADOErr.Description*

*Next*

На практике при возникновении ошибки пользователю предлагается ее исправить и еще раз произвести подключение.

Для получения информации о том, почему возникла ошибка, используется специальный объект ADOError (при возникновении ошибки он создается автоматически). Далее представлены самые важные свойства этого объекта.

* *Description*— описание ошибки. Обычно наиболее важная информация содержится именно в описании.
* *Number* — номер ошибки. По номеру удобно производить поиск в базе знаний Microsoft **(**[**www.microsoft.com/support**](http://www.microsoft.com/support)**)** и вИнтернете.
* *Source* — источник ошибки. Эта информация полезна только в том случае, если в коллекции Errors могут оказаться ошибки из разных источников.
* *SQLState и NativeError* — информация о возникшей ошибке, которая пришла с SQL-совместимого источника данных

# Лабораторная работа №2. Применение объектной модели ADO для получения данных из СУБД

**Цель лабораторной работы:** Изучить возможности объектной модели ADO.

**Задание:** Использовать объектную модель ADO для вставки по запросу записей из базы данных

**Ход работы:**

Для выполнения работы используйте базу данных Access, например, с именем Борей.mdb (находится в каталоге C:\Program Files\Microsoft Office11\SAMPLES).

1. Создайте новый документ Word и сохраните его.

2. Щелкните правой кнопкой мыши по любой панели инструментов или меню и в открывшемся списке доступных панелей инструментов выберите **Элементы управления**.

3.Поместите в документ Word новую кнопку.

4. Определите свойства для кнопки по вашему желанию. Например, как на рис.1.

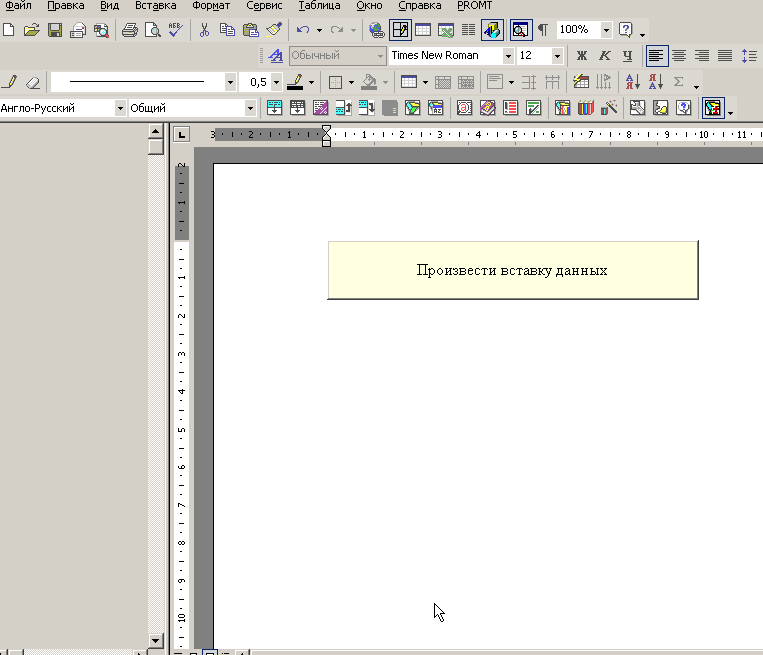


Рис.1. Кнопка в документе Word

5. При помощи меню **Вставка|Закладка** поместите под эту кнопку закладку с именем **Bookmark1**.

6. Щелкните правой кнопкой мыши на кнопке и в контекстном меню выберите **Исходный текст**. Откроется редактор кода VBA. Для события Click введите следующий код:

Private Sub CommandButton1\_Click()

Dim nEmpId As Integer

Dim sLastName As String

Dim sFirstName As String

Dim sTitle As String

nEmpId = CInt(InputBox("Введите номер сотрудника:"))

sLastName = "Иванов"

sFirstName = "Иван"

sTitle = "Начальник"

ThisDocument.Activate

ThisDocument.Bookmarks("Bookmarkl").Select

Selection.TypeText CStr(nEmpId) & " " & sLastName & sFirstName & " " & vbTab & sTitle & vbCrLf

End Sub

**Контрольное задание**

Измените код события кнопки таким образом, чтобы вместо присвоения переменным явно определенных значений им присваивались значения из таблицы ***Сотрудники*** базы данных Борей.mdb:

* Для переменной sLastName – значение из столбца Фамилия
* Для переменной sFirstName – значение из столбца Имя
* Для переменной sTitle – значение из столбца Должность

# Интеграция офисных приложений. Приложения Outlook и PowerPoint

Объектные модели приложений Outlook и PowerPoint имеют свои особенности. С этим связаны и ряд особенностей программирования в этих приложениях.

**Outlook**

Первая особенность программирования в Outlook заключается в том, где именно хранятся программные модули Outlook, в которых мы создаем код. Если вы помните, в Word они хранятся вместе с документами (или шаблонами, например, Normal.dot), в Excel— в файлах рабочих книг, в Access— в файлах баз данных MDB. В Outlook информация стандартных модулей хранится в файле личных папок PST, который по умолчанию создается в профиле данного пользователя. В результате, с одной стороны, работа с программным кодом VBA в Outlook упрощается: для данного пользователя на этом компьютере он становится доступен из Outlook всегда. С другой стороны, становится труднее предоставить этот код в распоряжение другого пользователя. В этой ситуации можно использовать два выхода:

* первый выход— *воспользоваться средствами экспорта и импорта программных модулей,* которые доступны из контекстного меню для модуля в **Project Explorer** (рис. 1.)
* второй выход— *создать контейнерное приложение,* например, файл Word или книгу Outlook, из которого программным образом запускать Outlook и выполнять в нем необходимые действия.

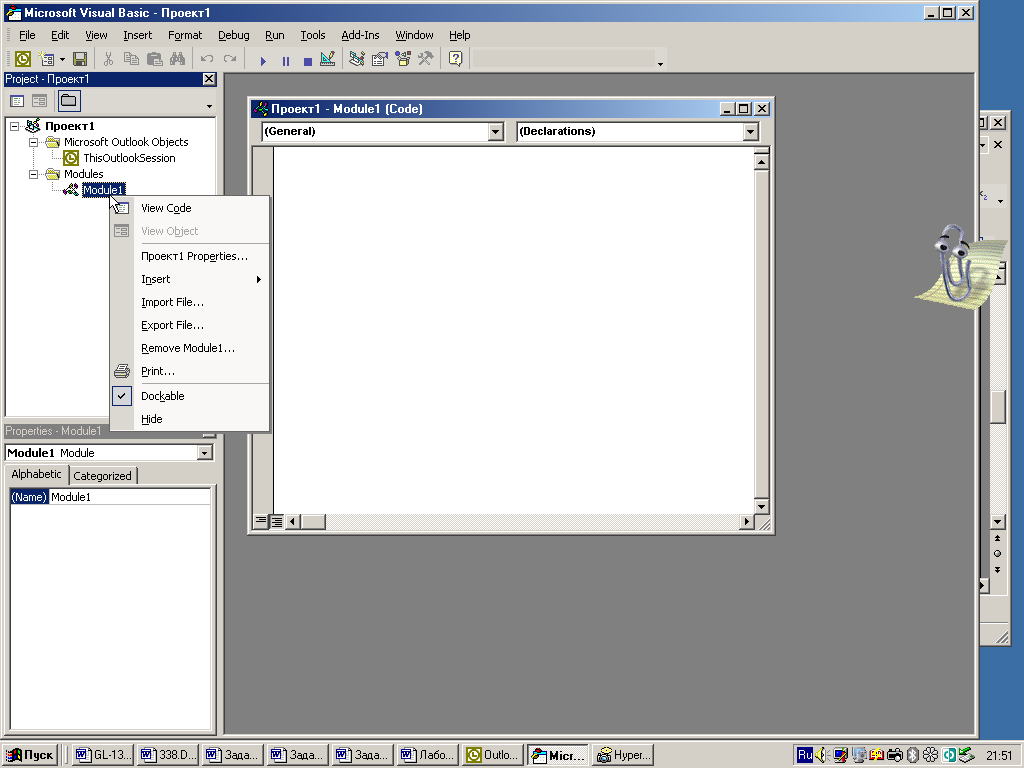


Рис.1 - Средства импорта и экспорта программного кода

Вторая особенность работы с VBA в Outlook заключается в том, что в Outlook  
реализована концепция пространства имен. Пространство имен в Outlook  
формально определяется как абстрактный корневой объект для любого ис-  
точника данных (например, папки в почтовом ящике Exchange или PST-  
файле на диске). Проще всего представить себе пространство имен Outlook  
как некий драйвер, который нужно использовать для подключения к данным.  
В настоящее время Outlook поддерживает только одно пространство имен —  
MAPI (Messaging Application Programming Interface, интерфейс прикладного  
программирования для работы с сообщениями), но разработчики Outlook  
требуют, чтобы это пространство имен явно указывалось при выполнении  
самых разных операций. Например, для запуска  
Outlook и открытия в нем папки **Inbox** (Входящие) из другой программы  
придется использовать следующий код (не забудьте подключить ссылку на  
библиотеку Microsoft Outlook 11.0 Object Library):

Dim oOutlook As New Outlook.Application

Set oNameSpace = oOutlook.GetNamespace("MAPI")

Set oInbox = oNameSpace.GetDefaultFolder(olFolderlnbox)

oInbox.Display

Третья особенность работы с Outlook заключается в некоторой терминологической путанице. Как правило, в документации по объектным моделям приложений Office термин item (элемент) применяется к элементам коллекций. В Outlook он получает второе значение: item— это все, что может хранитьсяв папках Outlook: почтовые сообщения (объект MailItem), контакты (объект ContactItem), встречи (объект Appointment) и т. п.

У Outlook есть еще одна особенность. На протяжении многих лет Outlook была программой, которая первой подвергалась атакам вирусов, троянских программ и прочего вредоносного программного обеспечения, приходящего по электронной почте. Иногда такие атаки бывали успешными, и Outlook на компьютере пользователя сам начинал рассылать электронные письма с вирусами (в том числе и при помощи кода VBA). Чтобы снизить вероятность развития событий по такому сценарию, разработчики Outlook сознательно поместили в его объектную модель ограничения, которые должны препятствовать распространению вирусов. Иногда эти ограничения (у них есть специальное название — *Outlook Object Model Guard)* мешают и нормальной работе программ VBA. Иногда по причине таких ограничений бывает удобнее использовать вместо объектной модели Outlook библиотеку CDO, которая имеется на любом компьютере с Windows 2000, ХР или 2003.

Интересной особенностью Outlook является и то, что в отличие от других приложений Office вы не сможете напрямую (при помощи ключевого слова New или команды CreateObject( )) создать ни одного объекта Outlook, кроме Application. Для создания всех остальных объектов придется использовать соответствующие методы уже созданных объектов.

На всякий случай также заметим, что макрорекордера в Outlook, как и в Access, к сожалению, нет. Всю необходимую дополнительную информацию вам придется искать при помощи документации.

**Объект Application, его свойства и методы**

Как и у всех приложений Office, на вершине объектной модели Outlook находится объект Application. Отличительной особенностью объекта Application в Outlook является относительно небольшое количество свойств и методов (за счет того, что часть из них переехала в объект Namespace). Наиболее часто используемые свойства объекта Application представлены далее.

* ***Explorers*** — это свойство возвращает коллекцию Explorers с объектами Explorer, каждый из которых представляет собой папку Outlook, открытую на просмотр пользователем. Основное назначение этой коллекции и объектов Explorer — проверка, не открыта ли уже пользователем в Outlook та или иная папка, и, в зависимости от результата, активизация этого окна (Explorer.Activate( )) или его закрытие (Explorer.Close( )). Метод ActiveExpiorer( ) объекта Application позволяет получить ссылку на окно, активное в настоящий момент, a GetExpiorer( ) — получить ссылку на объект Explorer для указанной папки Outlook (без его автоматической активизации).
* ***Inspectors*** — свойство, которое очень похоже на Explorers. Оно возвращает коллекцию inspectors с объектами inspector. Главное отличие в том, что объекты inspector представляют не открытые папки Outlook, как Explorer, а открытые на просмотр и редактирование элементы (например, почтовые сообщения). Объект inspector используется для таких же проверок, что и объект Explorer, его свойства и методы почти полностью совпадают со свойствами и методами объекта Explorer. Для тех же целей предусмотрены и методы ActiveInspector( ) И GetInspector( ) объекта Application.
* ***Reminders*** — позволяет вернуть коллекцию Reminders с объектами Reminder, представляющими текущие оповещения. Обычно это свойство используется для того, чтобы программным образом отключить все оповещения:

*Dim oOutlook As New Outlook.Application*

*Dim oReminder As Outlook.Reminder*

*For Each oReminder In oOutlook.Reminders*

*oReminder.Dismiss*

*Next*

* ***Session***— это свойство позволяет вернуть объект Namespace, представляющий пространство имен для текущего сеанса (т. е. пространство имен MAPI). Это свойство можно использовать вместо метода GetNamespace( ).

Аналогичное свойство Session предусмотрено для самого объекта Namespace и для множества других объектов Outlook.

**Методы объекта Outlook.Application**.

* Методы с префиксом **Active***...* — просто возвращают ссылку на активный в настоящее время объект Explorer или Inspector.
* **AdvancedSearch( )** — очень важный метод. Он позволяет производить поиск по папкам Outlook (что на практике требуется достаточно часто).
* **CopyFile( )** — позволяет скопировать файл с диска в папку Outlook. Можно использовать, например, для переноса всех файлов из каталога с документацией по проекту в общую папку Exchange Server или в библиотеку документов SharePoint Portal Server.

□ **CreateiIem( )** — метод, который используется очень часто. Он позволяет создать новые элементы в Outlook. Например, создать новый элемент типа контакт, заполнить его свойства, сохранить, а затем открыть для просмотра можно так:

Dim oOutlook As New Outlook.Application

Dim oContact As Outlook.ContactItem

Set oContact = oOutlook.CreateItem(olContactItem)

oContact.FirstName = "Лабораторная работа №4"

oContact.Email1Address = "chti@kaf.ru"

OContact.Save

oContact.Display

А теперь представьте, что вы создаете объекты контактов в цикле на основе записей из базы данных или строк в таблице Excel. Справочник контактов будет загружен в Outlook очень быстро и эффективно. Только не забывайте после каждого создания и сохранения контакта удалять его объект из оперативной памяти— иначе память на компьютере кончится и это приведет к ошибке. В нашем примере удалить объект контакта из памяти можно при помощи строки:

*Set oContact = Nothing*

* **GetNameSpace( ):—** метод, который используется, наверное, в большинстве программ VBA в Outlook. Позволяет получить объект пространства имен MAPI.
* **isSearchSynchronous( )—** используется для проверки режима поиска.
* **Quit( )** — осуществляет выход из Outlook

**Объект Namespace**

Как уже говорилось ранее, в Outlook используется понятие пространств имен — нечто вроде драйверов, которые, по замыслу разработчиков Outlook, должны обеспечивать доступ к различным хранилищам пользовательских данных. У каждого такого драйвера должны быть свои возможности. Однако уже на протяжении долгого времени в Outlook используется единственное пространство имен— MAPI, и пока нет никакой информации о том, что должно появиться что-то еще. Для наших целей пространство имен Outlook можно рассматривать просто как специальный служебный объект, в который "переехали" некоторые свойства и методы объекта Application.

Если в программе вам нужны свойства или методы объекта Namespace, получить ссылку на этот объект можно двумя способами:

1. **Воспользоваться методом** GetNameSpace( ) **объекта** Application:  
   Set oNameSpace = Application.GetNamespace("MAPI")
2. **Воспользоваться свойством** Session **того же объекта** Application:  
   Set oNameSpace = Application.Session

Эти две строки кода по функциональности полностью равнозначны, но в документации обычно всегда используется только первый способ.

Объект Namespace нужен для выполнения самых распространенных операций с электронной почтой: установка соединения с сервером электронной почты, отправка и получение электронной почты, выбор нужной папки, работа с адресными книгами и многих других. Кроме того, объект Namespace представляет еще и виртуальный корень папок Exchange, при помощи которого можно циклом проходить по всем папкам в Exchange.

В практической работе объект Namespace используется очень часто. Предположим, что вам нужно из Outlook скачать почтовые сообщения для всех учетных записей электронной почты и что-то сделать с каждым полученным сообщением. В нашем примере мы будем просто выводить тему каждого сообщения. На практике можно "разбирать" каждое сообщение в стандартном формате и помещать из него информацию в базу данных. Такое решение может пригодиться при обработке информации, получаемой из филиалов, с Web-сайта предприятия, который находится у провайдера, от торговых представителей, которые находятся в командировке, и т. п.

Как может выглядеть подобное сообщение?

Представим себе, что мы работаем из внешнего приложения. Удобнее всего нам будет использовать форму Word или Excel, чтобы работа с событиями производилась из окна редактора кода. Первое, что нужно сделать, — это поместить в проект ссылку на объектную библиотеку Microsoft Outlook 11.0 Object Library.

Затем мы занимаемся привычным делом — создаем самую обычную форму VBA и помещаем в нее кнопку commandButtonl. Перед созданием кода для этой кнопки необходимо сделать еще одно дело: зарегистрировать события для некоего абстрактного объекта Outlook.Items в разделе **General** — **Declarations** редактора кода. Это можно сделать при помощи строки:

*Public WithEvents oItems As Outlook.Items*

После этого объект oItems с событиями появится в окне редактора кода. Затем можно создавать код для события click( ) нашей кнопки. Он может быть, например, таким:

Private Sub CommandButtonl\_Click()

'Запускаем Outlook

Dim oOutlook As New Outlook.Application

'Очень важно: указываем, что события oItems — это события

'папки Inbox в Outlook

Set oltems = oOutlook.GetNamespace(""MAPI").GetDefaultFolder(olFolderlnbox).Items

'Дальше просто запускаем прием почты со всех учетных записей

Dim oNamespace As Outlook.NameSpace

Set oNamespace = oOutlook.GetNamespace("MAPI")

oNamespace.SyncObjects("Все учетные записи").Start

End Sub

'Ловим события появления нового сообщения   
Private Sub oItems\_ItemAdd(ByVal Item As Object)

MsgBox Item.Subject

End Sub

Конечно, если у вас на предприятии установлен Exchange Server, то для обработки всей входящей электронной почты правильнее использовать серверные скрипты Exchange и его событийную модель. Но реальная жизнь, как всегда, сложнее. Например, часто за Exchange Server отвечает администратор сети, который совершенно не собирается разрешать разработчикам настраивать на нем какие-то скрипты. Другой случай — когда на вашем предприятии работает не Exchange, а, к примеру, почтовая система на UNIX — SendMail, PostFix, CommunigatePro и т. п. Чтобы разобраться с их событиями, может потребоваться много времени и усилий, a Outlook всегда под рукой.

А теперь, рассмотрим информацию о самых важных свойствах и методах объекта Namespace.

□ ***AddressLists*** — это свойство возвращает коллекцию AddressLists, в которой находятся объекты AddressList, представляющие все адресные книги, доступные в настоящий момент. Например, получить список всех доступ ных в данный момент для пользователя адресных книг можно так (подразумевается, что вы работаете с Outlook из внешнего приложения):

Dim oOutlook As New Outlook.Application

Dim oNameSpace As Outlook.NameSpace

Dim oAddress As Outlook.AddressList

Set oNameSpace = oOutlook.GetNamespace("MAPI")

For Each oAddress In oNameSpace.AddressLists

Debug.Print oAddress.Name

Next

В объекте AddressList находится коллекция AddressEntries с объектами AddressEntry, представляющими записи в адресных книгах. При помощи этой объектной "ветви" вы можете программным способом добавлять записи в адресную книгу, удалять их, изменять свойства и т. п.

Если Outlook у вас работает сам по себе, в работе с коллекцией AddressLists нет никакого смысла — единственной доступной для пользователя адресной книгой будет являться книга **Контакты,** с которой проще работать другим способом (при помощи объектов СontactItem). Но если Outlook подключен к Exchange Server, то эта возможность становится очень интересной.

* ***CurrentUser*** — еще одно очень полезное свойство. Возвращает информацию о текущем пользователе (от имени которого открыт Outlook) в виде объекта Recipient, при помощи которого можно получить, например, адрес электронной почты данного пользователя, его имя и прочие атрибуты, которые предусмотрены для записи в списке адресов (если они были определены для пользователя). Например, получить доступ к информации об адресе электронной почты текущего пользователя можно так:

*Set oNameSpace = Application.GetNameSpace("MAPI")  
Set oRecipient = oNameSpace.CurrehtUser  
Debug.Print oRecipient.Address*

К сожалению, попытка выполнить этот код из внешнего приложения приведет к появлению окна сообщения с вопросом:, хотите ли вы предоставить программе доступ к адресам электронной почты в Outlook? Скорее всего, это окно вам совершенно не нужно, но оно было сделано специально в целях безопасности и избежать его вам не удастся. В вашем распоряжении два варианта: программно имитировать нажатие клавиш <Shift>+<Tab> и <Enter> в этом окне (заботливые разработчики отключили даже горячую клавишу для кнопки Да) или запускать этот код из уже работающего Outlook, тогда предупреждения не возникнет. Для программной имитации нажатий клавиш можно использовать объект wshshellобъектной библиотеки Windows Script Host, но поскольку при выполнении кода VBA ошибки не происходит, а просто "подвисает" последняя строка Debug.Print, то возникают дополнительные сложности. Их можно обойти только средствами Windows API.

* ***ExchangeConnectionMode*** — очень важное для практической работы свойство. Оно позволяет определить, настроен ли Outlook для работы с ExchangeServer и подключен ли он к Exchange Server в настоящий момент.
* ***Folders*** — это, наверное, самое главное свойство объекта Namespace. Возвращает коллекцию Folders с объектами MAPIFolder, представляющими папки верхнего уровня в Outlook (у каждой папки, в свою очередь, есть свое свойство Folders, так что вы вполне можете пройти циклом по всем без исключения папкам Outlook). Кроме того, что у папок Outlook есть множество своих собственных важных свойств и методов, через свойство items папки вы можете получить доступ ко всем элементам папки (сообщениям, контактам, элементам календаря и т. п.). offline — позволяет выяснить, подключен ли в настоящее время Outlook  
  к серверу электронной почты или нет.
* ***SyncObjects*** — возвращает одноименную коллекцию с объектами SyhcObject (объекты синхронизации)..Сами эти объекты представляют собой группы отправки (то, что при помощи графического интерфейсаOutlook можно найти в меню Сервис | Отправить/Получить). Самое важное, что можно сделать при помощи объектов SyncObject,— это программно инициировать соединение с сервером электронной почты или разорвать его.

Обычно в приложениях, использующих объектную модель Outlook, без методов объекта Namespace также не обойтись.

* ***Addstore( ) и AddStoreEx( )*** — позволяют программно открыть файл PST хранилища сообщений Outlook на диске. Обратите внимание на то, что если такого файла на диске нет, то при вызове этого метода Outlook просто создаст его. AddstoreEx( ) отличается тем, что позволяет указать кодировку для файла сообщений. Закрыть PST-файл можно при помощи метода RemoveStore( ).
* ***GetDefaultFolder*** ( ) — важнейший метод, возвращающий объект MAPiFolder для одной из двенадцати встроенных (используемых по умолчанию) папок Outlook: **Входящие, Контакты, Календарь, Отправленные** и т. п. Например, подключиться к папке **Контакты** (и для наглядности открыть ее) можно так:

Dim oOutlook As New Outlook.Application

Set oNameSpace = oOutlook.GetNamespace("MAPI")

Set oInbox = oNameSpace.GetDefaultFolder(olFolderContacts): *z*

oInbox.Display

**Коллекция Folders и объект MAPlFolder**

Обычно, когда мы программным образом работаем с Outlook, нам нужно что-то сделать с его элементами — почтовыми сообщениями, контактами, встречами в календаре и т. п. Все эти элементы расположены в папках Outlook, либо встроенных, либо созданных пользователем). Папкам в объектной модели Outlook соответствуют объекты MAPlFolder, которые сведены в коллекцию Folders.

В Outlook папки могут быть вложены друг в друга. На самом верху расположены папки верхнего уровня (это не **Входящие, Контакты, Черновики** и т. п., как вы могли подумать, а папки более высокого уровня, например, **Личные папки, Общие папки, Mailbox — Administrator).** Доступ к колтекции Folders, представляющей собой папки самого верхнего уровня, производится через свойство Folders объекта Namespace*:*

*Dim oOutlook As New Outlook.Application*

*Dim oNameSpace As Outlook.NameSpace*

*Dim oFolder As Outlook.MAPlFolder*

*Set oNameSpace = oOutlook.GetNamespace ("MAPI")!*

*For Each oFolder In oNameSpace.Folders*

*Debug.Print oFolder.Name  
Next*

Если вам нужна конкретная встроенная папка, например **Inbox,** то проще всего **найти ее при помощи метода** GetDefaultFolder( ) объекта Namespace:

Dim oOutlook As New Outlook.Application

Dim oNameSpace As Outlook.NameSpace

Dim oFolder As Outlook.MAPlFolder

Set oNameSpace = oOutlook.GetNamespace("MAPI")

Set oFolder = oNameSpace.GetDefaultFolder(olFolderlnbox)

Debug.Print oFolder.Name

Очень часто требуется пройти циклом по всем папкам в почтовом ящике Outlook (а иногда даже в нескольких ящиках), просмотреть все сообщения в каждой папке (и во всех вложенных папках) и что-то сделать с этими сообщениями, например, собрать все письма от определенного отправителя, разбросанные по разным папкам, в одну специальную. В нашем примере мы просто пройдем циклом по всем папкам Outlook и выведем их имена, но, конечно, этот пример несложно переделать так, чтобы он выполнял какое-нибудь действие над ними в зависимости от условия. В нашем случае удобнее всего использовать две процедуры, одна из которых будет вызывать саму себя:

*Public Sub StartProc1( )*

*Dim oOutlook As New Outlook.Application*

*Dim oNamespace As Outlook.NameSpace*

*Dim oChildFolder As Outlook.MAPlFolder*

*Set oNamespace = oOutlook.GetNamespace("MAPI")*

*'Перебираем каждую папку верхнего уровня и*

*'вызываем для нее процедуру DoFolderO*

*For Each oChildFolder In oNamespace.Folders*

*DoFolder oChildFolder*

*Next*

*End Sub*

'Эта процедура выводит имя всех сложенных папок и

'для каждой из них опять выз~~ы~~вает саму себя

Public Sub DoFolder(ByVal oFolder As MAPlFolder)  
Dim oChildFolder As Outlook.MAPIFolder  
For Each oChildFolder In oFolder.Folders  
 Debug.Print oChildFolder.Name  
'Для каждой вложенной папки опять вызываем процедуру DoFolderO  
 DoFolder oChildFolder  
Next

End Sub

**PowerPoint**

PowerPoint — это программа для работы с презентациями (т. е. с наборами графических изображений — слайдов, иногда со звуковым сопровождением).

В PowerPoint система объектов выглядит следующим образом:

* объект самого высокого уровня — Application с набором свойств и методов, очень похожим на аналогичные объекты в Word и Excel;
* уровень ниже — коллекция Presentations с объектами Presentation.  
  Можно сказать, что эти объекты по месту в иерархии примерно аналогичны объекту Workbook в Excel;
* в в объект Presentation встроена коллекция slides с объектами slide. В качестве аналога можно привести листы worksheet в книгах Excel;
* в объект *sh&ue* встроена коллекция shapes с объектами shape. Объекты  
  Shape представляют собой все элементы слайда (всего их 22 типа: изображение, надпись, диаграмма, заголовок, таблица, автофигура и т. п.).

Вокруг этих четырех объектов — Application, Presentation, Slide и Shape — и строится вся объектная модель PowerPoint

Проиллюстрируем работу с PowerPoint на примерах из практики.

Предположим, что нам нужно создать презентацию PowerPoint на основе набора JPG-картинок, которые будут лежать в каталоге [C:\Slides](file://C:/Slides) (например, они получены со сканера или цифрового фотоаппарата). Имена JPG-файлов следуют по порядку, например, с DSCN2440.JPG по DSCN2480.JPG. Файлов вкаталоге может быть произвольное количество, поэтому нам нужно взять все файлы из этого каталога. Наша задача — поместить их в презентацию по порядку. Задача усугубляется тем, что JPG-файлы разного размера (по высоте *i* ширине), а слайды желательно сделать одинаковыми.

Как ни удивительно, код VBA для PowerPoint удобнее запускать не и: PowerPoint, а из внешнего приложения, поддерживающего VBA, например из Word или Excel. Таким образом, на момент запуска у нас гарантировано не будет активных презентаций и мы ничего не перепутаем при вставке.

Наше решение может выглядеть следующим образом:

1. Создаем новый документ в Word или Excel, в Него помещаем кнопку шп обеспечиваем другой графический интерфейс по вкусу. Главное — это ж забыть добавить в проект ссылки на две объектные библиотеки:

*Microsoft PowerPoint 11.0 Object Library* (msppt.olb) — для объектов са мого PowerPoint;

*Microsoft Scripting Runtime* (ScrRun.dll) — для того, чтобы можно былс пользоваться объектом FileSystemObject и другими возможностями дл) работы с файловой системой.

2. Конечно, первое, что нам потребуется — запустить PowerPoint. Делается это точно так же, как и для Word, Excel, Access и т. п.:

Dim oApp As New PowerPoint. Application

oApp.Activate

oApp.Visible = msoTrue

3.Следующее действие — создание новой пустой презентации:

Dim oPresent As PowerPoint. Presentation

Set oPresent = oApp.Presentations.Add()

Все абсолютно стандартно, как будто мы создаем новый документ Word. А вот дальше начинаются моменты, специфические для PowerPoint.

4. Следующим действием должно быть создание слайда. Но нам придется создавать столько слайдов, сколько файлов находится в каталоге [C:\Slides](file://C:/Slides). Конечно, слайды будут создаваться в цикле. Вначале мы получаем при помощи библиотеки Scripting Runtime (можно было бы обойтись и средствами Office, но так проще) коллекцию всех файлов этого каталога:

Dim oFSO As New Scripting.FileSystemObject

Dim oFolder As Scripting. Folder

Dim oFile As Scripting.File

Set = oFSO.GetFolder("C:\Slides")

For Each oFile In oFolder.Files

...

Next

Если мы вместо многоточия вставим строку, например, такого вида:

MsgBox oFile.Name

то можно будет убедиться, что мы получили набор файлов в правильномпорядке.

5. Далее нам все-таки нужно создать слайды. Делается это при помощи метода Add() коллекции slides. В документации к русскому PowerPoint 2003 описание этого метода по непонятной причине отсутствует (несмотря на то, что справка по VBA все равно приводится на английском), но из всплывающей подсказки можно догадаться, что этот метод принимает два обязательных параметра: номер слайда в презентации (нумерация должна начинаться с 1), и одно из значений перечисления ppSlideLayout (из нескольких десятков), которое определяет шаблон слайда.

Номер слайда придется обеспечивать счетчиком, а лучший для нас шаблон — пустой:

*Dim nCounter As Integer*

*nCounter = 1*

*For Each oFile In oFolder. Files*

*Set oSlide = oApp.ActivePresentation.Slides.Add(nCounter,ppLayoutBlank)*

*...*

*nCounter = nCounter + 1*

*Next*

6. А теперь самое главное — вставляем в слайд изображение и настраиваем его размеры. Для этой цели можно использовать метод AddPicture () коллекции shapes каждого слайда:

*oSlide.Shapes.AddPicture FileName:="C:\Slides\" & oFile.Name, \_*

*LinkToFile:=msoFalse, SaveWithDocument:=msoTrue, \_*

*Left:=10, Top:=10, Width:=700, Height:=520*

Параметр FileName — это имя передаваемого файла. Именно он и будет меняться в цикле. Параметр LinkToFile определяет, будет ли файл изображения помещен внутрь презентации (msoFalse) или в презентацию будет помещена ссылка на него (msoTrue). Конечно, если вставляемые файлы не очень большие, то с точки зрения удобства и производительности презентации лучше помещать изображения внутрь презентации (файла РРТ). Параметр saveWithDocument определяет, сохранять ли изображения вместе с презентацией (в нашем случае сохранять). А параметры Left, Top, width и Height нужны, чтобы сделать изображения одинаковего размера (нужные значения подбираются вручную).

Естественно, код этого пункта помещается вместо многоточия в цикл пункта

5. Чтобы не возиться с удалением обработанных файлов, я бы поместил в цикл еще одну очевидную строку:

*oFile.Delete*

Итоговый код для нашей задачи может выглядеть так:

*Dim oApp As New PowerPoint.Application*

*oApp.Activate*

*oApp.Visible = msoTrue*

*Dim oPresent As PowerPoint.Presentation*

*Set oPresent = oApp.Presentations.Add( )*

*Dim oFSO As New Scripting.FileSystemObject*

*Dim oFolder As Scripting. Folder*

*Dim oFile As Scripting.File*

*Set oFolder = oFSO.GetFolder("*[*C:\Slides*](file://C:/Slides)*")*

*For Each oFile In oFolder.Files*

*Set oSlide = oApp.ActivePresentation.Slides.Add(nCounter, ppLayOutBlank)*

*oSlide.Shapes.AddPicture FileName:="C:\Slides\" & oFile.Name,LinkToFile:=msoFalse,\_SaveWithDocument:=msoTrue,\_\_*

*Left:=10, Top:=10, Width:=700, Height:=520*

*oFile.Delete*

*Next*

# Лабораторная работа №3. Механизмы интеграции офисных приложений

**Цель лабораторной работы:** изучить возможности объектных моделейприложений Outlook и PowerPoint.

**Задание:** На предприятии в одном из офисных приложений есть таблица со списком организаций, в которой хранится информация о представителях этих организаций и их координатах. Нужно сделать так, чтобы эта информация стала доступной из Outlook.

**Ход работы:**

Напишите макрос Outlook ImportContacts( ), который бы:

1. Создавал новую папку Outlook с элементами типа **Контакт** под именем **Контакты клиентов.**

2. Создавал в этой папке контакты для всех записей таблицы клиенты базы данных Борей.mdb. Чтобы упростить задачу, достаточно поместить в создаваемые контакты только информацию о названии организации, имени представителя и телефоне.

*Примечание*

*На практике, вполне возможно, удобнее было бы создать общую папку на сервере Exchange Server и программным образом создать в ней контакты, чтобы можно было предоставить доступ к ним сразу группе пользователей. В нашем случае для простоты мы создаем элементы контактов в локальной папке Outlook. Код VBA для создания контактов в любом случае будет идентичен.*

***Задание №2.***

Создайте макрос в PowerPoint, который бы добавлял во все слайды активной презентации в правый нижний угол надпись "Информационное обеспечение проекта ФИО год группа" .

Примечание

*В реальной работе, возможно, удобнее было поместить этот макрос во внешнее приложение VBA, например, в документ Word или лист Excel, чтобы не копировать этот код для каждой новой презентации. Но в этой работе для простоты пусть код выполняется из самого PowerPoint.*

***Задание 3 (По вариантам)***

1.Найти все новые сообщения от клиента (клиент1)

2. Найти все новые сообщения с приглашением «на конференцию»

3. Все сообщения с темой, в которой встречается слово «отчет» поместить в БД

4. Вывести в Word или Excel список всех адресатов из Адресной книги Outlook, место работы у которых компания ЗАО «СтройИнвест»

5.Все сообщения из папки InBox от клиента (клиент2) поместить в папку Клиент2

6. Вывести адрес электронной почты для всех адресатов из Адресной книги Outlook, у которых имя начинается с буквы «А»

# Основы разметки XML

XML - это аббревиатура от extensible Markup Language (расширяемый язык разметки). Однако XML можно назвать и расширяемым метаязыком (extensible Meta Language), так как он настолько стандартизирован и гибок, что может использоваться при создании других языков. Если говорить точнее, то XML позволяет создавать диалекты языков, которые точно следуют всем правилам, касающимся структуры, синтаксиса и семантики, которые определены консорциумом *IV3C* С некоторыми правилами вы познакомитесь уже в настоящей главе, а важность остальных правил вы поймете, когда прочтете книгу до конца и выполните все представленные в ней упражнения.

XML обеспечивает возможность сохранения данных в обычном текстовом формате. Любой человек или приложение, способное работать с текстовыми документами, способны прочитать XML-документы. Вам не требуется специальное программное обеспечение для того, чтобы получить доступ к подобным данным. Появляется возможность просмотра документа и внесения в него необходимых изменений с помощью простого текстового редактора на любой платформе. В поставку большей части операционных систем входят бесплатные текстовые редакторы. Другие редакторы, предназначенные для выполнения именно такой работы, доступны для загрузки из Internet, причем достаточно часто совершенно бесплатно.

Так же XML позволяет сохранять в текстовых файлах *структурированные* данные. XML - это набор правил для создания текстовых форматов, простых для обработки компьютерами различных типов. Получаемые текстовые файлы структурированы таким образом, что они:

■ точно выражены;

■ расширяемы;

■ платформно-независимы.

У XML есть несколько похожих характеристик HTML. Если вы хорошо знакомы с HTML, вы знаете некоторые синтаксические правила, которые в равной мере применимы и к XML. Ниже представлен пример разметки.

*<р>*

*Это пример текста, который*

*<ЕМ>отмечен</ЕМ>*

*для использования в Web-презентации.*

*</p>*

На самом деле, это пример хорошо оформленного фрагмента кода на HTML и XML. Он соответствует синтаксическим правилам, общим для обоих языков.

Однако XML разрабатывался для преодоления всех ограничений, свойственных HTML. Автор документа может определить ряд элементов XML; строго фиксированного набора заранее определенных дескрипторов, как в HTML, в XML не существует. Так как вы, как автор документа, можете определять свои собственные дескрипторы, вы может и выбирать для них наиболее описательные имена. Вместо того чтобы размечать абзацы с помощью дескриптора <р>, вы можете использовать такие имена дескрипторов, как <paragraph> или <рага>. Либо вы можете вообще использовать для разметки разных абзацев дескрипторы с различными именами. Например, вместо одного имени дескриптора <paragraph> вы можете использовать следующие варианты.

■ <introciuction>

■ <summary>

■ <sales\_info>

■ <address>

■ <description>

■ <etc>

Элементы разметки XML выступают в роли строительных блоков языка, являясь аналогами существительных в обычных "человеческих" языках. В XML также существуют и атрибуты, похожие на атрибуты HTML и предназначенные для изменения или классификации элементов, точно так же как прилагательные характеризуют существительные в обычных "человеческих" языках. Атрибуты, если они, конечно же, есть, должны указываться в открывающем дескрипторе элемента. В листинге 1.3 приведен пример немного более сложного элемента, содержащего атрибуты и другие элементы. Если элемент содержит дочерние элементы, те называются *вложенными.*

*1: <account type= "checking" currency ="U.S. Dollars">*

*2: <name>Smith</name>*

*3: <balance>34576.89</balance>*

*4 : </account>*

Строки с J по 4 образуют целый элемент account. Два атрибута дополняют элемент account точно так же, как прилагательные дополняют существительное, предоставляя новые сведения об элементе. Атрибуты называются type и currency, а их значения — checking и U.S. Dollars соответственно. Значения атрибутов всегда заключаются в одинарные или двойные кавычки. Элемент account обладает только содержанием (строки 2 и 3). Другими словами, этот элемент не содержит ни одного знака или текстовых данных, а только дочерние элементы, такие как name и balance (строки 2 и 3). Все элементы, account, name и balance, содержат открывающий и закрывающий дескрипторы. Все элементы ХМ L должны быть завершены.

Элементы XML могут содержать текст, другие элементы, любую комбинацию текста и других элементов или же быть просто пустыми элементами. Пустой элемент — это такой элемент, между открывающим и закрывающим дескрипторами которого отсутствует какое-либо содержание. Синтаксис пустого элемента следующий: <имя\_элемента/>

Пример пустого элемента XML:

*1: <data month="Сентябрь" year="2002"/>*

Элемент data пустой. Знак завершения (/) указывается прямо в открывающем де­скрипторе. Это краткая форма записи следующей инструкции: <data month="Сентябрь" year="2002"x/data>. Несмотря на то, что обе инструкции функционируют совершенно одинаково, для представления пустого элемента XML принято использовать краткую форму. Даже если такой элемент обладает атрибутами с определенными значениями, он все равно считается пустым, так как между открывающим и закрывающим дескрипторами отсутствуют какие-либо данные.

Рассмотрим структуру разметки документа. Один элемент содержит все остальные элементы в документе. Этот один элемент, находящийся на наивысшем уровне в структуре документа, называется *корневым элементом.* На HTML-странице корневым всегда является элемент HTML. В XML-документе вы сами принимаете решение о том, как назвать корневой элемент. В примере программного кода, приведенного в листинге ниже, корневой элемент называется Invoice.

*<Invoice>*

*<From>Devan Shepherd</From>*

*<To>Sally Jones</To>*

*<Data year="2001" month="jLily" day="26" />*

*<Amount currency="USD">$100.00</Amount>*

*<TextRate>2K<TextRate>*

*<TotalDue currency="USD\*\*>121.00</TotalDue>*

*</ Invoice>*

***Хорошо оформленный XML-документ должен содержать один и только один корневой элемент, в котором будут все остальные элементы.***

Вы уже увидели, что XML состоит из интуитивно понятных самоописываюшихся эле­ментов, которые содержатся в одном корневом элементе. Для выполнения упражнения вам потребуются следующие инструменты.

■ Текстовый редактор (Блокнот, SimpleText и т.д.) Вы можете использовать любой редактор, который способен сохранять текстовые документы ASCII, но если вы используете текстовый процессор, ни в коем случае не сохраняйте данные в двоичном формате.

■ Браузер Microsoft Internet Explorer (версии 5.0 или более поздней). Если на вашем компьютере еще не установлена соответствующая версия этого браузера, загрузите се с Web-узла <http://www.microsoft.com/downloads/searech.asp>.

Документ, который вы создадите, будет очень простым: в нем будут представлены такие сведения, как ваши имя и фамилия, а также шутка. Пусть имени соответствует элемент <first>, а фамилии <last>. Шуткой может быть любая текстовая строка, указанная в элементе <joke>. Начните с запуска текстового редактора и приступите к созданию документа XML. Не забывайте о том, что в документе может содержаться всего один корневой элемент и по одному элементу для каждого отдельного компонента данных. Вы должны выбрать наиболее подходящий вариант для имени корневого элемента. Вам также необходимо завершить каждый элемент; в нашем случае для этого используются закрывающие дескрипторы.

Закончив создание документа, сохраните его под именем me.xml, после чего запустите Internet Explorer для просмотра результата. Если вы сохраните файл с другим расширением, вы не сможете увидеть нужного результата.

Для того чтобы увидеть результат в Internet Explorer, введите полный путь в адресной строке, расположенной в верхней части окна браузера. При просмотре документа с помощью Internet Explorer документ проходит через специальный обработчик, который называется *парсером.* Парсер проверяет ваш документ на соответствие всем синтаксическим правилам XML. Если разметка вашего документа правильна, парсер преобразует ваш документ таким образом, чтобы его было легко читать, отображая дескрипторы разметки цветом, отличным от цвета заключенного между ними текста. Подобный процесс преобразования называется *преобразованием расширяемого языка таблиц стилей* XSLT. Подробные сведения об XSLT вы найдете в уроке дня 16-го "Язык преобразований XSLT".

В листинге приведен пример простого документа; ваш вариант может отличаться другими именами элементов, другой структурой вложений или более удачной шуткой.

Первый документ XML:

Результат отображения этого документа XML в окне браузера приведен на рис.2. Обратите внимание, что в случае правильно оформленного документа XML Internet Explorer преобразует документ, применяя к нему определенные стили.

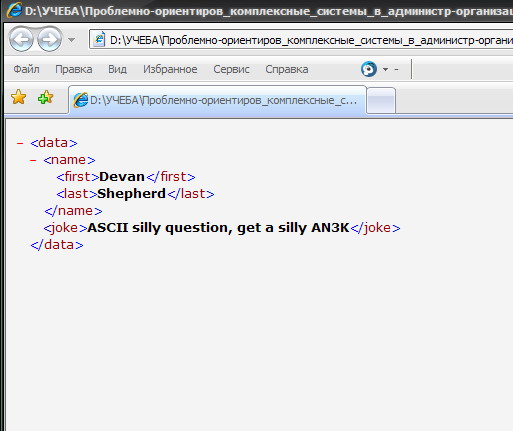
**

Рис.2. Результат выполнения кода.

**СИНТАКСИС ЯЗЫКА XML**

В этом разделе вы познакомитесь с некоторыми синтаксическими правилами XML, а также увидите, каким образом парсер обрабатывает документы XML. Для того чтобы извлечь максимум пользы из этих сведений, вы можете создать несколько документом и обработать их с помощью парсера.

Если вы создаете файл XML на основе существующего документа, вам необходимо очень внимательно изучить оригинал, чтобы понять структуру экземпляра для разметки. Этот процесс называется анализом документа и является важным компонентом процесса разметки. Для разметки рецепта приготовления блюда анализ рецепта будет совсем несложным. Начните с идентификации компонентов рецепта. Вы уже знаете, что основные строительные блоки называются элементами. Элементы чем-то сродни существительным в обычных языках; это контейнеры информации, которые содержат данные или другие элементы.

Для того чтобы разобраться, какие именно элементы должны быть в рецепте, вы должны разобраться со списком необходимых дескрипторов элементов. Для каждого основного компонента рецепта вам необходимо определить следующее.

■ Как должен называться элемент?

■ Где должен начинаться элемент (а значит, где должен находиться его открывающий дескриптор)?

■ Где должен заканчиваться элемент (а значит, где должен находиться его закрывающий дескриптор)?

■ Какие именно сведения должен содержать элемент?

■ Какова взаимосвязь между элементом и другими элементами рецепта?

На рис. 3 приведен типичный рецепт в виде обычной блок-схемы. Как видно из этой блок-схемы, элемент recipe содержит такие элементы, как title, ingredients, preparation и serving. Элемент ingredients содержит вложенные элементы item и соответствующие им дочерние элементы quantity.

Еще один полезный пример документа XML приведен на рис.4. Здесь рецепт представлен в виде древовидной структуры.

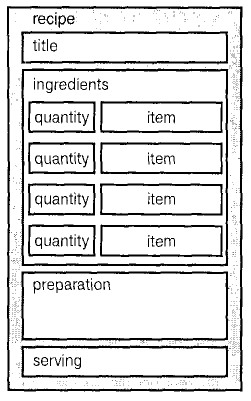
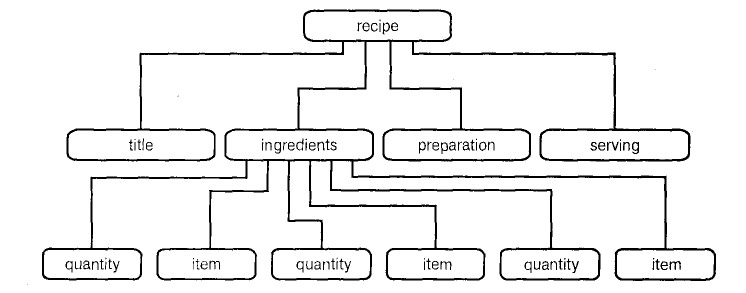


Рис.3. Блок-схема рецепта приготовления блюда

Рис.4. Древовидная структура рецепта приготовления блюда.

Из показанной на рисунке древовидной структуры можно увидеть взаимосвязи между элементами. В любом случае мы имеем дело только с одним корневым элементом recipe, который выступает в роли контейнера для всех остальных элементов. Близкое рассмотрение элемента ingredients позволяет увидеть, что он содержит вложенные элементы item и соответствующие им дочерние элементы quantity. Для того чтобы экземпляр XML считался хорошо оформленным, дочерние элементы и под элементы корневого элементы должны быть вложены без взаимного "перекрытия" дескрипторов. Сегодня вы познакомитесь с очень простым способом проверки правильной организации вложения элементов, однако сначала вам следует создать документ XML, введя приведенный код с помощью текстового редактора.

1. Начните с создания корневого элемента, назвав его recipe. Он должен начитаться с открывающего дескриптора <recipe> и заканчиваться закрывающим дескриптором </recipe>.
2. Корневой элемент recipe будет содержать все остальные элементы. Укажите откры­вающий и закрывающий дескрипторы для всех дочерних элементов корневого элемента recipe, которые называются title, ingredients, preparation и serving.
3. Затем укажите открывающий и закрывающий дескрипторы для всех пар дочерних элементов item и quantity в рамках открывающего и закрывающего дескрипторов элемента ingredients.
4. Расставив по местам все необходимые открывающие и закрывающие дескрипторы, заполните их необходимыми данными, приведенными раньше.
5. Сохраните полученный документ под именем bean.dip.xml.

Угловая скобка (<) в начале дескриптора в языках разметки называется MDO (Markup Declaration Open — Открытие декларации разметки). Угловая скобка (>) в конце дескриптора в языках разметки называется МDC (Markup Declaration Close - Закрытие декларации разметки). В качестве примера давайте рассмотрим следующий элемент: <item>sour cream</item>

Имена, используемые в документе recipe, просты и достаточно описательны. Несмотря на то, что XML считывается компьютером, он должен быть понятен и человеку. Поэтому вам следует выполнять несколько правил при определении имен элементов документе XML

■ Имя элемента XML может начинаться с буквы, знака подчеркивания (\_\_) или двоеточия (:).

■ После первого символа в имени элемента могут быть буквы, цифры, знаки переноса (-), знаки подчеркивания ( ), точка (.) или двоеточие (:).

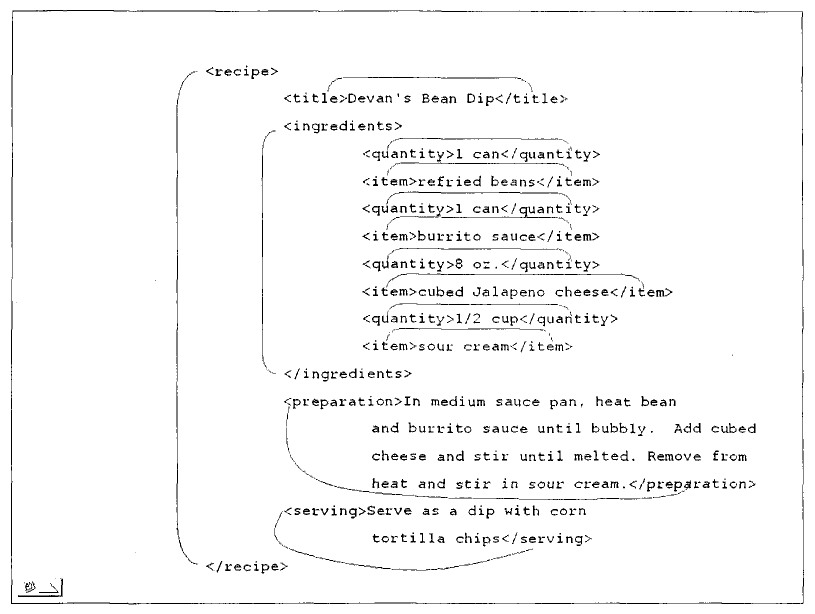


Рис.5. проверка на правильность вложения.

Вы уже создали несколько простых документов XML, которые хорошо оформлены, а значит, их может обработать парсер из состава Internet Explorer 5.0. Хотя Internet Explorer прекрасно определяет XML, хорошим тоном считается явно указывать, что это документ XML. Некоторые парсеры *требуют* наличия в документе соответствующей инструкции, которая называется декларацией XML.

Добавьте декларацию XML в начало документа bean\_dip.xml прямо перед открывающим дескриптором элемента recipe. Она выглядит следующим образом:

*<?xml version="1.0" ?>*

Обратите внимание на то, что первой строкой в документе является декларация XML. Внесите необходимые исправления в разметку рецепта и сохраните полученный результат под именем bean\_dip2.xml.

Декларация XML в первой строке указывает обработчику на то, что документ XML. Несмотря на то, что некоторые обработчики не требуют наличия такой декларации, ее включение в код документа считается хорошим тоном.

Синтаксис декларации XML несколько отличается от синтаксиса элемента. Декларация - это инструкция обработки (Processing Instruction - PI), которая передает специальные сведения обработчику, сообщая о том, что документ составлен в соответствии с правилами, определяемыми версией 1.0 стандарта XML. Подобная инструкция обработки всегда указывается в первой строке документа.

Для того чтобы инструкции обработки были понятны парсерам XML, в них должны использоваться только символы нижнего регистра. Кроме того, в любой декларации XML указано, что должна использоваться версия 1.0 стандарта XML. Это связано с тем, что на сегодня это единственная существующая версия стандарта. Возможно, в будущем эта ситуация изменится, поэтому указывать версию стандарта нужно обязательно. Атрибут encoding, используемый для указания нужной кодировки символов, не может принимать значения UTF-8 или UTF-16.

Атрибут standalone указывает обработчику XML, необходимо ли использовать внешнее определение типа документа DTD при проверке экземпляра. В большинстве случае хорошо оформленные документы XML не требуют использования определений DTD.

Как и другие языки программирования, XML позволяет добавлять к разметке документа различные комментарии. Однако в XML комментарии могут предназначаться и для обработчиков XML.

Комментарии в экземплярах XML имеют практически такой же вид, как и в HTML:

*<!--Это комментарий-->*

Воспользовавшись текстовым редактором, добавьте в документ bean\_dip2.xml комментарий, который будет напоминать вам об остром соусе при приготовлении блюда. Добавьте комментарий перед элементом serving, после чего просмотрите полученный результат с помощью Microsoft Internet Explorer 5.0. Ваш комментарий может выглядеть так:

*<!--При желании добавить к полученной смеси острый соус-->*

Вы уже знаете, что атрибуты напоминают прилагательные в обычном "человеческом" языке, поскольку они дополняют элементы, которые мы считаем аналогами существительных. В XML атрибуты всегда заключены в открывающий дескриптор элемента, о котором они предоставляют дополнительную информацию.

Используя текстовый редактор, измените документ bean\_dip2. xml таким образом, чтобы у элемента recipe появился атрибут title, а элементы quantity превратились в атрибуты соответствующих элементов item. После этого ваш документ примет более профессиональный вид. Синтаксис определения атрибута для элемента следующий:

*<имя элемента*

*Имя\_атрибута= "значение ">содержание\_злемента< / имя\_элемента>*

Атрибуты всегда объявляются в открывающем дескрипторе соответствующего элемента. Вы можете указать любое количество атрибутов для одного элемента, но все они должны быть заключены в его открывающий дескриптор. Ниже приведен соответствующий пример:

*1: <имя\_элемента*

*2: имя атри6ута1="значение"*

*3: имя\_атрибута2="значение"*

*4: имя\_атрибутаЗ="значение">содержание\_элемента</имя\_элемента>*

Иногда, когда разметка становится слишком длинной, для упрощения прочтения кода полезно использовать одинаковые отступы при указании подобных элементов. Все четыре строки, в которых определены атрибуты, все равно относятся к одному элементу, несмотря на то, что они занимают четыре строки в документе.

Пример кода XML с использованием атрибутов приведен на рис.6.

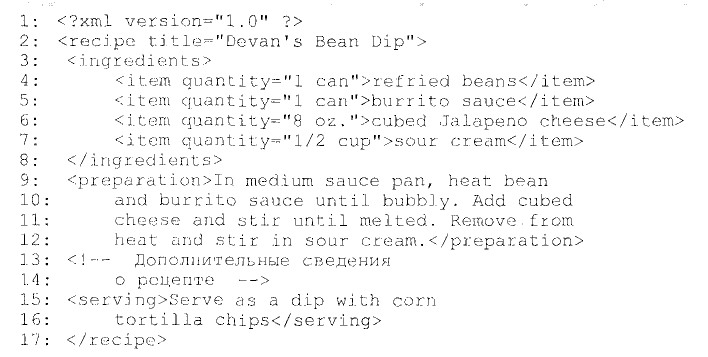


Рис.6. Документ XML с использованием атрибутов bean\_dip2. xml.

# Лабораторная работа №4. Создание и обработка XML-документов

**Цель лабораторной работы:** Изучить работу с документами в формате XML

**Задание №1:** Создать XML документ, в который включить следующую личную информацию: имя, домашний адрес, адрес электронной почты, дата рождения. При указании даты используйте атрибуты. Завершив создание документа, сохраните его и просмотрите с помощью Microsoft Internet Explorer

**Задание №2**.

необходимо создать документ XML, в котором будет содержаться рецепт приготовления одного интересного блюда. Пример рецепта рецепта приведен на рис.1. В рецепте обязательно должен быть заголовок, по которому его легко отличить от других рецептов. Список ингредиентов, а также количество каждого из них – это очень важные сведения, точно так же как инструкции по приготовлению

.

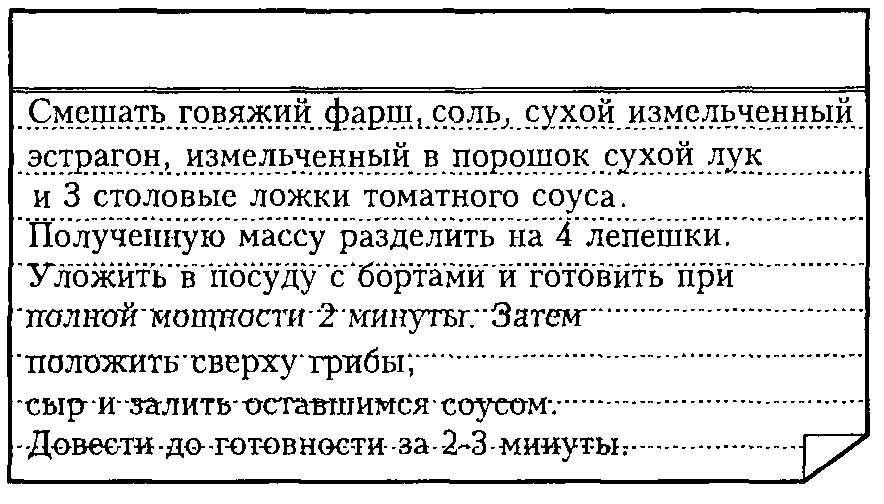
******

Рис.1. Рецепт приготовления блюда.

**Задание №3*.***

В нижеприведенном листинге приведен пример плохого оформления экземпляра XML. В этом документе перечисляются два музыкальных альбома и предоставлены некоторые сведения о них. Зная синтаксис XML, воспользуйтесь текстовым редактором для внесения необходимых исправлений. Проверьте работоспособность полученного результата, прогнав его через парсер из состава Microsoft Internet Explorer.

Пример плохого оформления экземпляра XML.

<?xml version=1.0?>

<cd number="432>

<title>The World Of Trance</title>

<artist>Van Helden</artist

<tracks total=20">

<cd number=87>

<title>Heart Breaker</title>

<subtitle>Sixteen Classic Perfomance</subtitle>

<artistPart Benatar</artist>

<tracks total=16>

</CD>

Программный код и результат работы кода включите в отчет и сдайте преподавателю.

# Контрольные вопросы

1. Для чего предназначена технология XML?
2. Назовите основное отличие языка HML и HTML.
3. С какого тега начинается документ в XML?
4. Учитывается ли в XML регистр символов?
5. Учитывается ли в XML символы форматирования (пробелы, перевод строки. табуляция), а в HTML?

**Литература**

Михеев Р.Н. VBA и программирование в MS Office для пользователей. – Спб: БХВ\_ Петербург, 2006

Шеперд Диван .Освой самостоятельно XML за 21 день, 2-е изд: Пер. с англ.:М.:Издательский дом «Вильямс», 2002

Пирогов В.Ю. MS SQL Server 2000: управление и программирование. – СПб.: БХВ – Петербург, 2005