**Лабораторная работа № 18: Работа с XML**

**Вводная часть**

Что такое XML?

XML (произносится [экс-эм-э́л]) — рекомендованный **Консорциумом Всемирной паутины** (**W3C**) язык разметки, фактически представляющий собой свод общих синтаксических правил. XML — текстовый формат, предназначенный для хранения структурированных данных (взамен существующих файлов баз данных), для обмена информацией между программами, а также для создания на его основе более специализированных языков разметки (например, **XHTML**). XML является упрощённым подмножеством языка SGML. Подробнее об XML будет рассказано ниже.

Что будет представлять собой приложение, разрабатываемое в данной работе?

Приложение будет представлять собой окно **Windows Forms**, поделённое на группы элементов, в каждой группе будут располагаться элементы управления для работы с той или иной задачей и базой данных.

**Создание приложения Windows Forms**

Запускаем Visual Studio 2010, откроется **Начальная страница**:

Для начала, надо создать проект, для этого выполним последовательно: **Файл** -> **Создать** -> **Проект**… (также можно просто нажать сочетание клавиш **Ctrl+Shift+N** или пункт «**Создать проект**…» на *Начальной странице*):

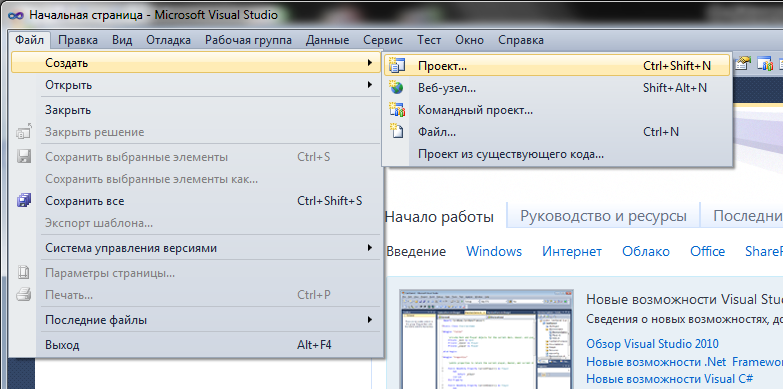


Рис. 2. 1. Создание нового проекта

Выберем слева в пункте **Установленные шаблоны** язык **Visual C#**, далее найдём в списке **Приложение Windows Forms**. Также здесь можно выбрать какой использовать «фреймворк» (набора компонентов для написания программ). В нашем случае выберем **.NET Framework 4**.

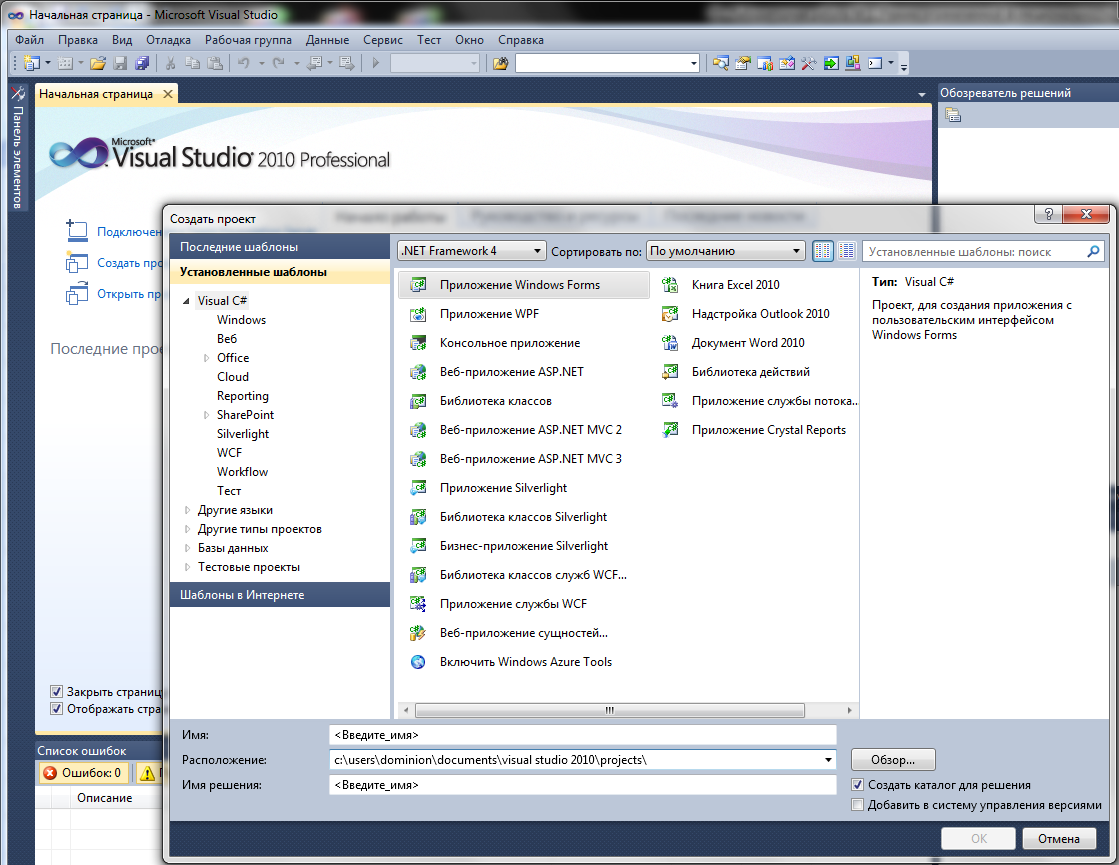


Рис. 2. 2. Окно создания нового проекта

В поле **Имя** вводим **LWP10DB** **—** это название программы (выбрано по названию лабораторного практикума, номеру и названию работы). В поле **Расположение** указана конечная директория, где будет находиться весь проект. Выберем расположение удобное для быстрого поиска. В поле **Имя решения** вводится либо название программы «по умолчанию» из поля *Имя* автоматически, либо можно ввести своё собственное. Под этим именем будет создана конечная папка проекта (если *Имя* и *Имя решения* разные).

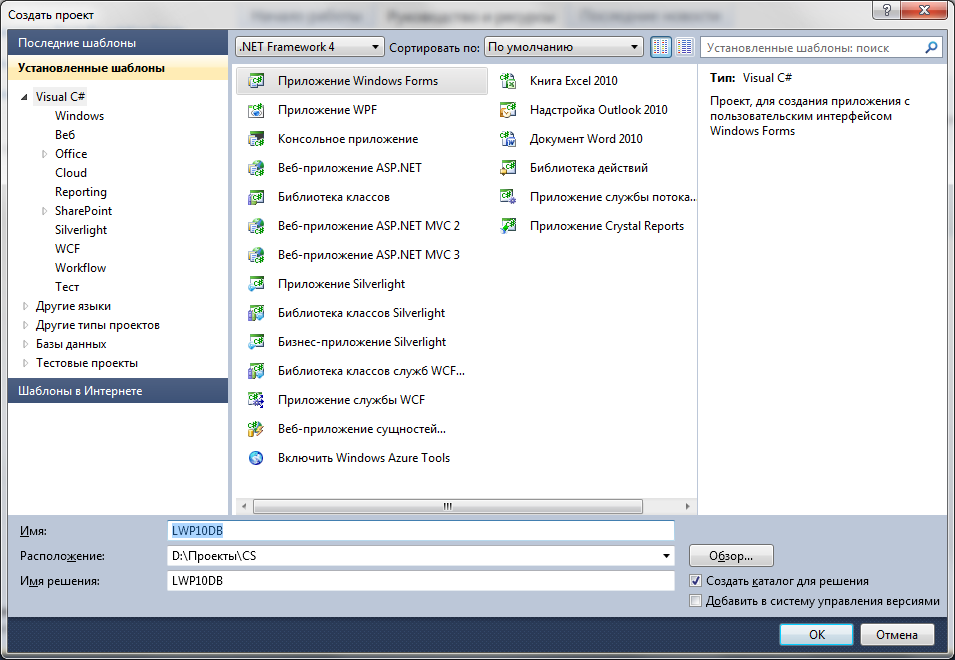


Рис. 2. 3. Вводим данные нового проекта приложения *Windows Forms*

После нажатия клавиши **ОК** мы увидим сформированный проект и исходный код приложения *Windows Forms* (не пустого изначально).

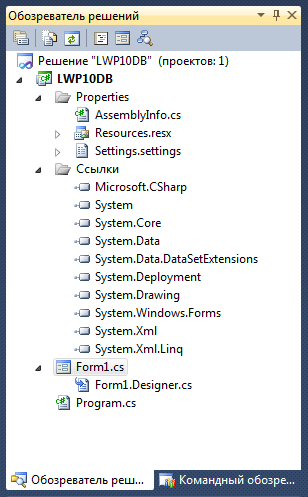


Рис. 2. 4. **Обозреватель решений**: состав проекта приложения *Windows Forms* сформированного средой разработки

Теперь, можно откомпилировать созданную программу, нажав клавишу **F5** (**Отладка** -> **Начать отладку** или нажав на иконку. Тем самым мы запускаем приложение в режиме отладки (и производим компиляцию debug-версии программы) (**Debug** выбрано изначально).

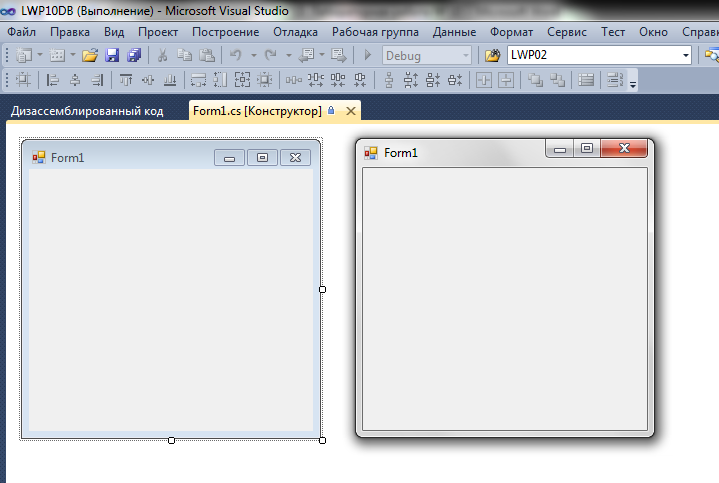


Рис. 2. 5. Запуск приложения *Windows Forms* по конфигурации *Debug*

**Модификация приложения Windows Forms: XML**

Немного общих слов об XML-формате в целом.

Язык XML (расширяемый язык разметки) предназначен для хранения структурированных данных. Данные, хранящиеся в формате XML, можно передавать между программами. Поскольку данные в XML структурированные, в некоторых случаях использование этого формата может заменить базы данных. Кроме этого, у XML есть еще много других полезных применений.

XML весьма похож на другой язык разметки **HTML**, но в *HTML* набор тегов фиксирован, и у каждого тега есть свое строго определённое назначение и правила написания. В XML можно определять собственные теги, но при этом все равно нужно соблюдать синтаксические правила языка разметки.

Типичный документ выглядит так:

<?xml version="1.0" encoding="UTF-8"?>

<recipe name="хлеб" preptime="5" cooktime="180">

<title>Простой хлеб</title>

<ingredient amount="3" unit="стакан">Мука</ingredient>

<ingredient amount="0.25" unit="грамм">Дрожжи</ingredient>

<ingredient amount="1.5" unit="стакан">Тёплая вода</ingredient>

<ingredient amount="1" unit="чайная ложка">Соль</ingredient>

<instructions>

<step>Смешать все ингредиенты и тщательно замесить.</step>

<step>Закрыть тканью и оставить на один час в тёплом помещении.</step>

<!-- <step>Почитать вчерашнюю газету.</step> - это сомнительный шаг... -->

<step>Замесить ещё раз, положить на противень и поставить в духовку.</step>

</instructions>

</recipe>

Главным является **объявление XML**:

<?xml version="1.0" encoding="UTF-8"?>

*Объявление* отвечает за кодировку документа (кодировка символов), наличие внешних зависимостей и версия документа. В версии **1.0** объявление можно не указывать. В версии **1.1** оно обязательно.

Важнейшее обязательное синтаксическое требование заключается в том, что документ имеет только один корневой элемент (англ. **root element**) (также иногда называемый элемент документа (англ. **document element**)). Это означает, что текст или другие данные всего документа должны быть расположены между единственным начальным корневым тегом и соответствующим ему конечным тегом.

Следующий простейший пример — правильно построенный документ XML:

<book>Это книга: "Книжечка"</book>

Комментарии доступны и здесь. Тэги внутри комментариев не обрабатываются:

<!-- Это комментарий -->

Остальная часть этого XML-документа состоит из вложенных элементов, некоторые из которых имеют атрибуты и содержимое. Элемент обычно состоит из открывающего и закрывающего тегов, обрамляющих текст и другие элементы. Открывающий тег состоит из имени элемента в угловых скобках, например, <step>, а закрывающий тег состоит из того же имени в угловых скобках, но перед именем ещё добавляется косая черта, например, </step>. Имена элементов, как и имена атрибутов, не могут содержать пробелы (иначе эти пробелы заменяются специальной последовательностью символов, что будет показано ниже), но могут быть на любом языке, поддерживаемом кодировкой XML-документа. Имя может начинаться с буквы, подчёркивания, двоеточия. Остальными символами имени могут быть те же символы, а также цифры, дефис, точка.

Содержимым элемента (англ. **content**) называется всё, что расположено между открывающим и закрывающим тегами, включая текст и другие (вложенные) элементы. Ниже приведён пример XML-элемента, который содержит открывающий тег, закрывающий тег и содержимое элемента:

<step>Замесить ещё раз, положить на противень и поставить в духовку.</step>

Кроме содержания у элемента могут быть атрибуты — пары имя-значение, добавляемые в открывающий тег после названия элемента. Значения атрибутов всегда заключаются в кавычки (одинарные или двойные), одно и то же имя атрибута не может встречаться дважды в одном элементе. Не рекомендуется использовать разные типы кавычек для значений атрибутов одного тега.

<ingredient amount="3" unit="стакан">Мука</ingredient>

Для данной работы вышеописанных свойств достаточно. Хотя XML обладает ещё целым рядом особенностей. Например, применение тэгов HTML, нетерпимость к перекрывающимся тэгам (один тэг, открывшийся раньше, закрывается раньше другого), наличие специальных символов HTML, пустых элементов и прочее.

Для начала добавим функциональности к уже имеющемся кода нашего приложения. Для этого в начале файла *LWP10Main.cs* добавим следующее:

using System.Xml; // XML

Найдём:

OleDbDataAdapter DataAdapter = new OleDbDataAdapter(); // DataAdapter - посредник между базой данных и DataSet

DataAdapter.SelectCommand = new OleDbCommand(SQL\_OLE, ConnectionOLE2);

DataAdapter.Fill(DataSetOLE); // Данные из адаптера поступают в DataSet

Добавим после:

DataSetOLE.DataSetName = "Главная таблица"; // Устанавливаем название корневого элемента XML-файла

DataSetOLE.WriteXml("D:\\LWP10-DB-XML.xml"); // Запишем данные таблицы в XML-файл

Скомпилируем приложение (**Debug**) и запустим. В третьем блоке элементов выберем базу данных и нажмём на *Прочитать все записи*. На диске D появится новый XML-файл: **LWP10-DB-XML.xml** с содержанием:



Рис. 5. 1. Содержание файла *LWP10-DB-XML.xml*

Как видно, пробелы в именах тэгов были заменены на символы «\_x0020\_». В остальном, документ точно повторяет структуру базы данных, из которой были выдраны данные.

Теперь создаём последнюю группу элементов. Она простая:

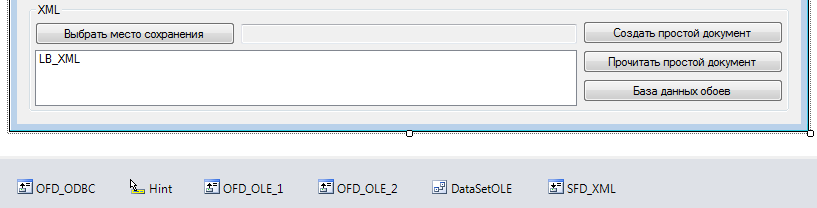


Рис. 5. 2. Расстановка элементов первой группы (**XML**)

Здесь у нас четыре кнопки, *TextBox* и большой *ListBox*. Также справа внизу элемент **SaveFileDialog**. *Свойства* элементов таковы:

*Button:*

|  |  |
| --- | --- |
| **(Name)**: | B\_XML\_Search |
| **Text**: | Выбрать место сохранения |
| **Size**: | 200; 23 |

*Button:*

|  |  |
| --- | --- |
| **(Name)**: | B\_XML\_Create |
| **Text**: | Создать простой документ |
| **Size**: | 200; 23 |

*Button:*

|  |  |
| --- | --- |
| **(Name)**: | B\_XML\_Read |
| **Text**: | Прочитать простой документ |
| **Size**: | 200; 23 |

*Button:*

|  |  |
| --- | --- |
| **(Name)**: | B\_XML\_DB |
| **Text**: | База данных обоев |
| **Size**: | 200; 23 |

*TextBox:*

|  |  |
| --- | --- |
| **(Name)**: | TB\_XML\_Path |
| **ReadOnly**: | True |

*ListBox:*

|  |  |
| --- | --- |
| **(Name)**: | LB\_XML |

*GroupBox:*

|  |  |
| --- | --- |
| **(Name)**: | GB\_XML |
| **Text**: | XML |

*SaveFileDialog:*

|  |  |
| --- | --- |
| **(Name)**: | SFD\_XML |
| **FileName**: | XML-Test |
| **InitialDirectory**: | D:\ |
| **Filter** | XML-файл|\*.xml |

Самая нижняя кнопка (**База данных обоев**) является кнопкой, открывающей новую форму, о чём пойдём немного ниже.

Событие *Click* кнопки **Выбрать место сохранения**:

private void B\_XML\_Search\_Click(object sender, EventArgs e)

{

SFD\_XML.ShowDialog();

TB\_XML\_Path.Text = SFD\_XML.FileName;

}

Событие *Click* кнопки **Создать простой документ**:

private void B\_XML\_Create\_Click(object sender, EventArgs e)

{

XmlWriterSettings SettingsXML = new XmlWriterSettings();

SettingsXML.Indent = true; // Включаем отступ для элементов XML-документа

SettingsXML.IndentChars = " "; // Задаём отступ (пробелами)

SettingsXML.NewLineChars = "\n"; // Задаём переход на новую строку

// Нужно ли опустить строку декларации формата XML документа

// Речь идёт о строке вида "<?xml version="1.0" encoding="utf-8"?>"

SettingsXML.OmitXmlDeclaration = false;

using (XmlWriter OutputXML = XmlWriter.Create(SFD\_XML.FileName, SettingsXML))

{

// Создали открывающийся тег

OutputXML.WriteStartElement("XML-Test");

// Добавляем атрибут для XML-Test

OutputXML.WriteAttributeString("Count\_Parameters", "4");

// Создаем элемент <имя>элемент</имя>

OutputXML.WriteElementString("Question", "Answer");

Random R = new Random();

OutputXML.WriteElementString("A", R.Next(0,1000).ToString());

OutputXML.WriteElementString("B", SQL\_OLE);

OutputXML.WriteElementString("C", TB\_XML\_Path.Text);

OutputXML.WriteStartElement("Names");

OutputXML.WriteStartElement("Name");

OutputXML.WriteAttributeString("Type", "Male");

OutputXML.WriteString("John");

OutputXML.WriteEndElement();

OutputXML.WriteStartElement("Name");

OutputXML.WriteAttributeString("Type", "Male");

OutputXML.WriteString("Teo");

OutputXML.WriteEndElement();

OutputXML.WriteStartElement("Name");

OutputXML.WriteAttributeString("Type", "Famale");

OutputXML.WriteString("Miana");

OutputXML.WriteEndElement();

// Закрываем XML-Test

OutputXML.WriteEndElement();

// Сбрасываем буфферизированные данные

OutputXML.Flush();

// Закрываем фаил, с которым связан output

OutputXML.Close();

MessageBox.Show("Документ " + SFD\_XML.FileName + " успешно создан!", "Работа с базами данных (C#) :: XML");

}

}

Событие *Click* кнопки **Прочитать простой документ**:

private void B\_XML\_Read\_Click(object sender, EventArgs e)

{

S = null;

LB\_XML.Items.Clear();

// Создаём экземпляр класса

XmlDocument InputXML = new XmlDocument();

// Загружаем XML-документ из файла

InputXML.Load(SFD\_XML.FileName);

// Загружаем XML-документ из строки

// InputXML.LoadXML(Path);

// Получаем всех детей корневого элемента

// InputXML.DocumentElement - корневой элемент

foreach (XmlNode Table in InputXML.DocumentElement.ChildNodes)

{

// Перебираем все атрибуты элемента

foreach (XmlAttribute A in Table.Attributes)

{

// A.Name - имя текущего атрибута

// A.Value - значение текущего атрибута

S = A.Name + ": " + A.Value;

}

// Перебираем всех детей текущего узла

foreach (XmlNode CN in Table.ChildNodes)

{

}

// Получаем текст хранящийся в текущем узле

S = S + Table.InnerText + "\n\t";

}

MessageBox.Show("Значения аттрибутов для элеметов и узлов файла "+ SFD\_XML.FileName +":\n\n\t" + S + "\nЗначения получены автоматическим перебором!", "Работа с базами данных (C#) :: XML");

// Вытаскиваем значения "руками"

XmlNodeList XMLTestAttributes = InputXML.SelectNodes("/XML-Test[@Count\_Parameters='4']");

foreach (XmlNode XN in XMLTestAttributes)

{

LB\_XML.Items.Add(XN.InnerText + " -- все значения"); // В ListBox получаем значения всех узлов

}

XmlNodeList Names = InputXML.SelectNodes("/XML-Test/Names/Name[@Type='Male']");

foreach (XmlNode XN in Names)

{

LB\_XML.Items.Add(XN.InnerText + " -- одно мужское имя"); // В ListBox получаем только два мужских имени по отдельности

}

XmlNode Question = InputXML.DocumentElement.SelectSingleNode("Question");

LB\_XML.Items.Add(Question.InnerXml);

XmlNode ANode = InputXML.DocumentElement.SelectSingleNode("A");

LB\_XML.Items.Add(ANode.InnerText);

XmlNode BNode = InputXML.DocumentElement.SelectSingleNode("B");

LB\_XML.Items.Add(BNode.InnerText);

XmlNode CNode = InputXML.DocumentElement.SelectSingleNode("C");

LB\_XML.Items.Add(CNode.InnerText);

XmlNode NamesNode = InputXML.DocumentElement.SelectSingleNode("Names");

LB\_XML.Items.Add(NamesNode.InnerText + " -- все имена"); // В Listbox получаем все имена

}

Компилируем приложение (**Debug**) и запускаем:

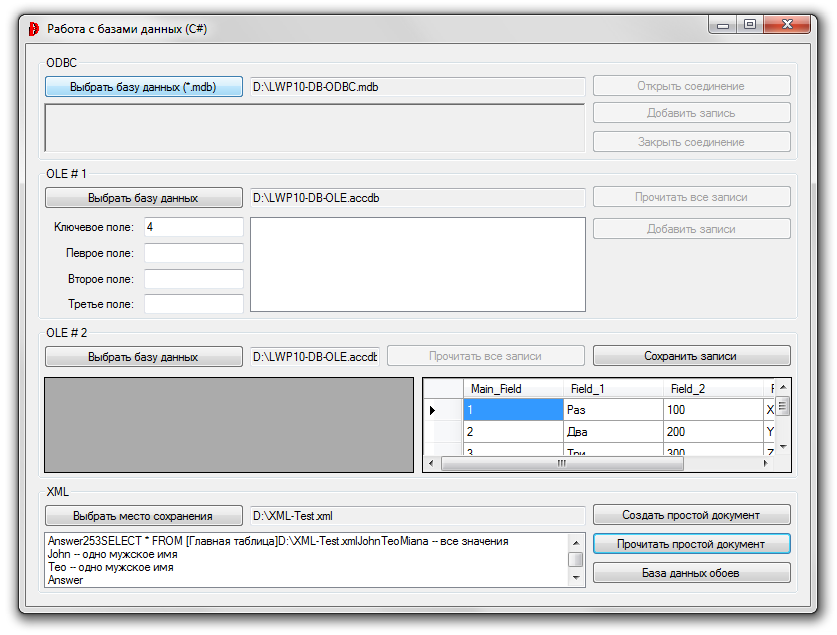
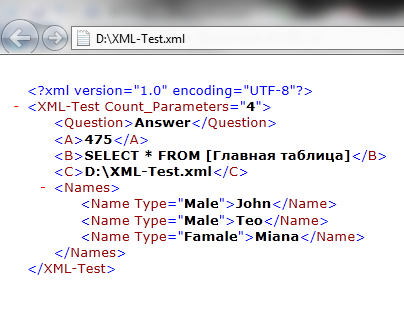


Рис. 5. 3. Окончательная работа блока: **XML**

Нажатие на кнопку *Создать простой документ*, создаёт по указанному пути файл (в данном случае по рисунку: **D:\XML-Test.xml**) следующего содержания:



При нажатии на кнопку *Прочитать простой документ*, происходит чтение всех записей в цикле (с выводом всплывающего окна), а затем вывод в ListBox. Выводятся все значения (верхняя строчка), затем только два мужских имени из блока **<Names>...</Names>**, затем все значения узлов **Question**, **A**, **B** и С и наконец все имена.

**Почему существует несколько классов для работы с одним и тем же (в частности в XML)?** На этот вопрос, есть два ответа: «на вкус и цвет все фломастеры разные» и «в зависимости от предоставляемой функциональности один класс может оказаться удобнее в той или иной ситуации».

Несколько причин, почему стоит использовать классы для генерации XML документа, а не создавать его вручную:

1. Практически нулевой риск создать неправильный XML документ.

2. Можно задать форматирование (отступы и прочее) XML документа (очень сильно улучшает читаемость).

Наиболее подходящим для формирования XML-документа «за раз» является класс **XmlWriter**. Плюсы данного класса:

1. Форматирование результирующего XML документа.

2. Создание открывающего и закрывающего тега в одну команду, причём для создания закрывающего тега не требуется знать имя открывающего.

Для форматирования выводимого XML документа в *XmlWriter* используется класс **XmlWriterSettings**.

Используя функцию **WriteElementString**,тэг не только открывается, в него записываются данные, но и тут же закрывается, поэтому добавить атрибут к такому элементу НЕЛЬЗЯ! Для решения данной проблемы нужно использовать **WriteStartElement**, **WriteAttributeString**, **WriteEndElement**.

Для **чтения XML-документа** предусмотрен класс **XMLReader**, но он крайне не удобен, потому что осуществляет последовательное считывание и не позволяет «прыгать через»/обращаться сразу к нужному элементу.

Чтение и редактирование удобно производить через класс **XmlDocument**.

Из чего состоит XML-документ:

1. Строка вида <?xml … ?>

2. Родительский элемент (единственный).

3. Дальше, внутри родительского элемента располагаются элементы-потомки (child’ы) и далее по иерархии.

**Теперь давайте рассмотрим как изменять уже имеющийся XML-документ?**

Для вставки элемента в иерархию XML-документа необходимо создать элемент типа **XmlNode** и задать его родителя, делается это следующим образом:

// Создаём node

// book - имя узла

XmlNode node = xmlDoc.CreateElement("book");

// Добавляем его в качестве ребенка

parentNode.AppendChild(node);

**Удаление узла?**

// Удаление узла

parentNode.RemoveChild(node);

**Работа с атрибутами**:

// Создаём новый атрибут

// genre - имя атрибута

XmlAttribute newAttr = doc.CreateAttribute("genre");

newAttr.Value = "novel"; // Задаём его значение

// Добавляем атрибут в коллекцию атрибутов элемента

node.SetNamedItem(newAttr);

// Удаляем атрибут

node.Attributes.RemoveNamedItem("genre");

// Изменяем значение атрибута

XmlAttributeCollection Attribs = node.Attributes;

XmlAttribute attr = (XmlAttribute)Attribs.GetNamedItem("genre");

attr.Value = "fiction";

Настало время разработать более-менее полезную функциональность. Используем возможности XML и сделаем на основе него полноценную базу данных.

Будем использовать загрузку данных из XML файла в *DataSet*, отметив, что после загрузки данных, доступ к данным ничем не отличается от работы с данными при их загрузке из таблиц БД.

Для XML справедливо: «Содержимым элемента (*content*) называется всё, что расположено между открывающим и закрывающим тегами. Это текст, вложенные элементы, комментарии и т.п.». Все спецификации XML документов подчёркивают, что именно текст, то, что является содержимым документа. Все средства для работы с XML документами, в том числе и методы класса *XmlDocument*, также ориентированы на текст (**CreateTextNode** и т.п.).

Из сказанного можно сделать вывод: хранить двоичные данные можно, только если они будут представлены как текстовые строки. Однако, прямое преобразование всего массива байт (двоичных данных) в текстовую строку не будет решением в силу того, что возможно присутствие в двоичном коде значений, совпадающих со служебными символами кодировки строк и значений, равных завершению строки. Как результат - мы сможем поместить в контент тэга только часть строки (до первого совпадения двоичного символа со значением окончания строки) при возможном отображении помещенного кода на нескольких строках XML файла для одного содержимого контента. Следовательно, строка должна формироваться и присваиваться значению контента как единое целое. Способы выполнить это могут быть разные, от посимвольного преобразования байт в значение char и последовательного добавления результата в содержимое контента (медленный способ), до использования **StringBuilder** (быстрый способ).

Наиболее часто хранение двоичных данных связано с задачей хранения рисунков (хотя приведенный ниже код будет равно хорошо работать и при использовании его для хранения в виде двоичных данных Web-страниц, Word или Excel-документов и прочее). Содержимое файла рисунка: массив **шестнадцатеричных символов**. Как следствие, и код представления рисунка в **content** опирается на шестнадцатеричное отображение строк.

**Так что же это будет за функциональность**? Приложение будет по указанному через диалог изображению создавать запись в базе данных (XML-файл), куда будет помещаться как сведения об изображении, так и само изображение в виде длинной строки символов. Также по номеру (ID рисунка) из базы можно будет восстановить это изображение и просмотреть его. Изображения можно будет только добавлять в базу, и просматривать добавленное уже из базы. Для этой функциональности у нас предусмотрена кнопка **База данных обоев**. А также заранее заготовленный файл **Wallpapper-DB.xml** с содержанием:

<?xml version="1.0" encoding="windows-1251"?>

<NewDataSet>

<wallpapper>

<id></id>

<name></name>

<pichash></pichash>

<picext></picext>

<size></size>

</wallpapper>

</NewDataSet>

**ПРИМЕЧАНИЕ**: Точно таким образом можно сохранять и восстанавливать любые данные (Web-страницы, Word или Excel-документы и прочее). Подобный способ хранения может быть полезен, для решения многих задач связанных с хранением и передачей информации, в том числе и конфиденциальной (например, после преобразования файла всю строку можно закодировать паролем через MD5-хэш).

Если по каким-то причинам русские символы не читаются, сделаем следующее:

Найдём:

using (StreamReader SR = new StreamReader(String\_Path + @"\Wallpapper-DB.xml", System.Text.Encoding.UTF8))

И заменим:

using (StreamReader SR = new StreamReader(String\_Path + @"\Wallpapper-DB.xml", System.Text.Encoding.Default))

Найдём:

// Сохраняем данные

WallpapperDataSet.WriteXml(String\_Path + @"\Wallpapper-DB.xml", XmlWriteMode.WriteSchema);

WallpapperDataSet = new DataSet();

// Вновь загружаем сохраненные данные

WallpapperDataSet.ReadXml(String\_Path + @"\Wallpapper-DB.xml", XmlReadMode.Auto);

WallpapperDataTable = WallpapperDataSet.Tables[0];

Заменим:

XML = new XmlDocument();

XML.InnerXml = WallpapperDataSet.GetXml();

XmlDeclaration XMLDeclaration = XML.CreateXmlDeclaration("1.0", "windows-1251", "yes");

XML.InsertBefore(XMLDeclaration, XML.DocumentElement);

XML.Save(String\_Path + @"\Wallpapper-DB.xml");

WallpapperDataSet = new DataSet();

WallpapperDataSet.ReadXml(String\_Path + @"\Wallpapper-DB.xml", XmlReadMode.Auto);

WallpapperDataTable = WallpapperDataSet.Tables[0];

**Варианты предметных областей создаваемых XML-документов:**

|  |  |
| --- | --- |
| вариант 1: | библиографическое описание списка литературы |
| вариант 2: | описание фильмов видеотеки |
| вариант 3: | список сотрудников организации |
| вариант 4: | список моделей мобильных телефонов |
| вариант 5: | список студентов факультета |
| вариант 6: | список изучаемых дисциплин |
| вариант 7: | список музыкальных произведений |
| вариант 8: | список моделей автомобилей |
| вариант 9: | список моделей компьютеров |
| вариант 10: | список аудиофайлов |
| вариант 11: | список медицинских услуг |
| вариант 12: | список арендных площадей |
| вариант 13: | список программного обеспечения |
| вариант 14: | список ассортимента обуви |
| вариант 15: | список моделей самолетов |
| вариант 16: | список туров туристической компании |