МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

В. В. Воронина

ТЕХНОЛОГИИ АВТОМАТИЗАЦИИ БИЗНЕС-ПРОЦЕССОВ ПРЕДПРИЯТИЙ

Учебное пособие

Ульяновск
УлГТУ
2013

УДК 004.42
ББК 32.973-018.1
В 12

Рецензенты:
профессор кафедры «Информационные технологии» УлГУ, д.т.н. И. В. Семушин;
доцент кафедры «Вычислительная техника» УлГТУ, к.т.н. А. Г. Игонин.

Утверждено редакционно-издательским советом университета в качестве учебного пособия

Воронина, В. В.
В 12 Технологии автоматизации бизнес-процессов предприятий : учебное пособие / В. В. Воронина. – Ульяновск : УлГТУ, 2013. – 204 с.

Представлены базовые технологии автоматизации бизнес-процессов предприятий.
Пособие предназначено для студентов направления 230700.62 «Прикладная информатика», профиль «Прикладная информатика в экономике», и студентов направления 231000.62 «Программная инженерия», изучающих дисциплину «Объектно-ориентированное программирование», а также для студентов других направлений, изучающих дисциплины, связанные с автоматизацией бизнес-процессов.

УДК 004.42
ББК 32.973.2-018.1

Учебное электронное издание
ВОРОНИНА Валерия Вадимовна
ТЕХНОЛОГИИ АВТОМАТИЗАЦИИ БИЗНЕС-ПРОЦЕССОВ ПРЕДПРИЯТИЙ
Учебное пособие
Редактор А. В. Ганина

Объем данных 1,25 Мб. ЭИ № 224. Заказ 152.
Ульяновский государственный технический университет, ИПК «Венец»
432027, г. Ульяновск, ул. Сев. Венец, д. 32.
Тел.: (8422) 778-113.
E-mail: venec@ulstu.ru
http://www.venec.ulstu.ru

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 4
1. ХРАНЕНИЕ ДАННЫХ ПРИЛОЖЕНИЯ 8
1.1. Работа с XML-сериализатором 9
1.2. Работа с базами данных 14
Контрольные вопросы к разделу 1 35
2. ПОСТРОЕНИЕ ОТЧЕТОВ 36
2.1. Работа с OpenOffice 37
2.2. Работа с MS Excel 53
2.3. Работа с MS Word 71
2.4. Работа с форматом PDF 83
2.5. Многопоточность 93
Контрольные вопросы к разделу 2 108
3. РАБОТА ПРИЛОЖЕНИЙ В СЕТЕВОМ И ЛОКАЛЬНОМ ОКРУЖЕНИЯХ 109
3.1. Организация клиент-серверного взаимодействия 109
3.2. Работа с электронной почтой 121
3.3. Работа приложения в сети Интернет 130
3.4. Работа с API Google.Маршруты 135
3.5. Работа с системой «Клиент-Банк» 146
3.6. Работа с ОС и аппаратной частью 149
Контрольные вопросы к разделу 3 164
4. ПРИМЕРЫ ЗАДАЧ АВТОМАТИЗАЦИИ И ПРОЕКТИРОВАНИЯ ПРИЛОЖЕНИЯ 165
4.1. Примеры задач автоматизации 165
4.2. Начальное проектирование приложений 187
Контрольные вопросы к разделу 4 198
ЗАКЛЮЧЕНИЕ 199
ГЛОССАРИЙ 200
ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ 202
БИБЛИОГРАФИЧЕСКИЙ СПИСОК 203

ВВЕДЕНИЕ

Когда мы говорим о решении задач автоматизации бизнес-процессов предприятий, то в качестве инструмента, в первую очередь, рассматриваем платформу и конфигурации 1С. Действительно, средствами этой технологии можно автоматизировать практически все. Но в некоторых случаях приходится искать альтернативные варианты. К примеру, серверы 1С из соображений безопасности часто размещают изолированно от глобальной сети, что ограничивает ее возможности. К тому же руководители предприятий не склонны покупать дополнительное количество ключей, поэтому в специфических задачах массового использования программного обеспечения решение, выполненное средствами другой технологии, будет ощутимо дешевле.

Задачи, возникающие на предприятиях, как правило, весьма стандартны. Вернее, можно выделить группу подзадач, комбинации которых можно найти в автоматизации практически любого бизнес-процесса. Приведем некоторые из них:

- работа с базами данных или альтернативными хранилищами информации;
- построение отчетов;
- массовая рассылка электронных писем;
- массовая печать пакетов регламентных документов для клиентов;
- разбор электронной почты;
- работа с платежными системами;
- реализация клиент-серверного взаимодействия;
- работа приложения с данными из сети Интернет.

Так как использование возможностей платформы 1С не всегда является подходящим способом решения поставленных задач, то в качестве альтернативы разумно рассматривать возможности

разработки программных решений на платформе .Net. Во-первых, потому, что .Net Framework поддерживает создание и выполнение нового поколения приложений и веб-служб. Она обеспечивает согласованную объектно-ориентированную среду программирования, минимизирующую конфликты при развертывании программного обеспечения и управлении версиями; гарантирует безопасное выполнение кода, включая код, созданный неизвестным или не полностью доверенным сторонним изготовителем; обеспечивает единые принципы работы разработчиков для разных типов приложений, таких как приложения Windows и веб-приложения, а также реализует взаимодействие на основе промышленных стандартов, которое обеспечит интеграцию кода .NET Framework с любым другим кодом. Во-вторых, в настоящее время доступно большое количество бесплатных многофункциональных библиотек, совместимых с данной платформой.

Цель данного пособия – изложить в сжатой и доступной форме азы наиболее часто используемых технологий автоматизации бизнес-процессов предприятий. В качестве основы взяты материалы официальных руководств для разработчиков, обучающих статей, специализированной литературы, тем обсуждений на компьютерных форумах, а также разработки самого автора. В ходе изучения материала статей, руководств, книг и форумов, представленных в библиографическом списке, были выявлены следующие недостатки изложенного в них материала. Во-первых, не вся информация доступна на русском языке. Во-вторых, большинство материала рассчитано на уже состоявшихся разработчиков и не всегда содержит необходимые пояснения. В-третьих, как правило, материал разрознен и требует больших временных затрат для изучения.

В данном учебном пособии рассматриваются технологии, совместимые с платформой .Net, позволяющие строить эффективные

многофункциональные приложения на языке C#, решая приведенные выше задачи. Используемая среда разработки – Visual Studio. Пособие предназначено для студентов направления 230700.62 «Прикладная информатика», профиль «Прикладная информатика в экономике», студентов направления 231000.62 «Программная инженерия», изучающих дисциплину «Объектно-ориентированное программирование», а также студентов других направлений, интересующихся современными технологиями автоматизации бизнес-процессов.

Выписка из ФГОС ВПО:

Цикл, к которому относится дисциплина	Компетенции студента, формируемые в результате освоения дисциплины
Б3.В.2 (для «Программной инженерии») и Б2.В.5 (для «Прикладной информатики»)	Для «Программной инженерии»: • стремление к саморазвитию, повышению своей квалификации и мастерства (ОК-6); • умение применять основы информатики и программирования к проектированию, конструированию и тестированию программных продуктов (ПК-10); • навыки использования операционных систем, сетевых технологий, средств разработки программного интерфейса, применения языков и методов формальных спецификаций, систем управления базами данных (ПК-15). Для «Прикладной информатики»: • способность самостоятельно приобретать и использовать в практической

Цикл, к которому относится дисциплина	Компетенции студента, формируемые в результате освоения дисциплины
	деятельности новые знания и умения, стремиться к саморазвитию (ОК-5); • способность ставить и решать прикладные задачи с использованием современных информационно-коммуникационных технологий (ПК-4); • способность применять к решению прикладных задач базовые алгоритмы обработки информации, выполнять оценку сложности алгоритмов, программировать и тестировать программы (ПК-10). В результате изучения дисциплины студент должен: • знать базовые принципы объектно-ориентированного программирования, основные приемы объектного анализа предметной области, разницу между структурным и объектно-ориентированным программированием; • уметь применять методы объектно-ориентированного анализа предметной области и реализовывать принципы объектно-ориентированного программирования; • владеть навыками объектно-ориентированной разработки программ.

Для формирования описанных выше компетенций во втором семестре изучения дисциплины «Объектно-ориентированное программирование» студенты должны, во-первых, получить теоретические знания, познакомившись с такими технологиями, как ADO.NET и LINQ, для работы с БД; ITextSharp для работы с pdf-документами; службы платформенного вызова (InteropServices) для прямого доступа к оборудованию и некоторым дополнительным возможностям операционной системы; технология отражения (Reflection) для построения отчетов в OpenOffice или Microsoft Office и прочее. Во-вторых, получить практические навыки объектно-ориентированного проектирования и разработки системы автоматизации некоего бизнес-процесса в рамках курсовой работы. Цель данного учебного пособия – помочь студентам в этом.

Материал сгруппирован в четыре раздела. Первый посвящен технологиям хранения и обработки информации. Во втором рассматриваются способы построения различных отчетов, а также технология повышения эффективности их построения. Третий раздел посвящен организации работы приложения в сетевом и локальном окружениях. В четвертом рассматриваются базовые приемы начального объектно-ориентированного проектирования приложения, а также приведены примеры задач автоматизации, которые могут быть использованы как задачи на курсовые работы.

1. ХРАНЕНИЕ ДАННЫХ ПРИЛОЖЕНИЯ

Существует два способа долговременного хранения данных, с которыми работает приложение. Первый – использование обычных файлов, второй – использование баз данных. Первый вариант, как правило, менее распространен в силу ограниченности возможностей, но удобен в том случае, если разрабатывается локальное приложение, объем хранимых данных которого не очень большой, и манипуляции

с ними ограничиваются редактированием, добавлением, удалением и несложными выборками. Тогда затраты на установку и настройку СУБД становятся нецелесообразными. Второй способ, соответственно, подходит для всех остальных случаев. Рассмотрим оба варианта подробнее.

1.1. Работа с XML-сериализатором

Пусть необходимо разработать приложение для проведения аттестации сотрудников фирмы. Функции программы: загрузка файлов с материалом по теме для прохождения обучения, загрузка или создание файлов с тестами и установление их связи с разделами материала, проведение тестирования и хранение списка тестируемых с оценками. Для подобного приложения идеально подойдет первый способ хранения данных, реализованный через механизм XML-сериализации. В данном случае под сериализацией будем понимать процесс перевода какого-либо объекта в двоичный файл на диске, например, в XML-файл. Восстановление объекта из файла называется десериализацией.

По сути XML представляет собой текстовый формат, предназначенный для хранения структурированных данных. Выгода от его использования обуславливается следующим:

- в этом формате легко могут быть описаны такие структуры данных, как записи, списки и деревья;
- формат представляет собой простой текст, свободный от лицензирования и каких-либо ограничений, кроме того, он не зависит от платформы;
- в C# для работы с этим форматом существуют очень удобные инструменты.

Любой XML-файл содержит в себе описанную в текстовом формате иерархическую структуру данных, визуально представленную как дерево элементов, каждый из которых

описывается тэгами (командами языка разметки, заключенными в угловые скобки). Основное преимущество языка XML в данном случае – возможность, а вернее необходимость, самостоятельно определять тэги, так как стандартного набора этих команд в языке нет. Любой XML-документ требует наличия так называемого корневого элемента – тела документа, в которое будут входить все прочие данные. Опишем в качестве примера список тестируемых для нашего приложения. Первая строка документа стандартная. Она называется объявление XML и указывает версию стандарта, кодировку символов и внешние зависимости (при необходимости). Далее идет корневой элемент и прочие элементы, расположенные ниже по иерархии. О каждом тестируемом нам необходимо хранить следующую информацию: табельный номер, ФИО и список пройденных тестов с оценками. Тогда вид XML-документа, хранящего эти данные, может быть следующим:

```
<?xml version="1.0" encoding="UTF-8"?>
<ListTestiruemih>
        <Testiruemii>
         <tnum>129</tnum>
         <F>Иванов</F>
         <I>Иван</I>
         <O>Иванович</O>
         <ListTest>
           <Test>
            <Kod>Раздел1Тест1</Kod>
            <Score>50</Score>
             <MaxScore>100</MaxScore>
            </Test>
         </ListTest>
        </Testiruemii>
        <Testiruemii>
           <tnum>130</tnum>
           <F>Иванов</F>
           <I>Петр</I>
           <O>Иванович</O>
           <ListTest>
            <Test>
```

```
            <Kod>Раздел2Тест1</Kod>
            <Score>30</Score>
            <MaxScore>100</MaxScore>
          </Test>
          <Test>
            <Kod>Раздел1Тест1</Kod>
            <Score>60</Score>
            <MaxScore>100</MaxScore>
          </Test>
        </ListTest>
      </Testiruemii>
</ListTestiruemih>
```

В данном случае корневой элемент описывается тэгом ListTestiruemih. В него входит три элемента, отмеченных тэгами Testiruemii. Каждый из них содержит элементы tnum (табельный номер), F (фамилия), I (имя), O (отчество), ListTest (список пройденных тестов), содержащий элементы Test (тесты). Последние, в свою очередь, содержат элементы Kod, Score и MaxScore.

Рассмотрим пример программного кода, позволяющий получить подобный файл. Для реализации этого мы используем возможности класса XMLSerializer. Выполнение функций сериализации и десериализации этот класс осуществляет методами Serialize и Deserialize соответственно. Первый метод является процедурным, принимающим в качестве параметров указатель на файловый поток для записи и объект, который необходимо сериализовать. Второй метод является функциональным. Он принимает указатель на файловый поток, откуда осуществляется чтение ранее сериализованного объекта, а в результате своей работы возвращает объект, если чтение прошло корректно, либо пустую ссылку – в противном случае.

Первое, что нам необходимо запомнить, это то, что способ работает только с открытыми типами и открытыми членами этих типов. Второе – сериализуем и десериализуем мы всегда объект. Поэтому сначала необходимо выделить классы-сущности, которые

будут использоваться в нашем приложении. По сути их два – тест и тестируемый, но так как необходимо хранить еще и сам список тестируемых, то понадобится дополнительный класс. Третье, что необходимо помнить, – сериализуемый объект должен обязательно иметь конструктор без параметров, даже если в программном коде он не будет использоваться. Итак, начинаем проектировать классы:

```
public class Test
{
public int Kod;
public int Score;
public int MaxScore;
public Test(){}
}
public class Testiruemii
{
public int tnum;
public string F;
public string I;
public string O;
public List<Test> Test;
public Testiruemii(){}
}

public class ListTestiruemih
{
public List<Testiruemii> list;
public ListTestiruemih(){}
}
```

Далее в основной программе мы создаем объект третьего класса:

```
public ListTestiruemih objForSer=new ListTestiruemih();
```

Затем каким-либо способом заполняем значениями его поля и поля вложенных в него объектов. Таким образом, далее мы предполагаем, что у нас есть заполненный список тестируемых с пройденными тестами, хранимый в объекте objForSer.

Теперь приступим к реализации самой сериализации. Для начала нам необходимо подключить нужные пространства имен:

```
using System.Xml;
using System.Xml.Serialization;
using System.IO;
```

Затем создать объект класса XmlSerializer, настроенного на наш класс:

```
XmlSerializer serializer = new XmlSerializer(typeof(ListTestiruemih));
```

Далее откроем файловый поток для записи:

```
FileStream f = new FileStream(@"C:\path\2.xml", FileMode.OpenOrCreate);
```

И выполним сериализацию в файл:

```
using (StreamWriter sw = new StreamWriter (f))
{
serializer.Serialize(sw, objForSer);
}
```

Для выполнения десериализации нам необходимо открыть файловый поток для чтения:

```
FileStream f2 = new FileStream(@"C:\path\2.xml", FileMode.Open);
```

И считать объект из файла:

```
using (StreamReader sr = new StreamReader(f2))
{
objForSer = serializer.Deserialize(sr) as ListTestiruemih;
}
```

Завершающее действие заключается в проверке корректности десериализации и выделении памяти под объект в противном случае:

```
if (objForSer==null)objForSer=new ListTestiruemih();
```

1.2. Работа с базами данных

Наиболее часто для хранения данных приложения используют базу данных, работающую под управлением какого-либо сервера (СУБД). Фактически база данных (БД) представляет собой лишь файл с информацией, а все запросы к нему обрабатывает именно сервер. Поэтому прежде чем приступить к написанию приложений, работающих с базами данных, необходимо установить какой-либо сервер, например MS SQL Express Edition. При работе с СУБД следует обратить внимание на некоторые вещи.

Во-первых, сервер на машине должен быть не только установлен, но и запущен. Во-вторых, следует избегать установки и одновременного запуска нескольких версий одного и того же сервера.

Фактически работа приложения строится следующим образом: выполняется соединение с сервером, отсылается запрос на обработку данных и принимается результат выполнения этого запроса. Для соединения с сервером используются специальные драйверы или коннекторы, которые различаются в зависимости от сервера. Параметры подключения приложения к базе данных описываются специальным свойством, называемым строкой подключения (connectionString). Рассмотрим пример создания базы данных на сервере MS SQL и установки соединения с ней из приложения Visual Studio 2010.

Для создания базы данных выполните следующие действия. Откройте любой проект и перейдите в окно Server Explorer (через пункт меню View). Найдите Data Connection и, нажав правую кнопку мыши на этом пункте, выберите Add Connection (рисунок 1.1.).

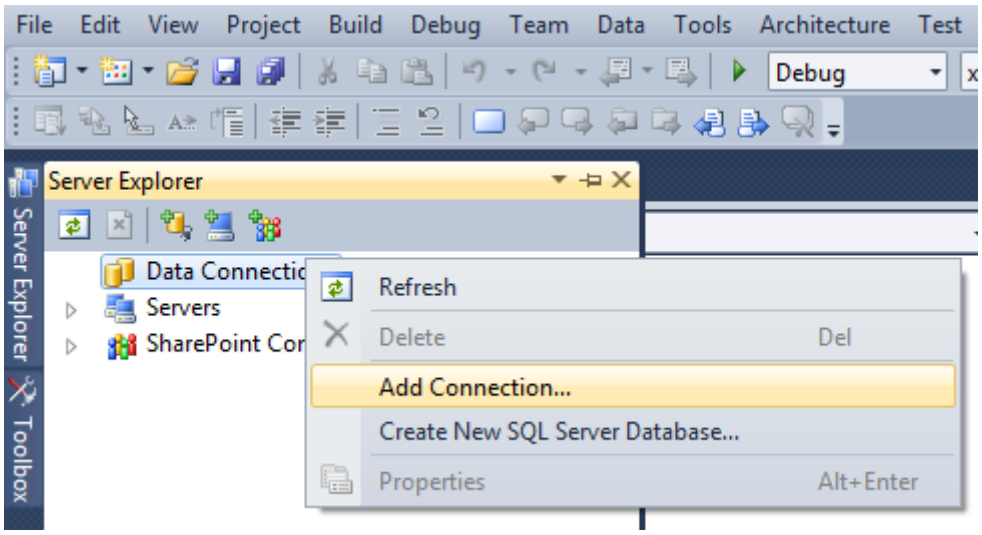

Рисунок 1.1. Создание базы данных

В открывшемся окне в разделе Data Source нажмите Change и выберите Microsoft Sql Server Database File. В поле Database file name введите полный путь к месту, где необходимо создать файл БД. Например, чтобы на диске D создать файл с именем MyDB.mdf, пишем: D:\MyDB.mdf и нажимаем ОК (рисунок 1.2.).

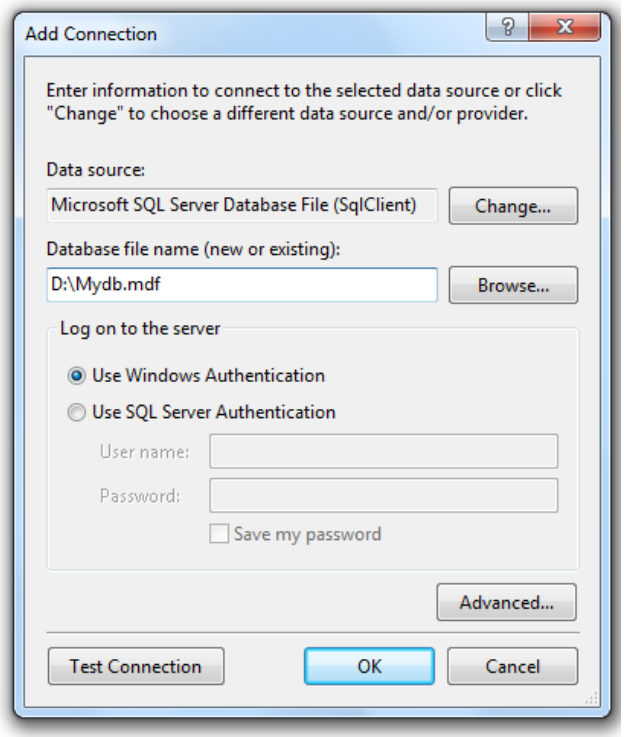

Рисунок 1.2. Создание БД

Так как данного файла на диске нет, то программа выдаст об этом предупреждение и предложит создать этот файл, с чем необходимо согласиться. Теперь в окне Server Explorer под Data Connection появился новый элемент – имя вашего файла. Правой кнопкой нажимаем на нем и переходим в его свойства (Properties), где ищем ConnectionString. Эта строка будет иметь примерно следующий вид:

Data Source=.\SQLEXPRESS;AttachDbFilename=D:\MyDB.mdf; Integrated Security=True;Connect Timeout=30;User Instance=True

Далее везде под строкой соединения будет подразумеваться эта строка.

Создать таблицы БД в редакторе Server Explorer можно следующим образом. Открываем БД, нажав на значок-стрелку рядом с ней, и ищем папку Tables. Нажимаем на этой папке правой кнопкой мыши и выбираем Add New Table (рисунок 1.3.).

В результате этих действий откроется окно-редактор столбцов. ColumnName – имя столбца, DataType – тип данных столбца. Заполняем эти данные. Для создания автоинкрементных столбцов делаем следующее. Переходим к окну свойств столбца (в нижней части экрана) и ищем свойство Identity Specification. Ставим Is Identity – Yes. Остальные свойства в категории Identity по 1. Для сохранения таблицы нажмите или кнопку «Сохранить все», или кнопку закрытия вкладки редактирования столбцов и в предложенном окне введите имя таблицы. После того как вы сформировали все таблицы, вы можете перейти к работе с ними из приложения.

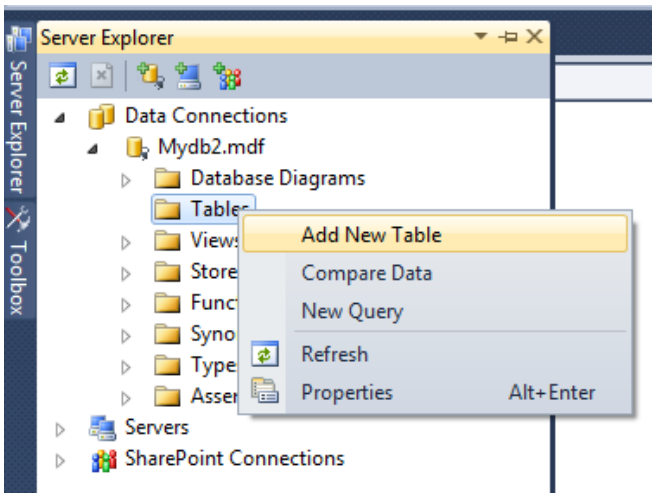

Рисунок 1.3. Добавление таблиц в БД

В данном разделе мы рассмотрим три метода работы с базой данных: использование технологии ADO.Net с визуальными компонентами, использование технологии LINQ и смешанный вариант.

Технология ADO.Net – это набор классов, предоставляющих доступ к реляционным данным. Главными достоинствами этой технологии являются, во-первых, то, что она позволяет работать с разными СУБД, обеспечивая легкость перехода с одной на другую всего лишь изменением поставщика данных. Во-вторых, то, что она позволяет снизить нагрузку на СУБД и повысить производительность приложений. Основной объект ADO.Net, представляющий множество данных, – DataSet. В нем могут храниться данные из одной таблицы, множества таблиц, а также данные-результаты запросов. Для связи данных с визуальными компонентами (например, с DataGridView) используются объекты bindingSource и tableAdapter. Эти объекты существенно уменьшают программный код приложения, но требуют достаточно сложной и внимательной настройки, а также имеют весьма ограниченные возможности управления.

При работе с визуальными компонентами через объекты bindingSource и tableAdapter необходимо учесть две вещи. Во-первых,

прежде чем переходить к проектированию их, убедитесь, что все таблицы БД у вас созданы. Во-вторых, при помещении этих объектов на формы необходимо менять их модификатор доступа на public.

Пусть в нашей БД уже созданы две таблицы Sotrudniki (сотрудники) и Kontr (контрагенты).

Разместим на форме два компонента DataGridView и сделаем так, что в одной отобразятся данные о сотрудниках, а во второй – о контрагентах. Переходим к первой DataGridView и нажимаем на значок треугольник в белом квадрате, расположенный в правом верхнем углу компонента. Нажимаем Choose Data Source (рисунок 1.4.).

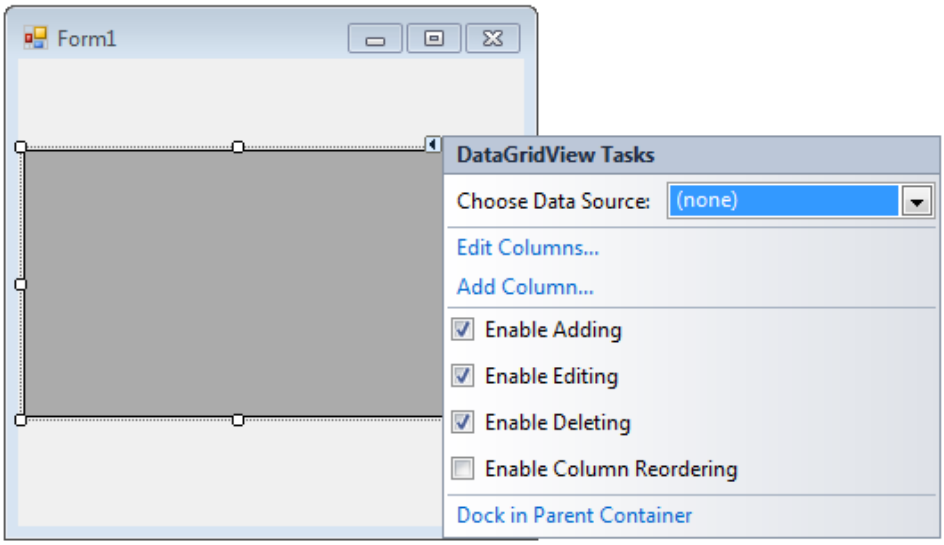

Рисунок 1.4. Настройка компонента DataGridView

Далее, если список доступных Data Source пуст (а для первого компонента он и должен быть пуст), выбираем Add Project Data Source -> DataBase->DataSource. Если в списке подключений есть подключение к БД, то выбираем его, если нет, то выбираем New Connection и создаем подключение, как было описано ранее. Пропускаем два диалоговых окна, нажимая Next, затем в списке доступных элементов подключения ставим галочку напротив Tables. Далее у DataGridView в Choose Data Source выбираем Other Data

Source -> Project DataSource и ставим галочку напротив первой таблицы, во втором DataGridView добираемся тем же путем до второй. После этого у вас должны появиться пять значимых для нас компонентов на форме: myDBDataSet, kontrBindingSource, sotrudnikiBindingSource, kontrTableAdapter и sotrudnikiTableAdapter. У обоих DataGridView снимите галочки Enabled Additing, Deleting и Editing. Удаление, добавление и редактирование будут реализованы по-другому.

Рассмотрим реализацию удаления. Пусть на форме будет расположена кнопка, по нажатию которой должно происходить удаление текущего элемента из таблицы Sotrudniki. Тогда в ней необходимо прописать следующий код:

```
//пометка на удаление:
sotrudnikiBindingSource.RemoveCurrent();

//сохранение изменений:
sotrudnikiBindingSource.EndEdit();

//выгрузка в DataGridView обновленных данных:
sotrudnikiTableAdapter.Update(this.myDBDataSet.Sotrudniki);
```

Теперь рассмотрим добавление и редактирование. Для начала пройдем подготовительный этап: добавим в проект новую форму и разместим на ней текстовое поле под ФИО нашего сотрудника и две кнопки. В форме настроим следующие свойства: FormBorderStyle – FixedDialog, ControlBox – false; AcceptButton – первая кнопка, CanselButton – вторая кнопка. Это сделано для того, чтобы форму можно было закрыть только по нажатию расположенных на ней кнопок, у которых настроим следующие свойства: у первой кнопки свойство dialogResult – OK, у второй это же свойство – Cancel. У TextBox выбираем в свойстве DataBindings->Text поле fio нашей таблицы. У появившегося на форме элемента sotrudnikiBindingSource ставим свойство Modifier в значение public. Таким образом мы

связали наш компонент TextBox с определенным полем таблицы БД. Теперь переходим к событиям формы и в событии FormClosing прописываем следующий код:

```
//если пользователь нажал на первую кнопку:
if (DialogResult == System.Windows.Forms.DialogResult.OK)
//сохранить изменения:
            sotrudnikiBindingSource.EndEdit();
        else
//не сохранять изменения:
            sotrudnikiBindingSource.CancelEdit();
```

Теперь в основной форме размещаем кнопку «добавление нового сотрудника» и в ней пишем код:

```
//создаем объект второй формы:
          Form2 f = new Form2();

//добавляем новую запись в таблицу:
           sotrudnikiBindingSource.AddNew();

//синхронизируем компоненты bindingSource обоих форм:
     f.sotrudnikiBindingSource.DataSource = sotrudnikiBindingSource;

//чтобы они указывали в таблице на одну и ту же запись:
     f.sotrudnikiBindingSource.Position                                    =
sotrudnikiBindingSource.Position;

//если пользователь в форме добавления нажал на первую кнопку:
        if (f.ShowDialog() == System.Windows.Forms.DialogResult.OK)

//осуществляем выгрузку в DataGridView обновленных данных:
            sotrudnikiTableAdapter.Update(this.myDBDataSet.Sotrudniki);
```

Для редактирования данных о сотруднике добавляем на форму кнопку «редактирование текущего» и в ней пишем практически такой же код. Разница будет только в отсутствии добавления новой записи в таблицу:

```
//создаем объект второй формы:
          Form2 f = new Form2();
```

```
//синхронизируем компоненты bindingSource обоих форм:
    f.sotrudnikiBindingSource.DataSource = sotrudnikiBindingSource;

//чтобы они указывали в таблице на одну и ту же запись:
    f.sotrudnikiBindingSource.Position =
sotrudnikiBindingSource.Position;

//если пользователь в форме добавления нажал на первую кнопку:
        if (f.ShowDialog() == System.Windows.Forms.DialogResult.OK)

//осуществляем выгрузку в DataGridView обновленных данных:
            sotrudnikiTableAdapter.Update(this.myDBDataSet.Sotrudniki);
```

Рассмотрим теперь редактирование связанных элементов. Пусть в базе данных есть еще таблица договоров, в которой, кроме номера и даты договора, хранится информация о том, какой сотрудник и с каким контрагентом этот договор заключил. Тогда при создании нового элемента этой таблицы нужно будет выбирать наименование контрагента и наименование сотрудника. То же самое должно происходить при редактировании: эти поля должны выбираться, а не вводиться. Сразу стоит уточнить, что в таблице нужно хранить не ФИО сотрудника и наименование контрагента, а идентификаторы этих записей, соответственно, выбираться должны поля fio (у сотрудника) и name (у контрагента), а записываться в обоих случаях – id. Первое, что вам необходимо сделать, – создать в DataSet новое представление данных, в которое выгрузить идентификатор и наименование для каждого из связываемых справочников. Начнем с контрагентов. Переходим в окне SolutionExplorer к файлу MyDBDataSet.xsd, нажимаем на нем правой кнопкой мыши для перехода во View Designer. Копируем и вставляем табличку с контрагентами. В результате создастся Kontr1TableAdapter. Нажимаем на нем правой кнопкой мыши, переходим в свойства и выставляем значения none в InsertCommand и UpdateCommand. Затем переходим к свойству SelectCommand и меняем следующим образом.

Во-первых, разворачиваем его и заходим в Command, в окне с SQL-запросом пишем следующую команду:

```
SELECT      id, CONVERT(varchar, id) + ' ' + Name AS NAME
FROM        Kontr
```

Аналогичное действие выполняем для сотрудников. Теперь добавляем в проект форму для создания или редактирования информации о договоре. Добавляем несколько компонентов TextBox для полей договора, а также помещаем два ComboBox: один для сотрудников, другой для контрагентов. Рассмотрим дальнейшие действия на примере контрагентов. Во-первых, в свойстве DataSource у ComboBox выбираем Kontr1bindingsource, в свойстве DisplaMember выбираем Name, а в свойстве ValueMember – id. В событии combobox1_selectedIndexChanged пишем следующий код:

```
//если что-то было выбрано
if (comboBox1.SelectedValue != null)
        {

//берем текущую строку таблицы по договорам
 MyDBDataSet.DogovorRow r =
  (MyDBDataSet.DogovorRow)
((DataRowView)dogovorBindingSource.Current).Row;

//меняем id контрагента на выбранное значение
  r.idkontr = comboBox1.SelectedValue.ToString();
        }

//сохраняем изменения
        dogovorBindingSource.EndEdit();
        dogovorTableAdapter.Update(myDBDataSet.Kontr);
```

Для сотрудников все выполняется аналогично.

Теперь рассмотрим решение следующей задачи: пусть при выборе определенного элемента справочника сотрудников в таблице отображаются только те договора, которые он заключил. На форме должно быть размещено два DataGridView: одно для сотрудников,

другое для договоров. Для DataGridView сотрудников (пусть оно будет называться DataGridView1) пишем в обработчике события CellClick:

```
//берем текущую строку таблицы сотрудников
MyDBDataSet.SotrudnikiRow r =
(MyDBDataSet.SotrudnikiRow)((DataRowView)sotrudnikiBindingSource.Curre
nt).Row;

//накладываем для договоров фильтр по id выбранной записи
  dogovorBindingSource.Filter = "idsotr = " + r.id.ToString();
```

Если необходимо, чтобы при нажатии на какую-либо кнопку отображались договора всех сотрудников независимо от выбранного, то в обработчике нажатия этой кнопки прописываем:

```
//сброс фильтра
        dogovorBindingSource.Filter =string.Empty;
```

Рассмотрим еще одно решение задачи фильтрации. Пусть нам необходимо отобразить в dataGridView список контрагентов, чье наименование содержит введенное пользователем значение. Для этого переходим в редактор DataSet (в окне SolutionExplorer на файле MyDBDataSet.xsd нажимаем правой кнопкой мыши и переходим во View Designer). На таблицу KontrTableAdapter нажимаем правой кнопкой и переходим к Add query. Выбираем «Use select statements», далее «Select with returns rows», далее дополняем имеющийся там запрос строчкой:

```
WHERE      (Name like '%@Name%')
```

В поле MethodName раздела FillDataTable пишем FillByName, во втором разделе – GetDataByName.

С датами и числовыми значениями все выполняется аналогично, согласно синтаксису SQL. Соответственно, если нужно будет сделать несколько разных фильтров, то будет несколько разных методов.

Теперь на форму помещаем DataGridView, настроенный на таблицу контрагентов, размещаем на форме текстовое поле, в которое пользователь будет вводить данные и кнопку, по нажатию которой будет происходить отображение данных. В ней пишем код:

```
kontrTableAdapter.FillByName(myDBDataSet.Kontr,textBox1.Text);
```

Мы рассмотрели работу с таблицами базы данных через визуальные компоненты, но некоторые управляющие функции (программное создание или очистка таблиц) необходимо реализовывать по-другому. Рассмотрим это на примере создания отдельной библиотеки.

Создаем новое приложение ClassLibrary и описываем в нем класс, который будет выполнять все управляющие функции. Так как он будет работать с базой данных, то не забываем подключить пространство имен:

```
 using System.Data.SqlClient;
```

Для сохранения настроек подключения описываем строковое поле:

```
class UpravClass
   {
      string connectionString = "";
   }
```

Для изменения поля определим свойство, в котором пропишем дополнительную информацию о подключении, а в параметре value будем ожидать только путь к файлу базы данных:

```
class UpravClass
  {
     string connectionString = "";
     public string ConsStr
     {
        get { return connectionString; }
        set { connectionString = @"Data Source=.\SQLEXPRESS;AttachDbFilename=" + value + ";Integrated Security=True;Connect Timeout=30;User Instance=True"; }
     }
  }
```

Теперь определим конструктор для создания объекта этого класса, передадим ему путь к файлу БД и установим значение этого поля.

```
class UpravClass
  {
     string connectionString = "";
     public string ConsStr
     {
        get { return connectionString; }
        set { connectionString = @"Data Source=.\SQLEXPRESS;AttachDbFilename=" + value + ";Integrated Security=True;Connect Timeout=30;User Instance=True"; }
     }
     public UpravClass(string dbpath)
     {        ConsStr = dbpath;      }   }
```

При выполнении различных управляющих функций в БД (создание, очистка, уничтожение таблиц, вставка) мы будем выполнять всегда следующие действия:

- открывать соединение с БД,
- выполнять команду,
- возвращать сообщение об успешном выполнении команды или ошибке.

Соответственно, у нашего управляющего класса можно определить метод:

```
public string ExecSQL(string query)
{
try
        {
            using (SqlConnection conn = new SqlConnection(ConsStr))
            {
                conn.Open();
                using (SqlCommand c = new SqlCommand(query))
                {
                    c.Connection = conn;
                    c.ExecuteNonQuery();
                }
            }
return “Execution completed sucsessful”;
        }
        catch (Exception exp)
        {
return “Excecution error: ”+exp.Message;
        }
}
```

Следовательно, для любого действия нам необходимо будет просто сформировать строку запроса и вызвать метод.

Теперь мы можем рассмотреть пример работы с этим классом. Компилируем библиотеку и подключаем ее к требуемому проекту WindowsForms. Помещаем на форме кнопку, в ней создаем объект нашего управляющего класса, в конструкторе передав ему путь к базе данных, затем вызываем его метод ExecSQL с запросом на создание таблицы и заполнение ее тестовыми данными. Пример для справочника Sotrudniki:

```
UpravClass u=new UpravClass(@”D:\MyDB.mdf”);

string query = “CREATE TABLE Sotrudniki (id INT Identity PRIMARY KEY,fio varchar(50));";

u.ExecSQL(query);
for(int i=0;i<10;i++)
{query ="Insert into sotrudniki(fio)values('Sotrudnik'”+i.toString()+”)";
u.ExecSQL(query);}
```

Для очистки таблицы Sotrudniki напишите следующий код:

```
UpravClass u=new UpravClass(@”D:\MyDB.mdf”);
string query = “truncate TABLE Sotrudniki";
 u.ExecSQL(query);
```

Для удаления таблицы Sotrudniki из БД напишите:

```
UpravClass u=new UpravClass(@”D:\MyDB.mdf”);
string query = “drop TABLE Sotrudniki";
 u.ExecSQL(query);
```

Обратите внимание, что если вы удалите таблицу из БД, вам ее придется заново создавать, если захотите с ней еще поработать.

Теперь рассмотрим работу с базами данных через технологию LINQ [6]. Напомним, что LINQ (Language-Integrated Query) — это набор функций, предоставляющий стандартные, простые в изучении шаблоны для запросов на выборку и изменение данных. Эти технологии могут быть расширены для поддержки практически любого типа источника данных. При работе с базами данных основным будет класс DataContext, предоставляющий широкий доступ к базе данных. Его функция: переводить ваши запросы к сущностям в операторы языка SQL, которые в итоге выполняются на подключенной базе данных. Для получения доступа к функционалу этого класса вам необходимо создать свой класс, производный от DataContext и тогда вам будут доступны таких методы, как ExecuteQuery, ExecuteCommand и SubmitChanges. Кроме того, ваш класс должен будет содержать свойства типа System.Data.Linq.Table<T> для каждой таблицы и представления в базе данных. То есть необходимо будет выполнить отображение базы данных в объектную модель, или, иными словами, выполнить Mapping базы данных. Mapping в данном случае выполняется декларативно, с помощью специальных атрибутов, указываемых при

описании классов-моделей таблиц БД. Рассмотрим эти атрибуты подробнее. Первый атрибут – Table. Он используется для связи класса с таблицей в базе данных. Причем, необходимо отметить, что технология LINQ в данном случае поддерживает только одиночный mapping, то есть один класс соответствует одной таблице. Атрибут Table может иметь свойство Name – название таблицы. Например, выполним mapping для таблицы, хранящей данные сотрудников в нашей БД:

```
[Table(Name = "Sotrudniki")] public class Sotr{ //описание класса }
```

В данном случае имя класса не совпадает с именем таблицы, поэтому у атрибута было использовано свойство Name. Если это свойство не использовать, то код будет выглядеть так:

```
[Table]public class Sotrudniki
{    //описание класса }
```

Следующий используемый в mapping атрибут – Column. Он описывает связь поля или свойства класса со столбцом в таблице. У атрибута могут быть следующие свойства:

IsPrimaryKey – указывает, является ли поле первичным ключом;

IsDbGenerated – указывает, является ли поле генерируемым базой данных;

AutoSync – указывает, в какой момент определяется значение;

DbType – указывает тип данных в БД;

CanBeNull – указывает, допускаются ли пустые значения (null) в данном поле;

Name – указывает название колонки в таблице БД.

Продолжим выполнять mapping для нашей таблицы сотрудников:

```
 [Table]public class Sotrudniki
{

private int  _id;

  [Column(IsPrimaryKey = true, IsDbGenerated = false, DbType = "INT",
CanBeNull = false, AutoSync = AutoSync.OnInsert)]
 public int Id
   {   get  { return  _id; }
       set  {  if (_id != value)
   {NotifyPropertyChanging(()=>Id);  _id = value;
NotifyPropertyChanged(()=>Id);}
            }
   }

         private string _name;

[Column(DbType = "NVARCHAR(300)")]
 public string fio
   {  get  {              return _name;       }
      set  {              if (_name != value) {
NotifyPropertyChanging(()=>fio); _name=
value;NotifyPropertyChanged(()=>fio);}
            }
    }
}
```

Следующий атрибут – Association – описывает связь типа «ассоциация», то есть связь по внешнему ключу, в терминах базы данных. С помощью этого атрибута можно настраивать связи следующего типа: «один к одному» и «один ко многим». Данный атрибут может иметь следующие свойства.

DeleteOnNull – указывает, надо ли удалять связанную сущность при связи «один к одному» и обнулении свойства.

DeleteRule – указывает правило удаления (NO ACTION,CASCADE).

IsForeignKey – указывает, является ли поле внешним ключом.

IsUnique – определяет уникальность поля.

Name – указывает название поля.

OtherKey – определяет внешний ключ.

Storage – указывает название поля, где хранится сущность/ коллекция.

ThisKey – указывает первичный ключ.

Рассмотрим работу с этим атрибутом на примере. Пусть в нашей базе данных есть таблица с адресами, которая связывается с таблицей сотрудников по идентификатору каждого:

```
[Table]
public class Sotrudniki
{
  private EntitySet<Address> _addresses = new EntitySet<Address>();

   [Association(Storage = "_addresses", OtherKey = “_sotrid”]
       public EntitySet<Address> Adres  {
          get { return _addresses; }
          set { _addresses.Assign(value);} }
 }

[Table]
public class Address
{
    private int _id;

    [Column]
     public int id  { get { return _id; } }

private  EntityRef<Sotrudniki> _sotr;

[Association(Storage = "_sotr", ThisKey = "_sotrId", OtherKey = "id",
IsForeignKey = true)]
 public Sotrudniki Sotrudniki
   {  get  {              return _sotr.Entity;           }
      set  {NotifyPropertyChanging(() => Sotrudniki);_sotr.Entity = value;
           NotifyPropertyChanged(() => Sotrudniki);
            if(value != null) _sotrId = value.Id;
           }
    }
}
```

Последний атрибут – Index – описывает индекс для ускорения выборок и сортировок. Свойства этого атрибута:

Columns – определяет список колонок для построения индекса. Можно указывать несколько и соответствующую сортировку, например Name ASC, Number DESC;

IsUnique – определяет уникальность;

Name – указывает название.

Рассмотрим пример создания индекса для поиска сотрудников по имени:

```
[Index(Columns = "fio", IsUnique = true, Name= "Sotr_Name")]
[Table]public class Sotrudniki{  // здесь описаны поля }
```

При следующем запросе построенный индекс будет использован:

```
var sotr = from c in db.Sotrudniki where c.fio ="Петров Иван" select c;
```

Рассмотрим пример реализации приложения, работающего базой данных, а в частности, с таблицей сотрудников и выполняющего удаление, редактирование, добавление и выборку записей.

Сначала подключаем пространства имен:

```
using System.Data.Linq.Mapping;
using System.Data.Linq;
```

Затем создаем класс-аналог нашей таблицы в БД, используя маппинг:

```
[Table]
  public class Sotrudniki
  {  [Column(IsPrimaryKey=true, IsDbGenerated=true) ]
     public int id;
     [Column]
     public string fio;   }
```

Далее создаем класс, наследник от DataContext, с конструктором и свойством – нашей таблицей:

```
public class DB : DataContext
  {
      public DB(string cs)
          : base(cs)
      {
      }

      public System.Data.Linq.Table<Sotrudniki> Sotrudniki
      {
          get { return this.GetTable<Sotrudniki>(); }

      }
  }
```

Конструктор класса принимает в качестве параметра имеющуюся у нас строку подключения.

В классе DB объявляем методы для соответствующих действий. Метод проверки наличия базы данных и создания ее при отсутствии:

```
public void Check()
{
if (!db.DatabaseExists())
  {
    db.CreateDatabase();
  }
}
```

Метод удаления базы данных:

```
public void Delete()
{
        if (db.DatabaseExists())
  {
    db.DeleteDatabase();
  }
}
```

Метод выборки данных из таблицы:

```
public List<Sotrudniki> Zapros(int id)
{
	return db.Sotrudniki.Where(c => c.Id > id).ToList();
}
```

Метод добавления записи:

```
public void ADD(string fio)
{
Sotrudniki sot = new Sotrudniki();
sot.name=fio;
db.Sotrudniki.InsertOnSubmit(sot);
db.SubmitChanges();
}
```

Метод обновления записи:

```
public void Edit(int id,string fio)
{
Sotrudniki sot = db.Sotrudniki.Where(c => c.id == id).FirstOrDefault();
sot.name=fio;
db.SubmitChanges();
}
```

Метод удаление записи:

```
public void Delete(int id)
{
Sotrudniki sot = db.Sotrudniki.Where(c => c.id == id).FirstOrDefault();
db.Sotrudniki.DeleteOnSubmit(sot);
db.SubmitChanges();
}
```

Обратите внимание, что все изменения сохраняются в базу данных только при вызове метода SubmitChanges.

Mapping базы данных позволяет глубоко интегрировать базу данных в платформу, что наряду с плюсами, таит и минусы. К плюсам можно отнести упрощение реализации приложения за счет

уменьшения времени и количества ошибок. К минусам относится тот факт, что мы должны заранее знать СУБД, которое будет управлять нашей базой данных. Кроме того, в настоящее время маппинг полностью поддерживается только для Microsoft SQL Server. Поэтому целесообразно рассмотреть вариант комбинации технологий: LINQ to DataSet [7].

Для работы с этой технологией нам необходимо подключить namespace:

```
using System.Data.SqlClient;
using System.Data.Linq;
```

Затем нам нужно получить строку подключения следующего вида:

```
string s=@"Data
Source=.\SQLEXPRESS;AttachDbFilename=D:\MyDB.mdf;Integrated
Security=True;Connect Timeout=30;User Instance=True";
```

Теперь можно создать DataSet и заполнить его имеющимися таблицами БД, которые нам будут нужны для запросов.

```
SqlDataAdapter da = new SqlDataAdapter("Select * from Kontr",s);
SqlDataAdapter da2 = new SqlDataAdapter("Select * from Sotrudniki", s);
   DataSet ds = new DataSet();
da.Fill(ds, "Kontr");
      da2.Fill(ds, "Sotrudniki");
```

Теперь мы готовы к извлечению данных. Допустим, мы хотим вывести в listBox всех сотрудников, тогда код будет следующим:

```
DataTable dt = ds.Tables["Sotrudniki
var q = dt.AsEnumerable()
        .Select(t => t);
listBox1.Items.Clear();
foreach (var i in q)
{ listBox1.Items.Add(i.Field<int>("id") + "   " + i.Field<string>("fio"));      }
```

Или осуществить выборку по условию (допустим, только сотрудника с id=8):

```
DataTable dt = ds.Tables["Sotrudniki"];
int id = 8;
var q = dt.AsEnumerable()
   .Where(t=>t.Field<int>("id")==id)
   .Select(t => t);
listBox1.Items.Clear();
foreach (var i in q)
{ listBox1.Items.Add(i.Field<int>("id") + "   " + i.Field<string>("fio"));  }
```

Или реализовать объединение таблиц (например, выведем для каждого контрагента сотрудника с таким же id, как и у него):

```
DataTable dt = ds.Tables["Kontr"];
var q = dt.AsEnumerable()
      .Select(t => t)
   .Join(ds.Tables["Sotrudniki"].AsEnumerable(),
   w => w.Field<int>("id"),
   t => t.Field<int>("id"), (w, t) => new
   {
      name1=w.Field<string>("name"),
      name2=t.Field<string>("fio"),
id1=w.Field<int>("id"),
      id2 = t.Field<int>("id")
   });
listBox1.Items.Clear();
foreach (var i in q)
{   listBox1.Items.Add(i.id1+" "+ i.id2+" " + i.name1+" "+ i.name2);   }
```

Мы рассмотрели технологии работы с базами данных. В каждой есть свои плюсы и минусы, и решение о применении каждой в отдельности или их комбинации зависит от специфики решаемой задачи.

Контрольные вопросы к разделу 1

1. Что такое сериализация?
2. Что такое десериализация?
3. Расскажите, как в C# можно осуществить XML-сериализацию.

4. Перечислите технологии, используемые для работы с базами данных.
5. Что такое маппинг?
6. Назовите основной класс, используемый в технологии LINQ.
7. Перечислите основные компоненты, используемые в технологии ADO.Net.
8. Перечислите плюсы и минусы каждой из рассмотренных технологий.
9. Напишите код выборки данных по товару с идентификатором=1 из таблицы товаров для технологии LINQ.
10. Что такое строка подключения?

2. ПОСТРОЕНИЕ ОТЧЕТОВ

В бизнес-процессах любого предприятия так или иначе фигурирует пункт «Формирование отчетов». Это могут быть ежедневные, еженедельные или ежемесячные документы, отражающие те или иные аспекты деятельности фирмы, но механизм их автоматизированного построения обычно одинаковый: запрос к базе данных, обработка и вывод результата в какой-либо файл. Обычно файл отчета – это или текстовый документ, или электронная таблица. Поэтому в данном разделе мы рассмотрим технологии создания отчетов в следующих форматах:

- в OpenOffice.calc;
- в OpenOffice.writer;
- в MS Excel;
- в MS Word;
- в формате PDF.

При выгрузке больших объемов данных для формирования отчета может потребоваться длительное время. Для того чтобы повысить эффективность приложений, строящих громоздские отчеты,

нередко используют технологию многопоточности. В данном разделе мы также рассмотрим ее возможности.

2.1. Работа с OpenOffice

В настоящее время возрастает популярность бесплатного программного обеспечения. Во многих офисах вместо платных программ устанавливают бесплатные аналоги. Одним из таких является свободный пакет офисных приложений Apache OpenOffice. В настоящее время он успешно конкурирует как на уровне форматов, так и на уровне интерфейса пользователя с коммерческими офисными пакетами. В состав OpenOffice входят следующие модули:

- Writer – это текстовый процессор и визуальный редактор HTML, успешно конкурирующий с Microsoft Word, Pages, AbiWord, KWord;
- Calc – табличный процессор, схожий с Microsoft Excel, Numbers, Gnumeric, KSpread;
- Impress – модуль для подготовки презентаций, похожий на Microsoft PowerPoint, Keynote, KPresenter;
- Base – модуль работы с БД, в чем-то даже превосходящий Microsoft Access и Kexi;
- Draw – векторный редактор, способный конкурировать с Microsoft Visio, Microsoft Expression, Desigh, Adobe Illustrator, CorelDRAW, Kivio, Dia;
- Math – редактор формул, схожий по функционалу с MathType, Microsoft Equation Tools, KFormula.

Как было сказано выше, обычно отчеты на предприятиях составляются в табличной или текстово-табличной форме, поэтому из всех модулей пакета OpenOffice мы рассмотрим только Writer и Calc. Существует два способа взаимодействия приложения .Net и этих модулей: использование библиотек (CLI-сборок), поставляющихся с

пакетом или с использованием технологии отражения. Первый способ, несомненно, удобнее в использовании для разработчика, но имеет ряд ограничений. Во-первых, он не будет работать с ранними версиями пакета, во-вторых, версия библиотек может конфликтовать с версией Framework, в-третьих, версия библиотек должна совпадать с версией OpenOffice, т. е. возможна проблема в переносимости приложения с одной машины на другую. Второй способ не обладает описанными недостатками, но существенно более сложен в использовании, так как необходимо знать все методы и свойства внутреннего языка пакета и их параметры. Рассмотрим оба метода более подробно и начнем с использования CLI-сборок.

Первое, что нам необходимо сделать, – подключить данные сборки к нашему проекту. Для этого выполняем следующие действия. Во-первых, находим эти библиотеки в каталоге, где установлен OpenOffice, в папке с именем «assambly». Там должны находиться следующие файлы:

- cli_basetypes.dll
- cli_cppuhelper.dll
- cli_types.dll
- cli_ure.dll

Скопируйте их и переместитесь в каталог проекта. Здесь создайте папку с именем «Resources», в которую и переместите сборки. Теперь подключите эти библиотеки к своему проекту (в обозревателе решений Solution Explorer щелкните правой кнопкой мыши на пункте «References», выберите «Add references…» и найдите свои сборки).

Для более удобной работы с классами из сборок вам необходимо подключить следующие пространства имен:

```
using unoidl.com.sun.star.lang;
using unoidl.com.sun.star.uno;
```

```
using unoidl.com.sun.star.bridge;
using unoidl.com.sun.star.frame;
```

Основным объектом для подключений к OpenOffice.org является ServiceManager. Он служит точкой входа для приложения и передается каждому компоненту при создании экземпляра. Часто компоненту требуется больше информации или функциональности после развертывания. В этом ключе возможности ServiceManager ограничены. Поэтому была создана концепция ComponentContext. Он представляет собой хранилище именованных объектов с доступом только для чтения. Один из таких объектов – ServiceManager. То есть по сути это среда, в которой живут компоненты приложения (как область памяти для переменных программы). ComponentContext передается приложению при его запуске. Поэтому первое, что нам необходимо сделать для работы с OpenOffice, это запустить его. Делается это вызовом статического метода bootstrap класса Bootstrap, возвращающего ссылку на объект ComponentContext:

```
unoidl.com.sun.star.uno.XComponentContext localContext =
 uno.util.Bootstrap.bootstrap();
```

Затем мы должны получить ссылку на объект диспетчера служб ServiceManager (экземпляр класса MultiServiceFactory), для того чтобы позже получить объект класса Desktop и создать новый объект CLI:

```
unoidl.com.sun.star.lang.XMultiServiceFactory multiServiceFactory =
  (unoidl.com.sun.star.lang.XMultiServiceFactory)
localContext.getServiceManager();
```

Desktop – центральный управляющий экземпляр для платформы приложений OpenOffice.org. Все их окна организованы как иерархия фреймов, содержащих видимые компоненты. Desktop – корень этой иерархии. Через этот компонент можно загрузить видимые

компоненты, завершить их работу или перенаправить запросы. Для создания нового экземпляра Desktop мы используем наш диспетчер служб, передав команду загрузчику XComponent:

```
XComponentLoader componentLoader = (XComponentLoader) multiServiceFactory.createInstance("com.sun.star.frame.Desktop" );
```

Итак, как видно из сказанного выше, работу OpenOffice обеспечивают три объекта: компонент, диспетчер служб и Desktop.

Теперь перейдем к созданию нового пустого документа writer, используя наш объект Desktop:

```
XComponent  xComponent = componentLoader.loadComponentFromURL(
   "private:factory/swriter",  "_blank",
 0, new unoidl.com.sun.star.beans.PropertyValue[0] );
```

Рассмотрим подробнее параметры этого метода. Первый содержит адрес того, что нужно загрузить. Если пишем «private:factory/swriter», то загрузится пустой документ модуля writer, а если укажем «private:factory/scalc», то загрузится пустой calc. Если вместо этого укажем имя файла, то загрузится требуемый файл. Второй параметр отвечает за то, в каком окне открыть загружаемый компонент: в новом или в одном из ранее открытых. Третий и четвертый – дополнительные параметры открытия.

Теперь рассмотрим запись текста в документ модуля writer. Основным элементом в данном случае является текстовое поле. Соответственно, сначала обращаемся к нему, вызывая метод getText, затем для записи текста используем метод setString.

```
((unoidl.com.sun.star.text.XTextDocument)xComponent).getText().setString(" Привет! Я - первый текст.");
```

Обратите внимание на то, что ваше приложение, возможно, будет работать медленно при первом запуске OpenOffice. Для того чтобы его ускорить, рекомендуется самостоятельно перед началом

работы открыть OpenOffice, дать ему пару минут поработать и затем закрыть. Это происходит из-за того, что при первом запуске происходит поиск обновлений в Интернет.

Необходимо отметить, что метод setString устанавливает содержимое всего документа. То есть если мы несколько раз вызовем этот метод с разными параметрами-строками, то в итоге в документ будет выведена только одна последняя строка. Для добавления текста в начало или в конец документа мы можем воспользоваться следующими способами:

```
unoidl.com.sun.star.text.XTextRange
x=((unoidl.com.sun.star.text.XTextDocument)
xComponent).getText().getStart(); // в начало
((unoidl.com.sun.star.text.XTextDocument)xComponent).getText().insertString
(x, "fgfdgdfgdfg", true);
```

и:

```
x =
((unoidl.com.sun.star.text.XTextDocument)xComponent).getText().getEnd(); //
в конец
((unoidl.com.sun.star.text.XTextDocument)xComponent).getText().insertString
(x, "324324234", true);
```

Вставка таблиц в документ более сложна, нежели чем работа с текстом. Сначала необходимо создать объекты XFrame и XDispatchHelper.

Первый необходим для получения ссылки на документ, в который вставляется таблица, а второй обеспечивает выполнение функций пользовательского интерфейса (в данном случае функции вставки таблицы в документ).

Формирование таблицы создается в два этапа: сначала создается массив свойств таблицы, а затем для записи текста в таблицу создается массив свойств ячейки. После этого с помощью объекта XDispatchHelper вставляется текст в ячейки, а сама таблица – в

документ. Ниже приведен пример, создающий таблицу из двух столбцов и одной строки.

```
XFrame frame = ((unoidl.com.sun.star.text.XTextDocument)xComponent).
getCurrentController().getFrame();

        XDispatchHelper xDispatchHelper =
 (XDispatchHelper)multiServiceFactory.createInstance
("com.sun.star.frame.DispatchHelper");

        unoidl.com.sun.star.beans.PropertyValue[] tableArgs =
 new unoidl.com.sun.star.beans.PropertyValue[4];
        tableArgs[0] = new unoidl.com.sun.star.beans.PropertyValue();
        tableArgs[1] = new unoidl.com.sun.star.beans.PropertyValue();
        tableArgs[2] = new unoidl.com.sun.star.beans.PropertyValue();
        tableArgs[3] = new unoidl.com.sun.star.beans.PropertyValue();
        tableArgs[0].Name = "TableName";
        tableArgs[0].Value = new uno.Any("MyFavouriteTable");
        tableArgs[1].Name = "Columns";
        tableArgs[1].Value = new uno.Any(2);
        tableArgs[2].Name = "Rows";
        tableArgs[2].Value = new uno.Any(1);
        tableArgs[3].Name = "Flags";
        tableArgs[3].Value = new uno.Any(9);

        unoidl.com.sun.star.beans.PropertyValue[] cellTextArgs =
new unoidl.com.sun.star.beans.PropertyValue[1];
        cellTextArgs[0] = new unoidl.com.sun.star.beans.PropertyValue();
        cellTextArgs[0].Name = "Text";
        cellTextArgs[0].Value = new uno.Any("Hello! My lovely world!");

        xDispatchHelper.executeDispatch((XDispatchProvider)frame,
".uno:InsertTable", "", 0, tableArgs);
        xDispatchHelper.executeDispatch((XDispatchProvider)frame,
".uno:InsertText", "", 0, cellTextArgs);


        xDispatchHelper.executeDispatch((XDispatchProvider)frame,
".uno:JumpToNextCell", "", 0, new
unoidl.com.sun.star.beans.PropertyValue[0]);
        xDispatchHelper.executeDispatch((XDispatchProvider)frame,
".uno:InsertText", "", 0, cellTextArgs);
```

Мы рассмотрели вставку текста в документ. Теперь рассмотрим его считывание. При работе через сборки для считывания текста будет использоваться метод getString (противоположный по действию методу setString). В приведенном ниже примере в консоль выводится содержимое документа:

```
string s = (string)((unoidl.com.sun.star.text.XTextDocument)xComponent).
 getText().getString();
 Console.WriteLine(s);
```

Здесь необходимо сделать небольшое отступление и рассказать об еще одном способе извлечения текста из OpenOffice.writer, не имеющем отношения ни к работе с библиотеками, ни к технологии отражения. Речь пойдет о ручном разборе контейнера ODT, который по сути представляет собой ZIP-архив с файлами данных, описанных в формате XML, а также другими необходимыми файлами описания и объектами документа. Для распаковки архива можно использовать как встроенные средства платформы .Net, так и сторонние библиотеки (например, Ionic.Zip).

Текст, таблицы и ссылки на объекты хранятся внутри расположенного в архиве файла content.xml в некой структуре. Хранение графических объектов осуществляется следующим образом: внутри архива находится папка Pictures, где хранятся сами графические файлы, а ссылка на них, как было сказано ранее, хранится в файле content.xml. Приведем пример ссылки на картинку [2]:

```
<draw:frame draw:style-name="fr2" draw:name="Графический объект1"
text:anchor-type="paragraph" svg:x="0.037cm" svg:y="0.201cm"
svg:width="4.217cm" svg:height="3.302cm" draw:z-index="1">
<draw:image
xlink:href="Pictures/1000000000000CC00000099028356212.jpg"
xlink:type="simple" xlink:show="embed" xlink:actuate="onLoad" />
</draw:frame>
```

Для хранения форматированного текста используется следующая конструкция. В самом начале файла content задаются именованные стили, а затем в тэге, хранящем сам текст, приводится ссылка на этот стиль. В приводимом ниже примере стиль именуется P37.

```
<style:style style:name="P37" style:family="paragraph" style:parent-style-name="Text_20_body">
<style:text-properties fo:color="#800000" style:font-name="Times New Roman1" fo:font-size="14pt" fo:language="ru" fo:country="RU" fo:font-style="italic" fo:font-weight="bold" /> </style:style>
….<text:p text:style-name="P37">В данном задании все выполнено верно (5 баллов)</text:p>
```

Извлечение текста сводится к разбору данного файла, а также при необходимости еще и к разбору файла styles.xml, где описаны стили форматирования данного документа.

Данный метод не очень удобен в применении, так как нет официальной документации, описывающей структуру архивных файлов.

Теперь рассмотрим работу с текстом в модуле calc. В данном случае процесс упрощается в плане навигации, но усложняется в плане количества задействованных в нем объектов. Если у модуля «Writer» был только Text, то у модуля «Calc» есть листы и ячейки. Соответственно, чтобы записать текст, необходимо обратиться к конкретной ячейке через содержащий ее лист.

Рассмотрим пример записи текста в первую ячейку первого листа. Для этого сделаем следующее. Сначала обратимся ко всем листам нашего документа, возьмем первый по имени, обратимся к его ячейке по индексам и запишем формулу в нее. Ссылку на коллекцию листов нам вернет метод getSheets, затем у объекта-ссылки на коллекцию вызывем метод getElementNames(), возвращающий ссылку на коллекцию имен листов книги. Передадим нулевой аргумент этой коллекции в метод getByName, вызываемый у объекта-ссылки на коллекцию листов, и получим ссылку на первый лист через свойсво Value.

```
unoidl.com.sun.star.sheet.XSpreadsheets d =
(unoidl.com.sun.star.sheet.XSpreadsheets)(((unoidl.com.sun.star.sheet.XSpre
adsheetDocument)xComponent).getSheets());

  uno.Any t= d.getByName(d.getElementNames()[0]);

    unoidl.com.sun.star.sheet.XSpreadsheet f =
(unoidl.com.sun.star.sheet.XSpreadsheet)t.Value;
```

Теперь у объекта-ссылки на лист вызовем метод getCellByPosition, передав в качестве параметров индексы ячейки, и у полученного объекта вызовем метод setFormula, передав в качестве параметра текст для записи.

```
   unoidl.com.sun.star.table.XCell cc=f.getCellByPosition(0, 0);

   cc.setFormula("gfdgg");
```

Теперь рассмотрим работу со свойствами отдельных ячеек или их диапазонов. Для начала приведем пример объединения ячеек. Для этого у объекта-сслыки на лист вызывается метод getCellRangeByName, которому в качестве параметров передается имя объединяемого диапазона. Метод возвратит объект-ссылку на указанный диапазон, который необходимо привести к типу unoidl.com.sun.star.util.XMergeable, а затем вызвать метод merge с параметром true:

```
unoidl.com.sun.star.table.XCellRange xCellRange =
f.getCellRangeByName("A1:E1");

unoidl.com.sun.star.util.XMergeable xMerge =
(unoidl.com.sun.star.util.XMergeable)xCellRange;

 xMerge.merge(true);
```

Установка выравнивания, цвета фона, параметров шрифта и т. д. конкретной ячейки или диапазона ячеек осуществляется следующим образом. Объект-ссылка на диапазон ячеек приводится к типу

unoidl.com.sun.star.beans.XPropertySet, а затем с помощью вызовов метода setPropertyValue устанавливаются требуемые свойства. Первый параметр метода – имя свойства, второй – устанавливаемое значение.

```
unoidl.com.sun.star.beans.XPropertySet xPropSet =
(unoidl.com.sun.star.beans.XPropertySet)xCellRange;
        xPropSet.setPropertyValue("CharHeight", new uno.Any((Single)14.0));
        xPropSet.setPropertyValue("CharFontName", new
 uno.Any((String)"Times New Roman"));
        xPropSet.setPropertyValue("HoriJustify", new uno.Any((Int32)(2)));
        xPropSet.setPropertyValue("VertJustify", new uno.Any((Int32)(2)));
        xPropSet.setPropertyValue("CellBackColor", new
 uno.Any((Int32)0x98CCFF));
```

Установка свойств столбца осуществляется похожим образом:

```
            xCellRange = f.getCellRangeByName("B1");
        unoidl.com.sun.star.table.XColumnRowRange xColRowRange =
            (OOo.table.XColumnRowRange)xCellRange;
        unoidl.com.sun.star.table.XTableColumns xColumns =
            xColRowRange.getColumns();

        uno.Any aColumnObj = xColumns.getByIndex(0);
        xPropSet                                                    =
(unoidl.com.sun.star.beans.XPropertySet)aColumnObj.Value;
        xPropSet.setPropertyValue("Width", new uno.Any((Int32)300));
        unoidl.com.sun.star.container.XNamed xNamed =
            (unoidl.com.sun.star.container.XNamed)aColumnObj.Value;
```

С установкой границы ячейки все несколько сложнее. Во-первых, необходимо создать один или несколько объектов типа BorderLine, отвечающие за параметры линии, затем создается объект типа TableBorder, свойствам VerticalLine, LeftLine, HorizontalLine, RightLine, TopLine, BottomLine которого присваиваются ссылки на объекты BorderLine, указав дополнительно, какие линии необходимо отобразить. В конце свойству диапозона ячеек TableBorder присваивается ссылка на объект типа TableBorder:

```
BorderLine theLine = new BorderLine();
        theLine.Color = 0x9911FF;
                theLine.OuterLineWidth = 7;
        theLine.InnerLineWidth = 10;

        TableBorder bord = new TableBorder();
        bord.VerticalLine = bord.HorizontalLine = bord.LeftLine = bord.RightLine
 = bord.TopLine = bord.BottomLine = theLine;

        bord.IsVerticalLineValid = bord.IsHorizontalLineValid = bord.IsLeftLineValid = bord.IsRightLineValid =bord.IsTopLineValid = bord.IsBottomLineValid = true;

        xPropSet.setPropertyValue("TableBorder", new uno.Any(typeof(TableBorder), bord));
```

Теперь рассмотрим, как сохранить документ и завершить работу с пакетом. Для сохранения документа необходимо вызвать метод storeToURL, передав ему в качестве параметра имя файла в преобразованном формате. Преобразование заключается в замене слешей и добавлении ключевого слова file перед именем:

```
((XStorable)xComponent).storeToURL(
“file:///”+FileName.Replace(@"\", "/"),
new unoidl.com.sun.star.beans.PropertyValue[0]);
```

После сохранения закрыть OpenOffice можно просто освободив память:

```
xComponent.dispose();
```

Мы рассмотрели работу с OpenOffice с использованием библиотек [17], теперь перейдем к рассмотрению второго метода [10].

Сначала рассмотрим теоретические аспекты данной технологии. Механизм отражения позволяет получать объекты типа Type, описывающие сборки, модули и типы. Эту технологию можно использовать для того, чтобы динамически создать экземпляр типа, привязать тип к существующему объекту, получить тип из

существующего объекта и вызвать его методы или получить доступ к его полям и свойствам. При наличии атрибутов отражение обеспечивает доступ и к ним. При работе с OpenOffice данный способ не столь удобен, как вариант с CLI, но свои плюсы в нем также есть. Для использования технологии отражения необходимо подключить только namespace System.Reflection.

Механизм работы с приложением в данном случае следующий: создается COM-объект (com-сервер) OpenOffice в оперативной памяти компьютера. Ваша программа посылает ему команды на выполнение методов или установку/чтение каких-либо свойств объектов, находящихся внутри сервера. Осуществляется это вызовом у объекта, хранящего ссылку на ваш сервер, метода InvokeMember со следующими параметрами:

- строка-название метода или свойства, понятного серверу;
- флаг типа интерпретации первого параметра: строка содержит имя метода или свойства;
- пустая ссылка для работы с OpenOffice (null), а в общем случае может содержать ссылку на дополнительный объект-посредник;
- ссылка на сам com-сервер;
- массив элементов типа object, хранящий параметры метода или значение свойства.

Теперь рассмотрим практический пример работы с модулем calc. Первым шагом подключаем:

```
using System.Reflection;
```

Теперь получим ссылку на ServiceManager, как com-сервер OpenOffice. Для этого совершаем следующие действия. Во-первых, выясняем его тип по идентификатору, а затем загружаем экземпляр в оперативную память:

```
Type tServiceManager =
Type.GetTypeFromProgID("com.sun.star.ServiceManager", true);

object oServiceManager =
          System.Activator.CreateInstance(tServiceManager);
```

Для простоты дальнейшего объяснения заключим вызов метода InvokeMember, описанного выше, в оболочку метода InvokeObj, который будет иметь следующий вид:

```
public static object InvokeObj(object obj, string method, BindingFlags binding,
params object[] par)
      { return obj.GetType().InvokeMember(method, binding, null, obj, par);}
```

Далее для работы с модулями «Calc» или «Writer» нам нужен объект Desktop. Получим ссылку на него, вызвав через технологию отражения соответствующий метод у объекта ServiceMeneger:

```
object oDesktop = InvokeObj(oServiceManager,
            "createinstance",
            BindingFlags.InvokeMethod,
            "com.sun.star.frame.Desktop");
```

Теперь создадим пустой документ calc. Как мы помним из описанного выше, для создания нового документа нам необходим метод loadComponentFromUrl с определенными параметрами. Первым шагом объявим массив для их хранения и заполним его:

```
                Object[] arg = new Object[4];
          arg[0] = "private:factory/scalc";
          arg[1] = "_blank";
          arg[2] = 0;
          arg[3] = new Object[] { };
```

Теперь вызовем метод InvokeObj у Desktop с соответствующими параметрами:

```
          object oComponent =
InvokeObj(oDesktop,"loadComponentFromUrl",BindingFlags.InvokeMethod,ar
g);
```

Теперь обратимся к листам данного документа через вызов метода getSheets у загруженного компонента. Обратите внимание, что, даже если мы не передаем методу ни одного параметра, массив с параметрами мы все-таки должны объявить, пусть даже пустой:

```
arg = new Object[0];
Object oText =
InvokeObj(oComponent,"getSheets",BindingFlags.InvokeMethod,arg);
```

Допустим, мы хотим записать текст в первую ячейку первого листа. Тогда необходимо осуществить два действия. Во-первых, обратиться к листу, в который будем записывать данные, и, во-вторых, получить конкретную ячейку для записи. Сделаем это следующим образом:

```
oText = InvokeObj(oText,
"getByName",
BindingFlags.InvokeMethod,
new Object[1] { "Лист1" }
);

arg = new Object[2];
arg[0] = 0;
arg[1] = 0;

oText = InvokeObj(oText,
  "getCellByPosition",
  BindingFlags.InvokeMethod,
  arg
  );
```

Теперь запишем в эту ячейку текст:

```
Object oText2 = InvokeObj(oText,
"setString",BindingFlags.InvokeMethod,
new Object[1]{"dfgjdkflgjldfkjgldkf"}   );
```

Или считаем текст из этой ячейки:

```
oText = InvokeObj(oText,
```

```
"getString",
BindingFlags.InvokeMethod,
new Object[0]
);
```

Теперь сохраним файл:

```
arg = new Object[2];
arg[0] = “file:///”+FileName.Replace(@"\", "/");
arg[1] = new Object[0] { };

InvokeObj(oComponent,"storeToUrl",BindingFlags.InvokeMethod,arg);
```

Закрыть OpenOffice после работы можно следующим образом:

```
InvokeObj(oComponent, "close", BindingFlags.InvokeMethod, new object[1] {
"true"});
```

Со свойствами ячеек в данном случае все гораздо сложнее. Рассмотрим пример установки границы. Как мы помним, первое, что нам необходимо сделать, создать структуру типа BorderLine, настроить ее параметры, затем структуру типа TableBorder, связать их между собой и привязать к свойству границы требуемой ячейки. Для простоты кода введем три дополнительных метода: установку значения свойства, создание значения свойства и метод создания структуры. С первыми двумя методами все ясно: они принимают в качестве парамтеров ссылку на объект, свойства которого надо изменить, имя свойства и массив значений для свойства.

```
protected object SetProperty(string name, object obj, object[] args)
{
   return obj.GetType().InvokeMember(name, BindingFlags.SetProperty, null,
obj, args);
}

protected object CreatePropertyValue(string name, object obj, object[] args)
{
   return obj.GetType().InvokeObjMember(name, BindingFlags.CreateInstance
| BindingFlags.InvokeMethod | BindingFlags.GetProperty, null, obj, args);
}
```

Метод создания структуры несколько более сложен. В качестве параметров он принимает ссылку на объект ServiceManager, имя структуры, а также массивы имен и значений ее параметров:

```
private object createStruct(object serviceManager, string name,string[] paramNames, object[] values)
{ object oStruct = CreatePropertyValue("Bridge_GetStruct", serviceManager, new object[] { name });
   for (int i = 0; i < parameterNames.Length; i++)
   { SetProperty(parameterNames[i], oStruct, new object[] { values[i] }); }
   return oStruct; }
```

Теперь мы можем перейти непосредственно к коду установки границы диапазона ячеек. Предположим, мы решили установить границы у области A1:B5. Создаем две требуемые структуры:

```
   object oBorderLine = createStruct(oServiceManager,
      "com.sun.star.table.BorderLine",
      new string[] { "Color", "OuterLineWidth", "InnerLineWidth" },
      new object[] { 0x9911FF, 5, 12 });
   object oTableBorder = createStruct(oServiceManager,
      "com.sun.star.table.TableBorder",
      new string[] { "VerticalLine", "HorizontalLine", "LeftLine", "RightLine", "TopLine", "BottomLine",
                  "IsVerticalLineValid", "IsHorizontalLineValid", "IsLeftLineValid", "IsRightLineValid", "IsTopLineValid", "IsBottomLineValid"},
      new object[] { oBorderLine, oBorderLine, oBorderLine, oBorderLine, oBorderLine, oBorderLine,
                  true, true, true, true, true, true});
```

Получаем ссылку на нужный диапазон:

```
      object oRange = InvokeObj(oSheet,
            " getCellRangeByName ",
            BindingFlags.InvokeMethod,
            object[] { “A1:B5” }
            );
```

Устанавливаем границу:

```
   SetProperty("TableBorder", oRange, new object[] { oTableBorder });
```

Как мы видим, метод отражения с точки зрения взаимодействия внутренних объектов OpenOffice ничем не отличается от способа работы через CLI-сборки. В связи с этим работу с модулем writer рекомендуется рассмотреть самостоятельно.

2.2. Работа с MS Excel

Также, как и при работе с OpenOffice, мы можем использовать два метода работы с табличным процессором: либо используя стандартные библиотеки, либо используя технологию отражения. Рассмотрим оба эти варианта, но начнем со стандартных библиотек [16].

Под стандартными библиотеками в данном случае имеются в виду поставляемые с платформой .Net сборки взаимодействия с приложениями Microsoft Office. При использовании .Net сборок нам после добавления ссылки на Microsoft.Office.Interop.Excel (References, вкладка .Net) потребуется ввести алиас пространства имен Excel, что позволит в дальнейшем сократить объем кода. Итак, прописываем:

```
using Excel = Microsoft.Office.Interop.Excel;
```

Сервер Excel оперирует с несколькими десятками объектов, но мы рассмотрим лишь некоторую их часть, которая непосредственно потребуется для организации взаимодействия между пакетом и приложением. Иерархия объектов может быть описана схемой, приведенной на рисунке 2.1.

Как мы видим, основной объект Excel – объект Application. Он представляет собой сам сервер, и именно с его запуска начинается работа:

```
Excel.Application  excelapp = new Excel.Application();
```

Этот объект имеет свойство Workbooks, в котором содержит ссылки на одну или более книг. Сами книги, представленные объектами Workbook, могут содержать одну или более страниц, а

также одну или более диаграмм. Ссылки на страницы содержит свойство Worksheets, на диаграммы – свойство Charts. Объекты типа Worksheet (страницы) содержат объекты ячейки или группы ячеек, ссылки на которые становятся доступными через Range. Далее идут строки и столбцы. Аналогично объект Chart содержит ссылки на серии линий и легенды.

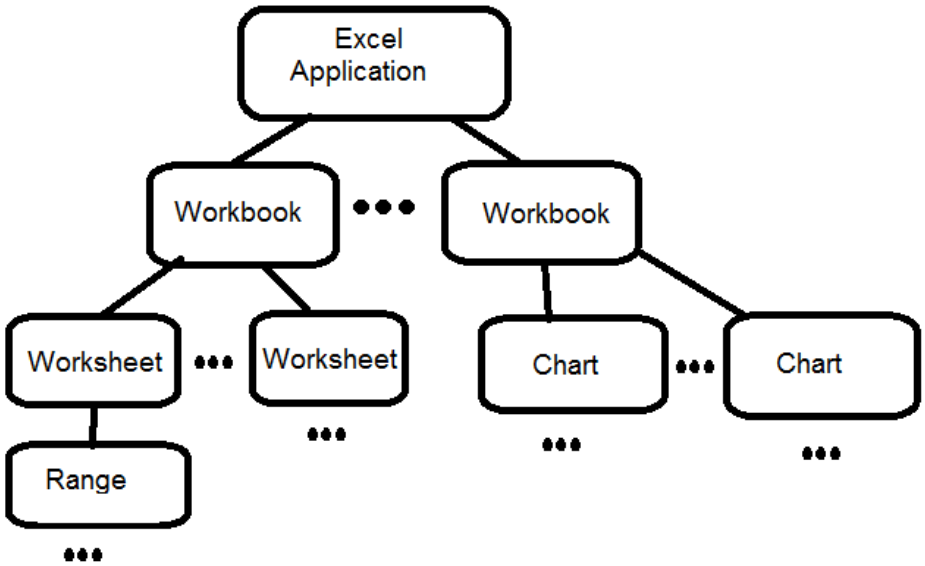

Рисунок 2.1. Иерархия объектов Excel

Ниже приведен простой пример запуска Excel, отображения его на экране:

```
Excel.Application  excelapp = new Excel.Application();
excelapp.Visible=true;
```

и закрытия:

```
excelapp.Quit();
```

Обратите внимание, что этот код откроет вам пустое окно без рабочих книг, так как ни одна из них не была добавлена. Для добавления книги необходимо прописать следующую строчку кода:

```
excelapp.Workbooks.Add(Type.Missing);
```

По умолчанию добавится книга с пятью листами. Для изменения количества листов в добавляемой книге необходимо изменить значение совйства SheetsInNewWorkbook, как показано ниже:

```
excelapp = new Excel.Application();
excelapp.SheetsInNewWorkbook=3;
excelapp.Workbooks.Add(Type.Missing);
excelapp.Visible=true;
```

Однако необходимо учесть, что данное свойство поддерживается не всеми версиями Excel.

Обратите внимание на значение параметра метода Add: Type.Missing. Это специальный тип, определенный в C# и обозначающий, что в метод передается пустое значение параметра. Необходимость введения этого типа объясняется тем, что некоторые методы Excel принимают необязательные параметры, которые не поддерживаются в C#.

Кроме отвечающего за количество листов, рассмотрим еще два важных свойства. Первое из них – TemplatesPath, отвечающее за путь к шаблону, на основе которого нужно создать книгу. Использовать его имеет смысл только для собственных шаблонов. Другое не менее важное свойство – StartupPath, возвращающее путь к папке, содержащей надстройки, выполняемые при запуске Excel.

Теперь рассмотрим закрытие книг. Его можно выполнить либо для конкретной книги, либо для всех книг сразу. В первом случае необходимо у объекта-книги вызывать метод Close с тремя параметрами: SaveChanges типа bool, FileName типа string и RouteWorkbook типа bool. Первый параметр при наличии изменений в книге или наличии ссылок на закрываемую книгу в других открытых определяет, надо ли сохранять изменения. Иначе параметр игнорируется. При значении Type.Missing и наличии изменений или ссылок вызывается диалоговое окно SaveAs. Второй параметр

содержит имя файла для сохранения, а третий отслеживает, должен ли файл быть перенаправлен другому получателю. Пример кода:

```
excelapp.Windows[1].Close(false, Type.Missing, Type.Missing);
```

Для закрытия всех открытых книг у свойства Workbooks вызывается метод Close без параметров:

```
excelapp.Workbooks.Close();
```

Для сохранения какой-либо книги при закрытии всех книг сразу можно обратиться к ее свойству Saved. Если оно стоит в значении false, а свойство сервера DisplayAlerts стоит в значении true, то при закрытии книги через общий метод Close или при выходе из Excel (методом Quit) будет вызвано окошко с запросом на сохранение. Рассмотрим пример, в котором добавим две книги и настроим их свойства таким образом, что при закрытии программы для первой книги запрос на сохранение выполнится, а для второй нет:

```
excelapp = new Excel.Application();
excelapp.SheetsInNewWorkbook=3;
excelapp.Workbooks.Add(Type.Missing);
excelapp.Workbooks.Add(Type.Missing);
excelapp.DisplayAlerts=true;
excelapp.Visible=true;
Excel.Workbooks  excelappworkbooks=excelapp.Workbooks;
Excel.Workbook excelappworkbook=excelappworkbooks[1];
excelappworkbook.Saved=true;
excelappworkbook=excelappworkbooks[2];
excelappworkbook.Saved=false;
excelapp.Quit();
```

Обратите внимание, что в списке книг нумерация начинается с 1.

Подводным камнем использования этого свойства является то, что на некоторых версиях Windows и Office запрос на сохранение может все равно присутствовать, хотя свойство Saved истинно.

Помимо принудительного сохранения, документы можно сохранить не закрывая книгу, используя методы Excel.Workbook Save, SaveAs и SaveCopyAs. Первый метод сохраняет книгу в папке «Мои документы» с именем по умолчанию («Книга1.xls», «Книга2.xls» ...) или в директорию и с именем, под которым документ уже был сохранен. Второй метод позволяет управлять процессом сохранения через свои параметры. Рассмотрим их подробнее. Первый параметр – Filename. Он содержит имя сохраняемого файла. Второй параметр – FileFormat, принимает значение одного из элементов стандартного перечисления и определяет формат сохраняемого файла. Третий параметр – Password, задает пароль доступа к файлу и должен иметь длину не более 15 символов. Далее – WriteResPassword – пароль на доступ для записи. Следующий параметр отвечает за режим доступа к книге: при истинном значении книга будет открыта только для чтения. Далее идет CreateBackup, указывающий, создавать ли резервную копию файла. Параметры AccessMode и ConflictResolution отвечают за режим доступа к рабочей книге и способ разрешения конфликтов, а параметр AddToMru определяет, добавлять ли файл в список ранее открытых. Последними идут кодовая страница, направление размещения текста и идентификатор ExcelApplication. Добавим в рассмотренный ранее пример код сохранения книги по умолчанию для первой и расширенного для второй:

```
excelapp = new Excel.Application();
excelapp.SheetsInNewWorkbook=3;
excelapp.Workbooks.Add(Type.Missing);
excelapp.Workbooks.Add(Type.Missing);
excelapp.DisplayAlerts=true;
excelapp.Visible=true;
Excel.Workbooks  excelappworkbooks=excelapp.Workbooks;
Excel.Workbook excelappworkbook=excelappworkbooks[1];
excelappworkbook.Save();
excelappworkbook=excelappworkbooks[2];
```

```
excelappworkbook.SaveAs(@"G:\a.xls",Excel.XlFileFormat.xlExcel9795,
"WWWWW","WWWWW",Type.Missing, Type.Missing,
 Excel.XlSaveAsAccessMode.xlNoChange, Type.Missing,
Type.Missing, Type.Missing, Type.Missing, Type.Missing);
excelapp.Quit();
```

Третий метод сохранения (SaveCopyAs) сохраняет копию рабочей книги в файле. Он полностью аналогичен методу SaveAs при задании всех параметров, кроме имени файла, как Type.Missing:

```
excelappworkbook.SaveCopyAs(@"G:\a1.xls");
```

Мы рассмотрели работу с новыми книгами, теперь обратимся к работе с существующими. Для этого необходимо познакомиться с методом Open набора Excel.Workbooks. Он имеет следующие параметры (перечисляются в порядке их следования в описании метода):

- FileName: содержит имя открываемого файла;
- UpdateLinks: содержит способ обновления ссылок в файле;
- ReadOnly: определяет, открывать ли файл только для чтения;
- Format: определяет формат символа разделителя;
- Password: содержит пароль доступа к файлу (длина не более 15 символов);
- WriteResPassword: содержит пароль на сохранение файла;
- IgnoreReadOnlyRecommended: игнорирование режима «только для чтения»;
- Origin: определяет тип текстового файла;
- Delimiter: определяет свой разделитель (при значении параметра Format = 6);
- Editable: используется для надстроек Excel 4.0;
- Notify: определяет, добавлять ли имя файла в список нотификации файлов;

- Converter: определяет индекс конвертера файла, используемого для открытия файла;
- AddToMRU: определяет, добавлять ли имя файла в список ранее открытых.

Таким образом, открытие сохраненного ранее файла можно выполнить так:

```
excelapp.Workbooks.Open(@"G:\a.xls",
  Type.Missing, Type.Missing, Type.Missing, "WWWWW", "WWWWW",
  Type.Missing, Type.Missing, Type.Missing, Type.Missing, Type.Missing,
  Type.Missing, Type.Missing, Type.Missing, Type.Missing);
```

Теперь рассмотрим вывод информации на конкретный лист в конкретную ячейку. Для получения ссылки на лист у свойства Worksheets можно вызвать метод get_Item, передав в качестве параметра номер требуемого листа. Для доступа к ячейкам используется метод листа get_Range, дающий доступ к диапазону, ограниченному указанной левой верхней и правой нижней ячейками:

```
Excel.Sheets excelsheets=excelappworkbook.Worksheets;
Excel.Worksheet excelworksheet=(Excel.Worksheet)excelsheets.get_Item(1);
Excel.Range   excelcells=excelworksheet.get_Range("A2","D5");
```

Далее через него может быть получен доступ к свойствам Rows и Cells:

```
Excel.Sheets excelsheets=excelappworkbook.Worksheets;
Excel.Worksheet excelworksheet=(Excel.Worksheet)excelsheets.get_Item(1);
excelcells=excelworksheet.get_Range("A2","D5").Cells;
excelcells=excelworksheet.get_Range("A2","D5").Rows;
```

Если мы хотим напрямую обратиться к нескольким столбцам или строкам, то должны работать со свойствами листа Rows и Columns:

```
Excel.Sheets excelsheets=excelappworkbook.Worksheets;
Excel.Worksheet excelworksheet=(Excel.Worksheet)excelsheets.get_Item(1);
```

```
excelcells=(Excel.Range)excelworksheet.Rows["1:5",Type.Missing];
excelcells=(Excel.Range)excelworksheet.Columns["A:D",Type.Missing];
excelcells=(Excel.Range)excelworksheet.Rows["1",Type.Missing];
excelcells=(Excel.Range)excelworksheet.Columns["D",Type.Missing];
```

Помимо прямого задания адреса требуемой ячейки, Excel позволяет задавать адреса, относительные текущей ячейки, через метод get_Offset(x,y) объекта Range. Метод возвращает ссылку на ячейку, находящуюся на заданном расстоянии по строкам и столбцам от текущей. Отсчет начинается с левого верхнего угла. Приведем пример перехода от ячейки B1 к C1:

```
excelcells=excelworksheet.get_Range("B1",Type.Missing);
excelcells=excelcells.get_Offset(0,1);
```

Получив ссылку на диапазон ячеек, мы можем выполнять с ним различные действия: объединять, выделять, делать его активным. Первое действие выполняется вызовом метода Merge у объекта Range:

```
excelcells.Merge(Type.Missing);
```

Выделение осуществляется вызовом метода Select. Обратите внимание, что ссылка на выделенный объект доступна через свойство Selection самого сервера Excel. Приведем пример выделения и объединения диапазона ячеек:

```
excelcells=excelworksheet.get_Range("A1","C5");
excelcells.Select();
((Excel.Range)(excelapp.Selection)).Merge(Type.Missing);
```

Работать со свойством Selection хотя и удобно, но опасно: во время выполнения приложения любой неосторожный щелчок мыши в переделах документа может изменить его значение.

Похоже работает метод Activate, делающий вызвавший его объект активным. Для таких объектов, как и для выделенных в Excel,

есть специальные свойства. Ключевая разница: для выделенных объектов свойство одно, а для активных – разные. К этим свойствам относятся ActiveWorkbook, ActiveSheet, ActiveChart, ActiveCell. Кроме того, если вы выполняете действия с каким-либо объектом, то он автоматически становится активным. Продемонстрируем на примере обращение к активной книге для получения ссылки на ее листы:

```
excelappworkbook=excelappworkbooks[1];
excelappworkbook.Activate();
excelsheets=excelapp.ActiveWorkbook.Worksheets;
```

После того как вы получили ссылку на ячейку, вы можете менять ее свойства и выводить в нее значения. Необходимо обратить внимание, что при работе с Excel для вывода информации средствами C# требуется работать со значением свойства Value2, а не Value объекта Range. Это связано с тем, что в Excel свойство Value параметрическое, а C# не поддерживает свойств с параметрами, кроме индексных, но это не тот случай. Рассмотрим пример вывода матрицы на первый лист книги. Размер 20 на 15, в каждую ячейку выводится ее координата по строке и столбцу. Шрифт – жирный курсив двенадцатого размера, выравнивание как по горизонтали, так и по вертикали, по центру:

```
  excelworksheet=(Excel.Worksheet)excelsheets.get_Item(1);
  excelworksheet.Activate();
  for(int m=1; m < 20; m++)
    for(int n=1; n < 15; n++)
   {
    excelcells = (Excel.Range)excelworksheet.Cells[m,n];
    excelcells.Value2=m.ToString()+" "+n.ToString();
    excelcells.Font.Size=12;
    excelcells.Font.Italic=true;
    excelcells.Font.Bold=true;
    excelcells.HorizontalAlignment=Excel.Constants.xlCenter;
    excelcells.VerticalAlignment=Excel.Constants.xlCenter;
   }
```

Для числовых данных Excel позволяет задавать формат вывода. Необходимо заметить, что перед выводом форматированного числа в ячейку рекомендуется очищать ее содержимое и текущий формат методом Clear:

```
excelcells.Clear();
excelcells.NumberFormat="### ##0,00";
excelcells.Value2=100000.5;
```

Для строковых данных Excel позволяет задавать частичное форматирование:

```
Range rng = excelapp.ActiveCell;
object start=3;
object end=6;

rng.get_Characters(start, end).Font.Bold = false;
rng.get_Characters(start, end).Font.Italic = false;
rng.get_Characters(start, end).Font.Size = 24;
rng.get_Characters(start, end).Font.ColorIndex = 4;
```

Операция чтения осуществляется также доступом к свойству Value2:

```
excelcells=excelworksheet.get_Range("A2",Type.Missing);
Console.WriteLine(Convert.ToString(excelcells.Value2));
```

Для обводки ячеек используется свойство Borders, через которое можно получить доступ к цвету границы, стилю и толщине линий. Цвет может быть выбран как 1 из 56. Его значение хранится в ColorIndex. Стиль линии описывается свойством LineStyle и может принимать одно из следующих значений: xlContinuous, xlDash, xlDashDot, xlDashDotot, xlDot, xlDouble, xlSlantDashDot, xlLineStyleNone. Толщина линии задается свойством Weight: lHairline, xlMedium, xlThick, xlThin.

Рассмотрим пример задания границы объединенного диапазона ячеек:

```
excelcells=excelworksheet.get_Range("B1","C7");
excelcells.Merge(Type.Missing);
excelcells.Borders.ColorIndex=3;
excelcells.Borders.LineStyle=Excel.XlLineStyle.xlContinuous;
excelcells.Borders.Weight=Excel.XlBorderWeight.xlThick;
```

Для того чтобы задать не все границы ячеек, а только некоторые, например, горизонтальные, необходимо вместо excelcells.Borders обратиться к excelcells.Borders[направление]. В качестве направления может быть одно из следующих значений:

- Excel.XlBordersIndex.xlDiagonalDown,
- Excel.XlBordersIndex.xlDiagonalxlDiagonalUp,
- Excel.XlBordersIndex.xlDiagonalUp,
- Excel.XlBordersIndex.xlEdgeBottom,
- Excel.XlBordersIndex.xlEdgeLeft,
- Excel.XlBordersIndex.xlEdgeRight,
- Excel.XlBordersIndex.xlEdgeTop,
- Excel.XlBordersIndex.xlInsideHorizontal,
- Excel.XlBordersIndex.xlInsideVertical.

Для работы с заливкой ячеек требуется обратиться к свойству Interior:

```
excelcells.Interior.ColorIndex=34;
```

Теперь перейдем к более сложному – к работе с диаграммами. В Excel диаграмма создается по числовым данным, размещенным на каком-нибудь листе рабочей книги. Поэтому сначала необходимо получить таблицу с данными. Будем считать, что она расположена в файле "G:\MyChart.xls" на первом листе в диапазоне от A1 до I3 и имеет вид таблицы 2.1.

Таблица 2.1. Данные для диаграммы

	A	B	C	D	E	F	G	H	I
1	Ряды/ Знач.	Знач. 1	Знач. 2	Знач. 3	Знач. 4	Знач. 5	Знач. 6	Знач. 7	Итого
2	Ряд1	1	2	4	5	4	5	4	25
3	Ряд2	3	4	5	6	3	2	4	27

Первым шагом объявим в нашем проекте требуемые для работы переменные:

```
Excel.Application excelapp=null;

Excel.Workbooks   excelappworkbooks=null;
Excel.Workbook    excelappworkbook=null;

Excel.Sheets      excelsheets=null;
Excel.Worksheet   excelworksheet=null;

Excel.Range       excelcells=null;

Excel.Window      excelWindow=null;
```

Первое, что нам необходимо сделать, – открыть книгу и выделить требуемый диапазон данных. Код будет следующим:

```
excelapp = new Excel.Application();
excelapp.Visible=true;
excelappworkbooks=excelapp.Workbooks;
excelappworkbook=excelapp.Workbooks.Open(@"G:\MyChart.xls",
Type.Missing,Type.Missing, Type.Missing, Type.Missing, Type.Missing,
Type.Missing, Type.Missing, Type.Missing,Type.Missing, Type.Missing,
Type.Missing, Type.Missing,Type.Missing, Type.Missing);
excelsheets=excelappworkbook.Worksheets;
excelworksheet=(Excel.Worksheet)excelsheets.get_Item(1);
excelcells=excelworksheet.get_Range("A1","I3");
excelcells.Select();
```

Теперь аналогично добавлению рабочих книг добавляем новую диаграмму, вызвав метод Add у свойства Charts:

```
Excel.Chart excelchart=(Excel.Chart)excelapp.Charts.Add(Type.Missing,
```

```
Type.Missing, Type.Missing, Type.Missing);
```

Так мы создали диаграмму с типом по умолчанию (xlDefaultAutoFormat). Кроме него можно задать и другие. Перечислим некоторые из них:

- xlArea,
- xlBar,
- xlColumn,
- xlLine,
- xlPie,
- xlRadar,
- xlXYScatter,
- xlCombination,
- xl3DArea,
- xl3DBar,
- xl3DColumn,
- xl3DLine,
- xl3DPie,
- xl3DSurface,
- xlConeCol,
- xlDoughnut.

Для того чтобы изменить тип диаграммы, небходимо выделить ее и изменить свойство ChartType:

```
excelchart.Activate();
excelchart.Select(Type.Missing);
excelapp.ActiveChart.ChartType =Excel.XlChartType.xlConeCol;
```

Теперь можно перейти к изменению свойств. Первое, с чем предлагается поработать, – заголовок диаграммы. Его отображение

задается свойством HasTitle, а прочие параметры доступны через свойство ChartTitle:

```
excelapp.ActiveChart.HasTitle=true;
excelapp.ActiveChart.ChartTitle.Text="Тестовая диаграмма";

excelapp.ActiveChart.ChartTitle.Font.Size = 13;
excelapp.ActiveChart.ChartTitle.Font.Color=254;

excelapp.ActiveChart.ChartTitle.Shadow = true;
excelapp.ActiveChart.ChartTitle.Border.LineStyle= Excel.Constants.xlSolid;
```

Похожие свойства есть и у отдельных осей, которых в Excel три. Это ось категорий, ось значений и ось рядов. Для обращения к осям используется метод Axes, вызываемый у свойства ActiveChart. В качестве параметров он принимает тип оси и группу, к которой эта ось относится. Отобразим заголовки у первых двух осей, оставив третью неозаглавленной:

```
((Excel.Axis)excelapp.ActiveChart.Axes(Excel.XlAxisType.xlCategory,
Excel.XlAxisGroup.xlPrimary)).HasTitle = true;
((Excel.Axis)excelapp.ActiveChart.Axes(Excel.XlAxisType.xlCategory,
 Excel.XlAxisGroup.xlPrimary)).AxisTitle.Text = "Категории";

((Excel.Axis)excelapp.ActiveChart.Axes(Excel.XlAxisType.xlValue,
 Excel.XlAxisGroup.xlPrimary)).HasTitle = true;
 ((Excel.Axis)excelapp.ActiveChart.Axes(Excel.XlAxisType.xlValue,
 Excel.XlAxisGroup.xlPrimary)).AxisTitle.Text = "Значения";

((Excel.Axis)excelapp.ActiveChart.Axes(Excel.XlAxisType.xlSeriesAxis,
 Excel.XlAxisGroup.xlPrimary)).HasTitle = false;
```

Кроме отображения заголовка у осей можно корректировать отображение координатной сетки через свойства HasMinorGridlines и HasMajorGridlines. Приведем пример для оси категорий:

```
((Excel.Axis)excelapp.ActiveChart.Axes(Excel.XlAxisType.xlCategory,
    Excel.XlAxisGroup.xlPrimary)).HasMajorGridlines = true;
((Excel.Axis)excelapp.ActiveChart.Axes(Excel.XlAxisType.xlCategory,
```

```
Excel.XlAxisGroup.xlPrimary)).HasMinorGridlines = false;
```

Для работы с легендой используются свойства HasLegend и Legend у ActiveChart. Обратите внимание, что легенда допускает изменение отображения отдельных рядов:

```
  excelapp.ActiveChart.HasLegend = true;
  excelapp.ActiveChart.Legend.Position  =
 Excel.XlLegendPosition.xlLegendPositionLeft;
((Excel.LegendEntry)excelapp.ActiveChart.Legend.LegendEntries(1)).Font.Siz
e=12;
((Excel.LegendEntry)excelapp.ActiveChart.Legend.LegendEntries(2)).Font.Siz
e=13;
```

Теперь настроим параметры отдельных рядов через метод SeriesCollection у ActiveChart:

```
Excel.SeriesCollection seriesCollection=
(Excel.SeriesCollection)excelapp.ActiveChart.SeriesCollection(
Type.Missing);
Excel.Series series = seriesCollection.Item(1);
series.Name="Первый ряд";
```

Для удаления ряда у объекта series необходимо вызвать метод Delete без параметров. Для переименования оси x необходимо обратиться к свойству XValues объекта series:

```
series.XValues= "Значение1; Значение2; Значение3; Значение4;
Значение5; Значение6; Значение7;Итого";
```

Таким образом, мы создали диаграмму на отдельном листе. Для того чтобы локализовать диаграмму в нужном месте первого листа, а также задать ее размер, выполняем такой код:

```
excelapp.ActiveChart.Location(Excel.XlChartLocation.xlLocationAsObject,"Лис
т1");  excelsheets=excelappworkbook.Worksheets;
  excelworksheet=(Excel.Worksheet)excelsheets.get_Item(1);
  excelworksheet.Shapes.Item(1).IncrementLeft(-200);
  excelworksheet.Shapes.Item(1).IncrementTop((float)18.5);
```

```
excelworksheet.Shapes.Item(1).Height=450;
excelworksheet.Shapes.Item(1).Width=450;
```

Для того чтобы поместить диаграмму сразу на нужном нам листе, необходимо немного изменить логику работы, отказавшись от свойства ActiveChart, и вместо свойства Chart использовать ChartObjects. Код добавления диаграммы в данном случае будет следующим:

```
Excel.ChartObjects chartsobjrcts =
(Excel.ChartObjects)excelworksheet.ChartObjects(Type.Missing);
Excel.ChartObject chartsobjrct = chartsobjrcts.Add(10,200,500,400);
excelcells=excelworksheet.get_Range("A1","I3");
Excel.Chart excelchart=chartsobjrct.Chart;
excelchart.SetSourceData(excelcells,Type.Missing);
```

Далее вся работа с диаграммой осуществляется через объект excelchart вместо excelapp.ActiveChart.

Мы рассмотрели работу с Excel с использованием стандартных библиотек. Теперь рассмотрим технологию отражения. Так как логика ее работы ничем не отличается от рассмотренной в разделе про OpenOffice, то здесь приведем лишь примеры кода изменения некоторых ключевых свойств. Не забудьте подключить пространство имен System.Reflection.

Запуск Excel:

```
object oExcel =
Activator.CreateInstance(Type.GetTypeFromProgID("Excel.Application"));
```

Управление видимостью:

```
oExcel.GetType().InvokeMember("Visible", BindingFlags.SetProperty,
                null, oExcel, new object[] { true });
```

Открытие существующего документа:

```
object WorkBooks = oExcel.GetType().InvokeMember("Workbooks", BindingFlags.GetProperty, null, oExcel, null);

object WorkBook = WorkBooks.GetType().InvokeMember("Open", BindingFlags.InvokeMethod, null, WorkBooks, new object[] { @“G:\a1.xls”, true });
```

Создание нового документа:

```
object WorkBooks = oExcel.GetType().InvokeMember("Workbooks", BindingFlags.GetProperty, null, oExcel, null);

object WorkBook = WorkBooks.GetType().InvokeMember("Add", BindingFlags.InvokeMethod, null, WorkBooks, null);
```

Закрытие документа:

```
WorkBook.GetType().InvokeMember("Close", BindingFlags.InvokeMethod, null, WorkBook, new object[] { true });
```

Сохранение документа по умолчанию:

```
WorkBook.GetType().InvokeMember("Save", BindingFlags.InvokeMethod, null, WorkBook, null);
```

Сохранение документа с назначением имени:

```
WorkBook.GetType().InvokeMember("SaveAs", BindingFlags.InvokeMethod, null, WorkBook, new object[] { @“G:\a1.xls” });
```

Установка альбомной ориентации страницы:

```
object PageSetup = WorkSheet.GetType().InvokeMember("PageSetup", BindingFlags.GetProperty, null, WorkSheet, null);

PageSetup.GetType().InvokeMember("Orientation", BindingFlags.SetProperty, null, PageSetup, new object[] { 2 });
```

Установка книжной ориентации страницы:

```
object PageSetup =
```

```
WorkSheet.GetType().InvokeMember("PageSetup",
BindingFlags.GetProperty, null, WorkSheet, null);

PageSetup.GetType().InvokeMember("Orientation", BindingFlags.SetProperty,
null, PageSetup, new object[] { 1 });
```

Получение ссылки на ячейку листа:

```
object WorkSheets = WorkBook.GetType().InvokeMember("Worksheets",
BindingFlags.GetProperty, null, WorkBook, null);

object WorkSheet = WorkSheets.GetType().InvokeMember("Item",
BindingFlags.GetProperty, null, WorkSheets, new object[] { 1 });

object Range =
WorkSheet.GetType().InvokeMember("Range",BindingFlags.GetProperty,null,
WorkSheet,new object[1] { "A1" });
```

Запись данных в ячейку листа:

```
Range.GetType().InvokeMember("Value", BindingFlags.SetProperty, null,
Range, new object[] { ”value” });
```

Чтение данных из ячейки:

```
Range.GetType().InvokeMember("Value", BindingFlags.GetProperty,
        null, Range, null).ToString();
```

Объединение ячеек:

```
Range.GetType().InvokeMember("MergeCells", BindingFlags.SetProperty,
null, Range, new object[] { true });
```

Установка ширины столбцов:

```
object[] args = new object[] { 15 };

Range.GetType().InvokeMember("ColumnWidth", BindingFlags.SetProperty,
null, Range, args);
```

Установка высоты строк:

```
object[] args = new object[] { 15 };
```

```
Range.GetType().InvokeMember("RowHeight",
BindingFlags.SetProperty, null, Range, args);
```

Установка стиля границы ячейки:

```
object[] args = new object[] { 1 };
object Borders = Range.GetType().InvokeMember("Borders",
BindingFlags.GetProperty, null, Range, null);

Borders = Range.GetType().InvokeMember("LineStyle",
BindingFlags.SetProperty, null, Borders, args);
```

Установка горизонтального и вертикального выравнивания:

```
object[] args = new object[] { 1 };
Range.GetType().InvokeMember("VerticalAlignment",
BindingFlags.SetProperty, null, Range, args);
Range.GetType().InvokeMember("HorizontalAlignment",
BindingFlags.SetProperty, null, Range, args);
```

Мы рассмотрели работу с пакетом MS Excel. Теперь перейдем к работе с MS Word. В данном разделе мы рассмотрим только работу со стандартными библиотеками, так как вариант использования отражения по сути своей полностью аналогичен рассмотренным ранее вариантам.

2.3. Работа с MS Word

Мы рассмотрим только вариант работы со стандартными библиотеками [16]. Вариант, использующий отражение, рекомендуется рассмотреть самостоятельно, так как он аналогичен работе с Excel. Более того, код работы со стандартными библиотеками тоже очень похож на рассмотренный ранее, поэтому подробно мы будем останавливаться только на специфических моментах.

При использовании стандартных библиотек мы добавляем в проект ссылку на Microsoft.Office.Interop.Word и вводим алиас пространства имен Word для дальнейшего упрощения кода:

```
using Word = Microsoft.Office.Interop.Word;
```

Так же, как и у Excel, объекты у Word иерархически упорядочены. Основный элемент: объект Application – это COM сервер и оболочка для других объектов. Он может содержать коллекцию под названием Documents, хранящую ссылки на объекты типа Document, каждый объект типа Document будет хранить коллекцию Paragraphs или ссылок на объекты типа Paragraph, Table, Range, Bookmark, Chapter, Word, Sentence и т. д. Работа с объектами осуществляется путем использования их свойств и методов.

Запуск Word аналогичен запуску Excel:

```
Word.Application  wordapp = new Word.Application();
wordapp.Visible = true;
```

Закрытие осуществляется методом Quit, имеющим следующие параметры.

- SaveChanges – определяет, как сохраняет Word измененные документы перед осуществлением выхода. Может быть одна из Word.WdSaveOptions констант: wdDoNotSaveChanges – не сохранять документ, wdPromptToSaveChanges – выдать запрос перед сохранением, wdSaveChanges – сохранить без предупреждения.
- OriginalFormat – необязательный параметр, определяет формат сохранения для документа. Возможна одна из следующих констант: Word.WdOriginalFormat констант: wdOriginalDocumentFormat – в оригинальном формате документа (не изменяя его), wdPromptUser – по выбору пользователя (актуально при открытии документа, а не при его создании, при создании и сохранении окно сохранения документа всегда присутствует при отсутствии заданного имени документа), wdWordDocument – формат .doc.

- RouteDocument – необязательный параметр. При true документ направляется следующему получателю, если документ является attached документом.

Пример кода:

```
Object saveChanges = Word.WdSaveOptions.wdPromptToSaveChanges;
Object originalFormat = Word.WdOriginalFormat.wdWordDocument;
Object routeDocument = Type.Missing;
wordapp.Quit(ref saveChanges,ref originalFormat, ref routeDocument);
wordapp = null;
```

Для создания рабочих документов используется метод Add со следующими параметрами:

- Template – имя шаблона, по которому создается новый документ. Если значение не указано, то используется шаблон Normal.dot.
- NewTemplate – при true новый документ открывается как шаблон. Значение по умолчанию: False.
- DocumentType – тип документа, может принимать одно из следующих значений констант типа word.WdNewDocumentType: wdNewBlankDocument – документ Word (используется по умолчанию); wdNewWebPage – Web-страница; wdNewEmailMessage – электронное сообщение; wdNewXMLDocument – XML документ.
- Visible – видимость документа. При значении true, которое является значением по умолчанию, документ отображается.

Рассмотрим пример создания рабочего документа:

```
wordapp = new Word.Application();
wordapp.Visible=true;
Object template = Type.Missing;
Object newTemplate = false;
Object documentType =
Word.WdNewDocumentType.wdNewBlankDocument;
Object visible = true;
wordapp.Documents.Add(
ref template, ref newTemplate, ref documentType, ref visible);
```

Для сохранения документа можно использовать стандартный метод Save() или расширенный SaveAs().

Рассмотрим параметры метода SaveAs():

- ref fileName – имя файла.
- ref fileFormat – формат сохраняемого файла, одна из констант типа Word.WdSaveFormat. Может иметь значения: wdFormatDocument, wdFormatWebArchive, wdFormatUnicodeText, wdFormatTextLineBreaks, wdFormatRTF, wdFormatText, wdFormatTemplate, wdFormatHTML, wdFormatFilteredHTML, wdFormatEncodedText, wdFormatDOSText, wdFormatDOSTextLineBreaks.
- ref lockComments – при значении true блокируется содержимое поля «Заметки», находящегося на вкладке «Документ» в диалоговом окне «Свойства» меню «Файл».
- ref password – пароль доступа к документу при открытии.
- ref addToRecentFiles – при значении true имя сохраняемого файла добавляется в список недавно открытых файлов в меню «Файл».
- ref writePassword – пароль для внесения изменений в документ.
- ref readOnlyRecommended – при значении true при открытии документа будет отображаться диалоговое окно с рекомендацией открывать документ только для чтения.
- ref embedTrueTypeFonts – при значении true «TrueType» шрифты сохраняются вместе с документом.
- ref saveNativePictureFormat – при значении true сохраняет только родную графику, графику Windows.
- ref saveFormsData – при значении true сохраняет только данные, введенные пользователем в формы.

- ref saveAsAOCELetter – если в документе используется программа доставки электронной почты адресату attached mailer, то при значении true документ-письмо сохраняется.
- ref encoding – кодовая страница, или набор символов, для документов, сохраненных как кодируемые текстовые файлы. По умолчанию – системная кодовая страниц.
- ref insertLineBreaks – если документ сохраняется как текстовый файл, то при значении true разрешается вставка разрывов строк.
- ref allowSubstitutions – если документ сохраняется как текстовый файл, то при значении true Word заменяет некоторые символы текстом. Например, символ авторского права как (c).
- ref lineEnding – если документ сохраняется как текстовый файл, то данный параметр принимает значение одной из констант типа Word.WdLineEndingType: wdCRLF, wdLSPS,wdCROnly, wdLFCR,wdLFOnly – и определяет, какие символы используются для отделения строк друг от друга.
- ref addBiDiMarks – при значении true Word добавляет к файлу символы управления вывода, чтобы сохранить двунаправленное размещение текста в оригинале документа.

Для открытия существующего документа основным методом является метод Open. Рассмотрим его параметры.

- ref FileName – полный путь к открываемому файлу.
- ref ConfirmConversions – при значении true в случае открытия документа не формата Word будет выводиться диалоговое окно конвертирования файла.
- ref ReadOnly – при значении true документ открывается только для чтения.

- ref AddToRecentFiles – при значении true имя открываемого файла добавляется в список недавно открытых файлов в меню «Файл».
- ref PasswordDocument – пароль открываемого документа.
- ref PasswordTemplate – пароль шаблона документа.
- ref Revert – при значении true возможно повторное открытие экземпляра того же документа с потерей изменений в открытом ранее. При значении false новый экземпляр не открывается.
- ref WritePasswordDocument – пароль для сохранения документа.
- ref WritePasswordTemplate – пароль для сохранения шаблона.
- ref Format – формат открытия. Одна из следующих констант типа Word.WdOpenFormat: wdOpenFormatAllWord, wdOpenFormatAuto, wdOpenFormatDocument, wdOpenFormatEncodedText, wdOpenFormatRTF, wdOpenFormatTemplate, wdOpenFormatText, wdOpenFormatUnicodeText или wdOpenFormatWebPages. По умолчанию имеет значении wdOpenFormatAuto.
- ref Encoding – кодовая страница, или набор символов, для просмотра документа.
- ref Visible – при значении true документ открывается как видимый.
- ref OpenAndRepair – при значении true делается попытка восстановить поврежденный документ.
- ref DocumentDirection – направление текста. Одна из констант Word.WdDocumentDirection: WdLeftToRight, WdRightToLeft.
- ref NoEncodingDialog – при значении true подавляется показ диалогового окна Encoding, которое отображается, если кодировка не распознана.

- ref xmlTransform – определяет тип XML-данных при проводимых XML-преобразованиях.

Для работы с текстовой информацией нам необходимо обратиться к параграфам документа Word. Один объект Paragraph представляет один абзац текста. Для манипуляции с параграфами нам понадобятся объекты:

```
Word.Paragraphs wordparagraphs;
Word.Paragraph  wordparagraph;
```

Любой документ Word всегда имеет хотя бы один параграф. Даже если он пуст, то всегда присутствует курсор ввода. Вывод текста выполняется не просто в параграф, а в диапазон параграфа – объект Range. В документе можно определить диапазон, вызвав метод Range с передачей ему начального и конечного значений позиций символов, а затем методом Select выделить его. Пример:

```
Object begin = 0;
Object end   = 5;
Word.Range wordrange = worddocument.Range(ref begin, ref end);
wordrange.Select();
```

Для выделения текста всего документа можно использовать код:

```
Object begin = Type.Missing;
bject end   = Type.Missing;
Word.Range wordrange = worddocument.Range(ref begin, ref end);
wordrange.Select();
```

Объект Range позволяет не только вывести текст, но и изменить его параметры:

```
wordrange.Font.Size=12;
wordrange.Font.Color=Word.WdColor.wdColorRed;
wordrange.Text="вводимый в выделенный участок текст";
```

Кроме того шрифт и другие параметры форматирования текста можно задать как параметры по умолчанию для всего документа:

```
worddocument.Content.Font.Size=25;
worddocument.Content.Font.Bold=1;
worddocument.Content.Font.Underline=
Word.WdUnderline.wdUnderlineSingle;
worddocument.Content.ParagraphFormat.Alignment=
Word.WdParagraphAlignment.wdAlignParagraphCenter;
worddocument.Content.ParagraphFormat.LeftIndent=
 worddocument.Content.Application.CentimetersToPoints((float)2);
worddocument.Content.ParagraphFormat.RightIndent=
 worddocument.Content.Application.CentimetersToPoints((float)1);
```

Кроме работы с параграфами нам необходимо рассмотреть работу с объектом Selection. Он может представлять блок, строку или столбец таблицы, курсор ввода, рисунок, фрейм, выделенный текст или некоторую комбинацию объектов, которые определяются через свойство Type и могут быть следующими:

- wdSelectionBlock,
- wdSelectionRow,
- wdSelectionColumn,
- wdSelectionIP,
- wdSelectionShape,
- wdSelectionInlineShape,
- wdSelectionNormal.

При работе с текстом через этот объект прежде всего требуется не определить, что в данный момент представляет объект Selection, а задать его. Для этого обычно требуется выполнять действия по перемещению курсора. В Word те или иные действия по перемещению курсора выполняют следующие методы. HomeKey([Unit], [Extend]) и EndKey([Unit], [Extend]) аналогичны нажатию клавиш Home и End на клавиатуре. MoveLeft([Unit], [Count], [Extend]), MoveRight([Unit], [Count], [Extend]), MoveUp([Unit],

[Count], [Extend]), MoveDown([Unit], [Count], [Extend]), Метод Move([Unit], [Count]) имитируют действия со стрелками.

Приведем пример использования методов MoveLeft и MoveRight:

```
object unit;
 object count;
 object extend;

 unit = Word.WdUnits.wdStory;
 extend = Word.WdMovementType.wdMove;
 wordapp.Selection.HomeKey(ref unit, ref extend);

 unit = Word.WdUnits.wdCharacter;
 extend = Word.WdMovementType.wdMove;
 count = 13;
 wordapp.Selection.MoveRight(ref unit, ref count,
ref extend);
 unit = Word.WdUnits.wdWord;
 count = 1;
 extend = Word.WdMovementType.wdExtend;
 wordapp.Selection.MoveLeft(ref unit, ref count,
ref extend);
```

Для вывода текста с использованием объекта Selection используется его свойство Text или метод объекта TypeText. Примеры вывода текста приводятся ниже:

```
object unit;
object count;
object extend;

unit = Word.WdUnits.wdStory;
extend = Word.WdMovementType.wdMove;
wordapp.Selection.HomeKey(ref unit, ref extend);
wordapp.Selection.TypeParagraph();
wordapp.Selection.TypeText("Текст  1 абзаца");
wordapp.Selection.TypeParagraph();
wordapp.Selection.HomeKey(ref unit, ref extend);
unit = Word.WdUnits.wdCharacter;
extend = Word.WdMovementType.wdMove;
```

```
count = 6;
wordapp.Selection.MoveRight(ref unit, ref count,
ref extend);
unit = Word.WdUnits.wdWord;
count = 1;
extend = Word.WdMovementType.wdExtend;
wordapp.Selection.MoveRight(ref unit, ref count,
ref extend);
wordapp.Options.Overtype = false;
wordapp.Selection.TypeText("Новый текст");
wordapp.Selection.Text="Еще новый текст";
```

Для работы с таблицами необходимо обратиться к свойству Tables. Для создания таблиц – объектов Table используется метод Add, имеющий следующий набор параметров.

- Word.Range Range – объект Range, задающий место формирования таблицы.
- int NumRows – число строк.
- int NumColumns – число столбцов.
- ref object DefaultTableBehavior – определяет, изменяет ли Word автоматически размеры ячеек в таблицах, чтобы они соответствовали содержанию ячеек. Может быть одной из констант типа Word.WdDefaultTableBehavior: wdWord8TableBehavior (нет) или wdWord9TableBehavior (да). По умолчанию – wdWord8TableBehavior.
- ref object AutoFitBehavior – автоподбор ширины столбцов, одна из констант типа Word.WdAutoFitBehavior: wdAutoFitContent – по содержимому, wdAutoFitFixed – фиксированная или wdAutoFitWindow – по ширине окна. При значении параметра DefaultTableBehavior=wdWord8TableBehavior параметр AutoFitBehavior игнорируется.

Пример создания таблицы:

```
Object start = 0;
Object end = 0;
Word.Range wordrange = worddocument.Range(ref start, ref end);
```

```
Object defaultTableBehavior =
  Word.WdDefaultTableBehavior.wdWord9TableBehavior;

Object autoFitBehavior = Word.WdAutoFitBehavior.wdAutoFitWindow;
Word.Table wordtable = worddocument.Tables.Add(wordrange, 4, 4,
            ref defaultTableBehavior, ref autoFitBehavior);
```

После того как таблица была добавлена в документ, она доступна по ее номеру в порядке добавления:

```
Word.Table wordtable = worddocument.Tables[1];
```

После того как мы добавили таблицу, можно изменить ее стиль, например:

```
Object style = "Классическая таблица 1";

wordtable.set_Style(ref style);
wordtable.ApplyStyleFirstColumn=true;
wordtable.ApplyStyleHeadingRows=true;
wordtable.ApplyStyleLastRow=false;
wordtable.ApplyStyleLastColumn=false;
```

Таблица состоит из ячеек – объектов Cell, ссылки на которые хранятся в наборе Cells данной таблицы. Следующие строки кода ссылаются на ячейку, расположенную в первой строке и во втором столбце:

```
Word.Range wordcellrange = worddocument.Tables[1].Cell(1, 2).Range;
wordcellrange=wordtable2.Cell(2,3).Range;
```

Для вывода текста в ячейку достаточно свойству Text объекта Word.Range присвоить значение типа string.

```
wordcellrange.Text="Строка для вывода";
```

В таблице и отдельно в ячейках можно изменять как параметры шрифта:

```
wordcellrange.Font.Size=(m+1)*5;
wordcellrange.Font.ColorIndex=Word.WdColorIndex.wdBlue;
wordcellrange.Font.Color=Word.WdColor.wdColorDarkRed;
```

так и любые другие параметры, например обводку:

```
wordcellrange.Borders[Word.WdBorderType.wdBorderBottom].LineStyle =
  Word.WdLineStyle.wdLineStyleTriple;
wordcellrange.Borders[Word.WdBorderType.wdBorderLeft].Color =
  Word.WdColor.wdColorDarkYellow;

wordcellrange.Borders[WdBorderType.wdBorderRight].ColorIndex =
  Word.WdColorIndex.wdBlue;
wordcellrange.Borders[Word.WdBorderType.wdBorderRight].LineWidth =
  Word.WdLineWidth.wdLineWidth025pt;
```

или заливку ячеек:

```
wordcellrange.Shading.ForegroundPatternColor=
                Word.WdColor.wdColorDarkYellow;
wordcellrange.Shading.BackgroundPatternColorIndex=
 Word.WdColorIndex.wdGreen;
```

а также выравнивание:

```
wordcellrange.ParagraphFormat.Alignment=
  Word.WdParagraphAlignment.wdAlignParagraphCenter;
```

Объединение ячеек таблиц выполняется следующим образом:

```
   object begCell = wordtable.Cell(1, 1).Range.Start;
   object endCell =  wordtable.Cell(2, 1).Range.End;
   Word.Range wordcellrange = worddocument.Range(ref begCell, ref
endCell);
   wordcellrange.Select();
   wordapp.Selection.Cells.Merge();
```

Таким образом, мы рассмотрели основные методы, которые потребуются для формирования отчетов в формате Word.

2.4. Работа с форматом PDF

Данная аббревиатура расшифровывается как Portable Document Format. PDF представляет собой кроссплатформенный формат электронных документов, созданный фирмой Adobe Systems с использованием ряда возможностей языка PostScript и предназначенный, в первую очередь, для представления в электронном виде полиграфической продукции. Просмотр документов, созданных в этом формате, осуществляется через специальную программу Adobe Reader, а также программы сторонних разработчиков, которые являются официально бесплатными. С 1 июля 2008 года формат PDF является открытым стандартом ISO 32000.

Функциональные возможности формата достаточно богаты. Он позволяет внедрять в документ необходимые шрифты, изображения, формы, а также мультимедиа-вставки. Имеет собственные технические форматы для полиграфии и позволяет использовать механизмы электронных подписей для защиты и проверки подлинности документов.

Так как любые внедренные объекты могут существенно увеличить объем документа, то рекомендуется использовать векторную графику и «безопасные» шрифты. Всего имеется 14 таких шрифтов. Это шрифты семейства Times: обычный, курсив, полужирный и полужирный курсив; семейства Courier: обычный, наклонный, полужирный и полужирный наклонный; семейства Helvetica: обычный, наклонный, полужирный и полужирный наклонный; а также Symbol и Zapf Dingbats. Эти шрифты будут корректно отображаться любыми программами без необходимости их внедрения. Если используемый в документе шрифт не был внедрен и отсутствует в системе, то это может повлечь ошибки отображения.

В первое время своего существования данный формат был крайне непопулярен, так как, во-первых, программное обеспечение от Adobe для чтения и создания его было платным. Во-вторых, в нем отсутствовала поддержка внешних ссылок, что делало его практически бесполезным в сети Интернет. В-третьих, создаваемые документы были большего размера по сравнению с обычным текстом, что означало более длительную загрузку и отображение с заметными задержками. После выпуска бесплатного программного обеспечения популярность формата стала заметно возрастать, и в настоящее время появилось множество задач, связанных с обработкой этого формата. Это создание и изменение pdf-документов, печать, заполнение полей форм и т. д.

В данном разделе нам предстоит рассмотреть функциональные возможности одной из наиболее популярных библиотек для работы с форматом pdf на языке C#. Речь пойдет об iTextSharp, позволяющей создавать pdf-документы и манипулировать ими. Эта бесплатная библиотека является наиболее упоминаемой в Интернет-сообществе [12]. Ее функционал позволяет осуществлять передачу pdf в браузер, генерировать динамические документы из xml-файлов и баз данных, манипулировать страницами pdf-документа, автоматизировать заполнение форм и добавлять цифровую подпись в документ. Единственная задача, реализация которой в данной библиотеке практически не поддерживается, – извлечение текста из pdf-документа.

Начнем с создания документов. Первое, что необходимо сделать, подключить к проекту библиотеку itextsharp.dll и подключить следующие пространства имен:

```
using iTextSharp.text;
using iTextSharp;
using System.IO;
using iTextSharp.text.pdf;
```

Теперь необходимо обеспечить работу с документом на логическом и физическом уровне. В данном случае физический уровень представляет собой файловый поток записи на диск, а логический уровень – связанный с потоком объект, позволяющий манипулировать содержимым документа. Реализуем это следующим образом:

```
var doc = new Document();
 PdfWriter.GetInstance(doc, new FileStream(@"G:\Document.pdf",
 FileMode.Create));  doc.Open();
```

Документ в pdf-формате в самом простом варианте может содержать главы, абзацы, таблицы и рисунки. Мы рассмотрим работу именно с этими объектами. Глава начинается с новой страницы. Может иметь заголовок, номер и закладку. Заголовок может быть представлен просто строкой текста, тогда никакие его параметры (как цвет, шрифт, выравнивание) не будут доступны для редактирования. Или же можно представить заголовок через объект класса параграф. Тогда можно будет менять шрифт, размер, выравнивание и т. д. Поэтому рассмотрим именно создание главы через объект «параграф». Для начала нам необходимо создать объект типа «фраза», который представляет собой текст параграфа. Затем создаем сам параграф, на основе него – главу. Последним шагом помещаем созданную главу в документ, предварительно задав ей закладку. Помещение любого объекта в документ осуществляется вызовом метода Add:

```
iTextSharp.text.Phrase j = new Phrase("My first chapter", new
iTextSharp.text.Font(iTextSharp.text.Font.FontFamily.COURIER, 11,
iTextSharp.text.Font.BOLDITALIC, new BaseColor(Color.Green)));
Paragraph a1 = new Paragraph(j);
Chapter a = new Chapter(a1,1);
a.BookmarkTitle = "My_Zakladka";  doc.Add(a);
```

Параграфы можно размещать просто в тексте документа. Рассмотрим добавление параграфа, содержащего две строки. Первая строка зеленого цвета, шрифт Courier, жирный курсив. Вторая строка красного цвета, шрифт Times, обычный. Выравнивание абзаца – по центру, отступ после – 5 пикселей:

```
iTextSharp.text.Phrase j = new Phrase("My first paragraf", new
 iTextSharp.text.Font(iTextSharp.text.Font.FontFamily.COURIER, 11,
iTextSharp.text.Font.BOLDITALIC, new BaseColor(Color.Green)));
Paragraph a1 = new Paragraph(j);
a1.Add(Environment.NewLine);
a1.Add(new Phrase("New line in paragraf", new
iTextSharp.text.Font(iTextSharp.text.Font.FontFamily.TIMES_ROMAN, 11,
iTextSharp.text.Font.NORMAL, new BaseColor(Color.Red))));
a1.Alignment = Element.ALIGN_CENTER;
a1.SpacingAfter = 5;
doc.Add(a1);
```

При работе с текстом у вас могут возникнуть проблемы с кириллицей. Для их решения необходимо подгрузить в ваш документ файл, содержащий шрифт, и использовать примерно так:

```
BaseFont baseFont = BaseFont.CreateFont(@"G:\ARIAL.TTF",
BaseFont.IDENTITY_H, BaseFont.NOT_EMBEDDED);
iTextSharp.text.Phrase j = new Phrase("Вторая глава",
new iTextSharp.text.Font(baseFont, 11,
iTextSharp.text.Font.BOLDITALIC, new BaseColor(Color.Green)));
Paragraph a1 = new Paragraph(j);
Chapter a = new Chapter(a1, 2);
doc.Add(a);
```

Кроме текста в документ можно помещать изображения:

```
iTextSharp.text.Image j =
iTextSharp.text.Image.GetInstance(@"G:/images.jpg");
 j.Alignment = Element.ALIGN_CENTER;
 doc.Add(j);
```

Теперь рассмотрим работу с таблицами, представленными объектом класса PdfPTable. Конструктор в качестве параметров принимает количество столбцов в будущей таблице, дальше идет

последовательное построчное добавление ячеек, представленных объектами класса PdfPCell. Ячейка может быть создана на основе фразы или картинки. Добавление ее в талицу осуществляется вызовом метода AddCell. Причем этот метод может в качестве параметра принимать простую строку, тогда на ее основе автоматически создастся и добавится ячейка с параметрами по умолчанию. Для ячеек, созданных на основе фразы, параметры (например, шрифт) можно задавать самостоятельно. Из ключевых свойств объекта ячейки необходимо выделить BackgroundColor, Padding, HorizontalAlignment, VerticalAlignment, Colspan и Rowspan. Первое отвечает за цвет заливки ячейки. Второе – за отступы содержимого от границ, третье и четвертое – за вертикальное и горизонтальное выравнивание соответственно. С помощью пятого и шестого свойств происходит объединение ячеек, также как в талицах на языке HTML. Colspan отвечает за то, на сколько ячеек должна расшириться данная по горизонтали, а Rowspan – по вертикали. Например, для объединения трех столбцов в один необходимо поставить свойство Colspan у первой ячейки в значение 3. Ниже приведен пример добавления таблицы в документ:

```
PdfPTable table = new PdfPTable(3);
PdfPCell cell = new PdfPCell(new Phrase("Simple table",
new     iTextSharp.text.Font(iTextSharp.text.Font.FontFamily.TIMES_ROMAN,
16,             iTextSharp.text.Font.NORMAL, new BaseColor(Color.Orange))));
cell.BackgroundColor = new BaseColor(Color.Yellow);
cell.Padding = 5;
cell.Colspan = 3;
cell.HorizontalAlignment = Element.ALIGN_CENTER;
table.AddCell(cell);

PdfPCell cell1 = new PdfPCell(new Phrase("11"));
cell1.Rowspan = 2;
table.AddCell(cell1);

table.AddCell("Col 2 Row 1");
table.AddCell("Col 3 Row 1");
```

```
table.AddCell("Col 2 Row 2");
table.AddCell("Col 3 Row 2");

jpg = iTextSharp.text.Image.GetInstance(@"G:/left.jpg");
cell = new PdfPCell(jpg);
cell.Padding = 5;
cell.HorizontalAlignment = PdfPCell.ALIGN_LEFT;
table.AddCell(cell);

cell = new PdfPCell(new Phrase("Col 2 Row 3"));
cell.VerticalAlignment = PdfPCell.ALIGN_MIDDLE;
cell.HorizontalAlignment = PdfPCell.ALIGN_CENTER;
table.AddCell(cell);

jpg = iTextSharp.text.Image.GetInstance(@"G:/right.jpg");
cell = new PdfPCell(jpg);
cell.Padding = 5;
cell.HorizontalAlignment = PdfPCell.ALIGN_RIGHT;
table.AddCell(cell);
doc.Add(table);
```

После того как вы заполнили содержимое документа, необходимо сохранить его:

```
doc.Close();
```

Мы рассмотрели создание документа с нуля. Теперь рассмотрим манипуляции с готовыми документами и начнем с частичного копирования исходного документа в другой. Допустим, нам необходимо сохранить вторую страницу в отдельный файл. Здесь мы будем работать с двумя классами: PdfReader и PdfCopy. Первый будет считывать информацию, второй будет ее записывать. Алгоритм действий в данном случае следующий: открыть копируемый документ для чтения:

```
PdfReader reader = new PdfReader(@"G:\Document.pdf");
```

установить аналогичный размер страниц:

```
iTextSharp.text.Document pdfDoc =
 new iTextSharp.text.Document(reader.GetPageSizeWithRotation(1));
```

создать файл для записи на основе исходного:

```
PdfCopy writer = new PdfCopy(pdfDoc, new
 FileStream(@"G:\Document2.pdf",FileMode.OpenOrCreate,
FileAccess.Write));
pdfDoc.Open();
```

задать параметры, аналогичные параметрам копируемой страницы:

```
 pdfDoc.SetPageSize(reader.GetPageSizeWithRotation(2));
```

добавить страницу в новый файл и сохранить его:

```
pdfDoc.NewPage();
iTextSharp.text.pdf.PdfImportedPage page =
writer.GetImportedPage(reader, currentPage);

writer.AddPage(page);
pdfDoc.Close();
```

Если нам необходимо скопировать со второй страницы до конца, то код будет таким:

```
pdfDoc.Open();
for (int k = 2; k <= reader.NumberOfPages; k++)
{int currentPage = k;
pdfDoc.SetPageSize(reader.GetPageSizeWithRotation(currentPage));
pdfDoc.NewPage();
iTextSharp.text.pdf.PdfImportedPage page =
 writer.GetImportedPage(reader, currentPage);
writer.AddPage(page);}
pdfDoc.Close();
```

Эта схема подойдет для решения задачи соединения нескольких документов в один.

Далее рассмотрим вопрос вывода созданного документа на печать или на экран. К сожалению, функционал самой iTextSharp в

этом плане ограничен. Поэтому будем действовать напрямую через программу AcobatReader. Мы запустим процесс этой программы с определенными аргументами. Для начала подключаем пространство имен System.Diagnostics. Затем запускаем процесс-AcrobatReader с аругементами /h – спрятать окно программы, /p – вывод сразу на печать и s – имя файла для печати. Обратите внимание, что путь к файлу acrord32.exe может быть другим на вашем компьютере:

```
System.Diagnostics.Process command = new System.Diagnostics.Process();
command.StartInfo.FileName = @"C:\Program Files\Adobe\Reader 10.0\
Reader\acrord32.exe";
command.StartInfo.Arguments = "/h /p " + @"G:\Document.pdf";
```

Для вывода на печать отключаем показ сообщений приложением:

```
command.StartInfo.UseShellExecute = false;
command.StartInfo.CreateNoWindow = true;
```

И осуществляем запуск:

```
command.Start();
```

После печати документа необходимо завершить процесс. Величина параметра в методе WaitForExit зависит от величины печатаемого файла и определяется эмпирически. Это время, за которое процесс должен успеть передать весь файл принтеру. Код выхода будет следующим:

```
command.WaitForExit(31000);
if (!command.HasExited)
command.Kill();
else  command.Close();
```

Теперь рассмотрим задачу заполнения форм в pdf-документе. Пусть у нас есть документ, содержащий следующую форму (рисунок 2.2.). Нам надо автоматически вывести информацию в ее поля.

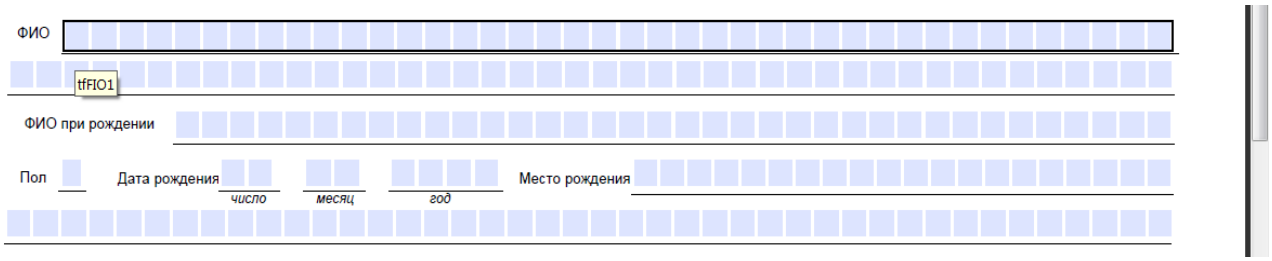

Рисунок 2.2. Pdf-форма

Заметим, что каждое поле имеет имя. Соответственно, обращаться к ним достаточно легко. Алгоритм работы в данном случае крайне простой: открываем исходный файл, считываем его поля, заполняем их, открываем файл для записи, копируем содержимое первого файла с заполненными полями. Работать будем с объектами классов PdfReader (объект pdfr), PdfStamper (объект pdfs) и AcroFields (объект pdfFormFields). Пусть в переменной Fish11 хранится имя файла, содержащего форму для заполнения, а temp11 – имя файла для промежуточного хранения.

Сначала загружаем файл с полями:

```
pdfr = new PdfReader(Fish11);
```

Затем создаем его копию:

```
pdfs = new PdfStamper(pdfr, new FileStream(temp11, FileMode.Create));
```

Выгружаем поля:

```
pdfFormFields = pdfs.AcroFields;
```

Записываем в поле данные:

```
pdfFormFields.SetField("tfFIO1", «Иванов»);
```

Оставляя поля открытыми для редактирования, сохраняем документ:

```
pdfs.FormFlattening = false; pdfs.Close();
```

Именно класс PdfStamper позволяет нам работать с формами. Рассмотрим пример его использования для добавления в наш файл одного текстового поля и одной стандартной кнопки. Сначала через объект нашего класса создаем копию документа:

```
PdfReader reader = new PdfReader(@"G:\Document.pdf");
int n = reader.NumberOfPages;
PdfStamper stamper  =
new PdfStamper(reader, new FileStream(@"G:\Document3.pdf",
FileMode.Create));
```

Теперь создаем объект-кнопку с надписью «Save»:

```
PushbuttonField sav =
new PushbuttonField(stamper.Writer,
new iTextSharp.text.Rectangle(136, 10, 216, 30), "Save");
sav.BorderColor=BaseColor.BLACK;
sav.Text="Save";
sav.TextColor=BaseColor.RED;
sav.Layout=PushbuttonField.LAYOUT_LABEL_ONLY;
sav.Rotation=0;
```

и текстовое поле:

```
PdfFormField ff = PdfFormField.CreateTextField(stamper.Writer, false, false,
0);
ff.FieldName=("name");
ff.SetWidget(new iTextSharp.text.Rectangle(336, 10, 416, 30),
PdfAnnotation.HIGHLIGHT_NONE);
ff.Quadding=(PdfFormField.Q_RIGHT);
```

Назначаем кнопке функциональность:

```
PdfAnnotation saveAsButton = sav.Field;
PdfAnnotation Pole =ff;
saveAsButton.Action=PdfAction.JavaScript(
"app.execMenuItem('SaveAs')", stamper.Writer);
```

и размещаем все на первой странице:

```
stamper.AddAnnotation(saveAsButton, 1);
```

```
stamper.AddAnnotation(Pole, 1);
```

Затем сохраняем документ:

```
stamper.Close();
```

Класс Rectangle в данном случае отвечает за позицию вашего элемента на странице. Заметьте, что начало координат – левый нижний угол.

Похожим образом можно создавать другие виды полей формы (например, checkbox) и кнопки со своей функциональностью. Функции для них на языке JavaScript пишутся в строке и подгружаются как:

```
String script=”тут должен быть текст скрипта”;
writer.addJavaScript(script);
```

Таким образом, мы рассмотрели технологии, позволяющие строить отчеты в различных форматах. Теперь перейдем к средству повышения эффективности программ, строящих отчеты по большим массивам данных.

2.5. Многопоточность

Современные информационные системы имеют дело с огромными и всевозрастающими объемами информации. С ростом объемов данных возрастает и время их обработки. Соответственно, на время манипуляции с информационными массивами (при извлечении их большого количества из баз данных для построения отчетов) возможно кажущееся «повисание» программы – она перестает реагировать на действия пользователя из-за полной занятости работой с данными. Для предотвращения подобных ситуаций рекомендуется использовать технологию многопоточности, которую мы и рассмотрим в данном разделе.

Под потоком будем понимать независимый путь исполнения программы, способный выполняться вместе с другими потоками. Любая программа имеет как минимум один поток, более того, изначально она запускается как единственный поток, который называется «главным». При создании дополнительных потоков программа становится многопоточной.

Для управления потоками в операционной системе есть так называемый «планировщик потоков». Его задача – гарантировать, что активным потокам будет выделено соответствующее время на выполнение, а неактивные (ожидающие или блокированные) не будут потреблять процессорного времени.

Работа планировщика потоков зависит от количества доступных ему процессоров. На однопроцессорных машинах потоки выстраиваются в очередь и получают определенный квант процессорного времени на выполнение. По истечению его происходит переключение на другой активный поток. Размер кванта обычно составляет десятки миллисекунд, так как это намного больше, чем затраты CPU на переключение контекста между потоками. Переключение этого контекста обычно занимает несколько микросекунд. На многопроцессорных машинах к каждому процессору выстраивается своя очередь. Остановка выполнения потока из-за истечения времени или каких-либо внешних факторов называется вытеснением, и обычно потоки не могут контролировать, когда это может случиться.

В плане обработки процессором потоки очень схожи с процессами: между процессами, исполняющимися на одном компьютере, время разделяется так же, как между потоками одного приложения. Основное отличие потоков и процессов заключается в том, что все потоки содержатся в одной области памяти, выделенной процессу, т. е. потоки разделяют между собой одну область памяти.

Благодаря такому принципу работы один поток может предоставлять данные в фоновом режиме, а другой – показывать эти данные по мере их поступления. Процессы же в этом плане полностью изолированы друг от друга.

На платформе .Net основные классы работы с потоками находятся в пространстве имен System.Threading. С их помощью можно явно создавать дополнительные потоки и управлять ими, делая приложение многопоточным. Помимо этого .Net Framework предлагает возможности неявного создания потоков: использование таких инструментов, как BackgroundWorker, пул потоков, потоковый таймер, Remoting-сервер, Web-службы или приложение ASP.NET.

Рассмотрим пример простейшего многопоточного приложения, созданного с помощью явного управления потоками. Первое, что необходимо сделать, это подключить пространство имен:

```
using System.Threading;
```

Теперь опишем метод, который будет выполняться в фоновом потоке. Пусть он будет генерировать случайное число и выводить его на экран:

```
void WriteNum()
 {   Random r=new Random();
   while (true)
     Console.Write(r.Next());   }
```

Теперь в основной программе создаем объект класса Thread, передавая в его конструктор ссылку на наш метод:

```
      Thread t = new Thread(WriteNum);
```

С помощью метода Start запускаем новый поток, а в первом прописываем аналогичный код вывода, но только не числа, а знака «*»:

```
   t.Start();
```

```
while (true)
  Console.Write("*");
```

Выполнив программу, мы увидим примерно следующее:

```
**************************8746575938756578484835875676 7*******************
*45645654654634989089 87****938474356567645375432547 6************435
345435657657567...
```

Многопоточность можно применять в любой программе, но не стоит делать этого без действительной необходимости. Существует две типовых задачи, которые решаются с помощью организации многопоточности. Первая – программа выполняет длительные вычисления. В приложениях Windows Forms получается, что и вычисления, и обработка сигналов от пользователя происходят в одном потоке. Так как вычисления выполняются непрерывно, то сообщения с клавиатуры и мыши не обрабатываются, то есть программа перестает откликаться. Причем, даже если на форме будет демонстрироваться сообщение «Подождите, происходят вычисления», это не гарантирует того, что операционная система не отметит приложение как «не отвечающее», заставив пользователя задуматься.

Вторая типовая задача для многопоточности – задача интенсивных вычислений, когда, например, требуется обработать несколько независимых массивов данных. Особенно эффективно это работает на многопроцессорных машинах: работа распределяется в несколько параллельных очередей, что позволяет сэкономить время. Однако данная модель, как было замечено выше, подойдет только для независимых обработок.

С одной стороны, многопоточность представляет собой весьма привлекательный способ ускорения работы приложения, но, с другой стороны, ее применение существенно усложняет программу. Причем, сложность создают не сами потоки, а организация их взаимодействия.

При недостаточной квалификации разработчиков это может привести к увеличению длительности цикла разработки и появлению трудноуловимых ошибок в приложениях. Еще один недостаток – необходимость выделения дополнительных ресурсов и времени процессора на создание потоков и переключение между ними.

Необходимым условием организации взаимодействия между потоками является синхронизация их выполнения. Под синхронизацией в данном случае понимается согласование действий различных потоков. Согласование может быть либо в самой работе, либо в доступе к ресурсам. Проиллюстрируем оба случая. Пусть один поток выполняет несколько блоков вычислений и при переходе от одного блока к другому ему необходимы данные, обрабатываемые другим потоком. Тогда по окончании блока вычислений один поток должен подождать данные от другого, прежде чем переходить к следующему. Или же пусть один поток должен считать из файла данные, записанные туда другим потоком. Тогда, чтобы чтение не произошло раньше времени, записывающий поток должен получить эксклюзивный доступ к ресурсу, заблокировав его от других.

Рассмотрим инструменты синхронизации потоков, предоставляемые платформой .Net. С точки зрения согласования доступа к ресурсам можно выделить следующие конструкции блокировки ресурсов:

- lock – только один поток может получить доступ к ресурсу или секции кода;
- Mutex – только один поток может получить доступ к ресурсу или секции кода;
- Semaphore – ограничивает число потоков, которые могут получить доступ к ресурсу.

Механизмы блокировки ресурсов являются важнейшими способами организации потоковой безопасности и получения

потокобезопасного кода. Под потокобезопасным кодом будем понимать код, который при любых сценариях многопоточного исполнения не имеет никаких неопределенностей. Если метод при любых сценариях работы многопоточного приложения является потокобезопасным, то он называется реентерабельным. Потенциальная неопределенность в работе в данном случае возникает из-за неизвестности, что поток успеет сделать за отведенный ему период времени, и из-за разделения памяти и ее содержимого (переменных) с другими потоками. При разработке многопоточных приложений на платформе .Net следует учитывать, что типы общего назначения крайне редко являются полностью потокобезопасными. Это обуславливается следующим. Во-первых, учет полной потокобезопасности при разработке может сделать ее очень трудоемкой. Во-вторых, потокобезопасность может отразиться на производительности тогда, когда многопоточность даже не будет использоваться. В-третьих, потокобезопасный тип не гарантирует потокобезопасность использующей ее программы. Поэтому основные типы .NET Framework потокобезопасны только на уровне «доступ только для чтения», и весь труд по обеспечению потокобезопасноти ложится на разработчика. Одна из стратегий решения проблемы – минимизация взаимодействия потоков через минимизацию общих данных. Но она не всегда полностью подходит. Наиболее часто для получения потокобезопасного кода используются блокировки. Рассмотрим пример кода, когда два потока одновременно добавляют объекты в один и тот же список, а затем выводят его элементы:

```
class MyClass
{
  static List <int> list = new List <int>();

static void AddPrintItems()
  {
```

```
  for (int i = 0; i < 10; i++)
    lock (list)
      Add(i);

  int[] items;

  lock (list)
    items = list.ToArray();

  foreach (int i in items)
    Console.WriteLine(i);
}
static void Main()
{
 Thread t1= new Thread(AddPrintItems);
 Thread t2= new Thread(AddPrintItems);
 t1.Start();
 t2.Start();
 } }
```

Блокировка самого объекта-списка обеспечивает полную потокобезопасность в данном сценарии. Обратите внимание, что перед перечислением элементы коллекции выгружаются в массив. Это сделано потому, что операция копирования занимает меньше времени, чем операция перечисления, а на момент вывода элементы списка не должны меняться. Если бы мы работали напрямую с коллекцией, то пришлось бы ее заблокировать на все время вывода.

С точки зрения согласования в работе можно выделить пять основных конструкций блокировки потока:

- Sleep – блокировка потока на указанное время;
- Join – поток ожидает окончания работы другого потока;
- EventWaitHandle – позволяет потоку ожидать сигнала от другого потока;
- Wait – освобождает блокировку объекта и блокирует текущий поток до тех пор, пока объект не получит блокировку снова;
- Pulse – уведомляет поток в очереди готовности об изменении состояния объекта с блокировкой.

Поток называется блокированным, если он остановлен путем использования указанных выше конструкций. У любого потока есть свойство ThreadState, описывающее текущее состояние потока. В период блокировки это свойство имеет значение WaitSleepJoin. Как было сказано выше, блокированные потоки не тратят процессорное время. Снятие блокировки может произойти, если выполнится установленное условие или истечет таймаут. Кроме того, разблокировать поток можно прерыванием или аварийным завершением из другого потока. На платформе .Net для реализации двух последних случаев используются методы Thread.Interrupt и Thread.Abort.

Вызов Interrupt для блокированного потока принудительно разблокирует его и сгенерирует исключение ThreadInterruptedException. Вызов этого метода для неблокированного потока приведет к тому, что поток будет стабильно работать до точки блокировки, попав в которую, сгенерирует описанное выше исключение и выйдет из блокировки. Опасность использования данного метода заключается в том, что любой метод в стеке вызовов может получить запрос на выход из блокировки раньше, чем тот код, которому он действительно предназначался. Если при разработке метода не была учтена возможность такого прерывания, то объекты могут остаться в неработоспособном состоянии или ресурсы не будут полностью освобождены.

Вторым методом аварийного выхода из блокировки является метод Abort. Для блокированного потока он работает аналогично предыдущему, только генерирует исключение ThreadAbortException. Кроме того, даже если исключение произошло в блоке try, оно будет повторно сгенерировано в блоке catch, если только до этого не будет вызван метод Thread.ResetAbort. До его вызова свойство ThreadState будет иметь значение AbortRequested.

Основное отличие между Interrupt и Abort состоит в их работе при вызове для неблокированного потока. Abort в отличие от Interrupt не ждет блокировки потока, а генерирует исключение непосредственно в том месте, где сейчас находится поток.

Если резюмировать все вышесказанное, то можно схематически изобразить следующую диаграмму состояний потока [9] (рисунок 2.3.).

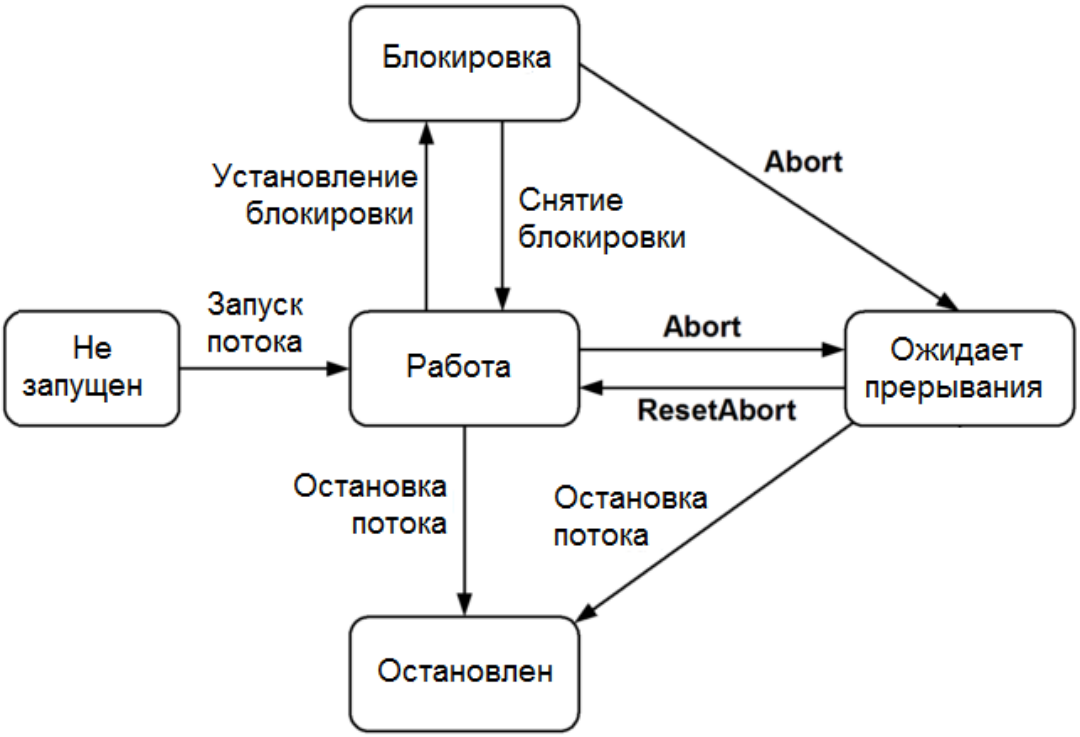

Рисунок 2.3. Диаграмма состояний потока

Рассмотрим более подробно работу с потоками на C#. Как было рассмотрено выше, конструктор класса Thread получает ссылку на метод, то есть объект-делегат типа ThreadStart, который определяется следующим образом:

```
public delegate void ThreadStart();
```

Выполнение потока начинается с вызовом метода Start и продолжается до выхода из исполняемого метода. При этом поток имеет свойство IsAlive, возвращающее true после вызова Start() и до

завершения потока. Если поток закончил свое исполнение, то он не может быть начат заново.

Данный способ подходит для тех случаев, когда методу в дополнительном потоке не требуется никаких внешних данных.

Рассмотрим пример. Пусть нам необходимо сделать так, чтобы один и тот же метод выполнялся в двух потоках, но в первом он выводил бы случайное положительное число, а во втором – случайное отрицательное. Для таких случаев платформа .NET Framework определяет другую версию делегата – ParameterizedThreadStart, которая может принимать один аргумент:

```
public delegate void ParameterizedThreadStart(object obj);
```

Тогда код решения описанной задачи будет таким:

```
class MyClass
{
  static void Main()
  { Thread t = new Thread(Go);
    t.Start(true);
    Go(false);  }
  static void Go(object param)
  {    bool flag = (bool)param;
    Random r=new Random();
      if (flag)   Console.WriteLine(r.Next(5,10));
       else        Console.WriteLine(r.Next(-15,-10));  }}
```

Так как главный поток первым получит «право на выполнение», то в результате работы программы на консоль выведется примерно следующее:

```
-11
6
```

Особенность использования делегата ParameterizedThreadStart состоит в том, что, во-первых, он принимает только один параметр, а во-вторых, перед использованием нужно произвести распаковку

(unboxing) аргумента из типа object к нужному типу. К тому же существует только версия, принимающая единственный аргумент.

Когда вы пишете многопоточное приложение, то его отладка представляет собой более сложную задачу. Одним из методов ее упрощения и средством навигации в массиве потоков является их именование. У каждого потока есть свойство Name, значение которого можно вывести на экран и увидеть в окне Debug – Threads в Microsoft Visual Studio. Имя потоку может быть назначено только один раз. При дальнейшей попытке его изменения будет сгенерировано исключение.

Изменим предыдущий пример, назначив имена потокам:

```
class MyClass
{
  static void Main()
  {  Thread.CurrentThread.Name = "first";
    Thread t = new Thread(Go);
    t.Name=”second”;
    t.Start(true);
    Go(false);  }
  static void Go(object param)
  {    bool flag = (bool)param;
    Random r=new Random();
      if (flag)
        Console.WriteLine(r.Next(5,10) " “+
     Thread.CurrentThread.Name);
       else
        Console.WriteLine(r.Next(-15,-10) "  ”+
     Thread.CurrentThread.Name);  }}
```

Консольный вывод:

```
-12 first    7 second
```

Кроме имени, у потока есть еще одно важное свойство – IsBackground, показывающее, является ли поток основным или фоновым. Причем необходимо пояснить, что фоновые потоки с человеческой и программной точек зрения могут быть разными. Например, мы можем считать, что любой поток, не отвечающий за

интерфейс взаимодействия с пользователем, – фоновый, но для программы он может быть основным. Ключевая разница между фоновыми и основными потоками заключается в том, что, пока выполняется хотя бы один из основных потоков, приложение будет выполняться. Фоновые потоки автоматически завершаются вместе с завершением работы последнего основного.

Следующее свойство, которое нам нужно рассмотреть, – Priority, приоритет потока. Его значение определяет, сколько времени будет выделено данному потоку относительно остальных. Это свойство имеет тип перечисления, состоящий из пяти значений:

```
enum ThreadPriority
{
Lowest, BelowNormal, Normal, AboveNormal, Highest
 }
```

Приоритет играет важную роль, когда несколько потоков выполняются одновременно. Грамотное управление приоритетами потоков позволяет существенно повысить эффективность приложения.

Мы рассмотрели основные свойства потоков, теперь остановимся на вопросе обработки исключительных ситуаций, возникающих при работе потоков. Основное, что здесь необходимо помнить: исключение должно обрабатываться в том потоке, в котором оно было сгенерировано.

Рассмотрим следующий пример:

```
public static void Main()
{  try   {    new Thread(Go).Start();  }
  catch(Exception ex)
  {    Console.WriteLine("Исключение!");  }
  static void Go() { throw null; } }
```

Несмотря на наличие блока try/catch, программа все равно падает, так как каждый поток имеет свой независимый путь исполнения. Корректный код будет таким:

```
public static void Main()
{
  new Thread(Go).Start();
}

static void Go()
{
  try
  {    throw null;   }
catch(Exception ex)   {   ...  } }
```

Более того, обработку исключений необходимо предусматривать во всех потоковых методах, так как начиная с .NET 2.0 необработанное исключение в любом потоке приводит к закрытию всего приложения.

Как было сказано выше, помимо работы с потоками через класс Thread, на платформе .Net существуют так называемые классы-обертки для потоков. Один из них – BackgroundWorker – удобный инструмент для управления фоновыми потоками.

Рассмотрим пример. Пусть нам необходимо обрабатывать большой объем строк из базы данных, выдавая на форму сообщение о состоянии обработки (процент выполненной работы). Допустим, что обработка каждой строки занимает несколько секунд. Для моделирования этого используем два цикла:

```
for (int i = 0; i <= 10000; i++) //счетчик записей
{  for (int l = 0; l <= 1000000000; l++) //долгая обработка
    {
     }
 }
```

Попробуем для начала решить эту задачу без использования дополнительных потоков. Создадим приложение WindowsForms,

разместим на форме метку и кнопку. Сделаем так, что по нажатию кнопки будет происходить наша обработка, а после окончания обработки каждой записи ее номер будет выводиться в label:

```
private void button1_Click(object sender, EventArgs e)
 {
      for (int i = 0; i <= 10000; i++) //счетчик записей
        { for (int l = 0; l <= 1000000000; l++) //долгая обработка
          {
           }
         label1.Text=i.ToString();
        }
  }
```

Запустив приложение и нажав на кнопку, мы видим, что оно «зависает», и в метку выводится только итоговая цифра по окончанию работы (если вы его дождетесь).

Рассмотрим теперь решение этой же задачи с помощью класса BackgroundWorker. Поместим его на нашу форму, перетащив с панели инструментов, и настроим параметры. Важнейшее свойство, которое нам нужно изменить, – WorkerReportsProgress. Если оно не будет в значении true, мы не сможем выводить информацию о состоянии работы. Теперь перейдем к событиям. У BackgroundWorker их три: DoWork, ProgressChanged и RunWorkerCompleted. В обработчике первого события необходимо написать код, который должен выполняться в фоновом потоке. В нашем случае это будет выглядеть так:

```
private void backgroundWorker1_DoWork(object sender,DoWorkEventArgs e)
    {
     for (int i = 0; i <= 10000; i++) //счетчик записей
      {
      for (int l = 0; l <= 1000000000; l++) //долгая обработка
          {
           }
      }
     }
```

Так как этот код будет работать в фоновом потоке, то в нем нельзя напрямую обращаться к метке и менять ее свойства. Для этого у BackgroundWorker есть специальные перегруженные методы – ReportProgress. Первая перегрузка принимает один параметр – процент выполненной работы, а вторая – два: процент выполненной работы и переменную типа object, содержащую дополнительные сведения:

```
ReportProgress(int percent);
ReportProgress(int percent,object UserState);
```

Метод необходимо вызывать тогда, когда надо передать в основной поток сообщение о ходе работы. В нашем случае это будет выглядеть так:

```
private void backgroundWorker1_DoWork(object sender,DoWorkEventArgs e)
    {
      for (int i = 0; i <= 10000; i++) //счетчик записей
        {          for (int l = 0; l <= 1000000000; l++) //долгая обработка
           {           }
           backgroundWorker1.ReportProgress(i);
        }
     }
```

Вызов метода ReportProgress вызывает событие ProgressChanged, в обработчике которого необходимо описать, что делать с информацией о состоянии работы. В нашем случае мы хотим вывести ее в метку, поэтому код будет таким:

```
private void backgroundWorker1_ProgressChanged(object sender,
ProgressChangedEventArgs e)
    {
       label1.Text = e.ProgressPercentage.ToString();
    }
```

Наконец, третье событие RunWorkerCompleted возникает по окончании работы фонового потока. В его обработчике обычно помещают сообщение о том, что все выполнено:

```
private void backgroundWorker1_RunWorkerCompleted(object sender,
RunWorkerCompletedEventArgs e)
    {        MessageBox.Show("Done!");      }
```

Запуск фонового потока осуществляется вызовом метода RunWorkerAsync. В нашем случае мы сделаем это нажатием кнопки:

```
private void button1_Click(object sender, EventArgs e)
    {        backgroundWorker1.RunWorkerAsync();      }
```

Запустив приложение, мы видим результат, требующийся нам изначально: после каждой обработки записи выводится ее номер.

Как мы видим, многопоточность позволяет повысить эффективность приложений, обрабатывающих большие массивы данных. Однако ее применение сопряжено с увеличением сложности приложений, поэтому при ее использовании к вопросам отладки и тестирования нужно подходить тщательнее.

Контрольные вопросы к разделу 2

1. Перечислите основные объекты MS Excel.
2. Перечислите основные объекты MS Word.
3. Перечислите основные объекты OpenOffice.
4. Что такое технология отражения?
5. Перечислите основные объекты, используемые в библиотеке iTextSharp.
6. Что такое поток?
7. Что такое многопоточность?
8. Объясните разницу между фоновыми и основными потоками.
9. Расскажите об основных способах организации потоковой безопасности.
10. Воспроизведите диаграмму состояний потоков.

3. РАБОТА ПРИЛОЖЕНИЙ В СЕТЕВОМ И ЛОКАЛЬНОМ ОКРУЖЕНИЯХ

В данном разделе мы рассмотрим технологии, применяемые для решения задач получения приложением сведений о своем локальном окружении (аппаратная часть и операционная система компьютера), рассмотрим технологии, применяемые для решения задач взаимодействия с такими объектами, как журнал событий Windows, службы Windows, а также работу приложения в сети. К задачам последнего типа относятся:

- организация клиент-серверного взаимодействия;
- работа с электронной почтой;
- работа приложений в сети Интернет;
- взаимодействие с различными сервисами и системами.

В данном разделе мы рассмотрим технологии платформы .Net, используемые для решения всех перечисленных задач.

3.1. Организация клиент-серверного взаимодействия

Самым простым способом организации клиент-серверного взаимодействия на платформе .Net является использование технологии Windows Communication Foundation, или сокращенно WCF. Она представляет собой программный фреймворк, предназначенный для реализации обмена данными между приложениями, входящими в состав .NET Framework. Данная технология делает возможным построение безопасных и надежных транзакционных систем посредством упрощенной унифицированной программной модели межплатформенного взаимодействия. WCF предоставляет единую инфраструктуру разработки, комбинируя функциональность существующих технологий .NET по разработке распределенных приложений. Это позволяет повысить

производительность и снизить затраты на создание безопасных, надежных и транзакционных Web-служб нового поколения.

Организация межсетевого взаимодействия может реализовываться с использованием таких технологий, как COM, DCOM, .NET Remoting, SOAP, TCP/IP, HTTP, именованные каналы и т. д. Теперь программный фреймворк WCF служит для всех них оболочкой и разграничивает то, с чем происходит взаимодействие, и то, как оно осуществляется. Переход от одного протокола к другому (например, от TCP/IP к HTTP) сводится к изменению настроек конфигурации.

Технология WCF была разработана специально для унификации коммуникационных механизмов в рамках одной платформы, причем разработана «с нуля». С помощью нее легко строятся клиент-серверные приложения. В качестве сервера обычно выступают некие службы или сервисы, предоставляющие некоторые операции. Клиенты подключаются к таким серверам и используют предоставляемые возможности. В данном разделе мы рассмотрим азы этой технологии.

В общем случае любое WCF-приложение состоит из трех частей:

- интерфейс и реализация WCF-службы;
- сервер, на котором реализована служба (хост);
- клиент, вызывающий сервер через прокси-класс.

Описанное выше можно представить в виде схемы (рисунок 3.1.):

Рисунок 3.1. Схема WCF-взаимодействия

Как видно из описанного выше, WCF-служба не может существовать самостоятельно, она должна находиться под управлением некоторого процесса Windows, называемого хостовым процессом (хостом). Есть несколько вариантов реализации хостинга:

- автохостинг (т. е. хост-процессом является, к примеру, отдельное консольное или графическое Windows-приложение);
- хостинг в одной из служб Windows;
- хостинг с использованием специальных серверов, таких как IIS (Internet Information Server) или WAS (Windows Activation Services) (англ.).

Выбор варианта реализации хоста обычно зависит от квалификации разработчика, объемов ресурсов и специфики решаемой задачи.

Рассмотрим в качестве примера построение простого WCF-приложения, суть которого – контроль за правильностью введения десятизначных телефонных номеров. В данном случае нам предстоит разработать сервер, предоставляющий операции проверки корректности введенного номера и клиента, который будет обращаться к серверу с «просьбой» проверить выдаваемые им телефоны. Мы будем строить приложение для Framework 4.0, используя протокол подключения HTTP.

Первое, что мы должны сделать, – познакомиться с таким понятием, как «контракт». В технологии WCF это понятие обозначает коллекцию операций, определяющих, как именно сервер общается с внешним миром. Каждая операция – это единица обмена сообщениями, представляющая либо односторонний обмен, либо обмен по схеме запрос-ответ.

Для описания контракта WCF и его операций используется класс ContractDescription. Внутри него каждая операция представляется соответствующим экземпляром класса

OperationDescription. Этот класс описывает различные аспекты операции и содержит коллекцию MessageDescriptions, описывающую сообщения данной операции. Класс ContractDescription обычно создается на основе типа или интерфейса, который описывается с помощью атрибута ServiceContractAttribute, а его методы, которые соответствуют операциям, помечаются атрибутом OperationContractAttribute.

Итак, первый шаг, который мы должны сделать, – это создать интерфейс для нашей операции. Для этого мы создадим новую библиотеку классов, где определим саму службу. В Visual Studio выбираем пункт меню «New Project» и в качестве шаблона проекта берем Class Library. Для простоты дальнейших объяснений определимся с именами. Укажем название проекта «PhoneService» и имя решения «WCFPhoneExample», затем переименуем файл в IPhoneValidator.cs и создадим интерфейс с таким же именем.

Следующим шагом создадим описание операции, добавив сигнатуру метода ValidatePhone, который принимает один аргумент Phone типа string и возвращает булево значение.

```
public interface IPhoneValidator
{
   bool ValidatePhone(string Phone);
}
```

Теперь мы должны превратить это описание в контракт путем использования атрибутов. Для этого сделаем следующее. Во-первых, добавим ссылку на System.ServiceModel (щелкнем правой кнопкой мыши по проекту и выберем пункт Add reference, затем выберем System.ServiceModel из списка). Для упрощения дальнейшего кода пропишем using System.ServiceModel в самом начале файла. Затем перед определением интерфейса поместим атрибут ServiceContract, а

над операцией – атрибут OperationContract. В итоге код интерфейса должен выглядеть так:

```
[ServiceContract]
public interface IPhoneValidator
{

   [OperationContract]
   bool ValidatePhone(string Phone);

}
```

Теперь мы готовы к тому, чтобы написать код реализации самой службы. Для этого создаем новый класс, реализующий интерфейс, в теле метода которого пишем код проверки телефона, используя регулярное выражение.

```
public class PhoneValidator : IPhoneValidator
{
   public bool ValidatePhone(string Phone)
   {
      Console.WriteLine("Validating: {0}", Phone);

      string pattern =@"/^\(\d{3}\) \d{3}-\d{3}-\d{1}$/";
      return Regex.IsMatch(Phone, pattern);

   }
}
```

Завершив описание класса, откомпилируйте полученную библиотеку.

Следующим шагом нам необходимо реализовать хостовый процесс, на котором будет размещаться служба. Для этих целей напишем консольное приложение, добавив новый проект в то же решение, в котором писали библиотеку. Выберем Console Application в качестве шаблона проекта и назовем его «ConsoleHost». Так же, как делали при написании интерфейса, добавьте ссылку на System.ServiceModel, а также на созданную ранее библиотеку. В

методе Main создайте объект класса ServiceHost, как показано в примере ниже. Конструктор этого класса принимает два параметра. Первый – тип-класс, который реализует службу. Второй параметр – адрес хоста. После того, как объект-сервер создан, его надо запустить. Делается это вызовом метода Open.

```
ServiceHost host = new ServiceHost(typeof(PhoneValidator), new Uri("http://localhost:8080/"));

host.Open();
```

Для проверки работы сервера запустите браузер и наберите указанный адрес. Если вы вышли на страницу сервера, то это означает, что процесс запущен и работоспособен. Если вы видите сообщение об ошибке (указанный адрес не найден), то с сервером что-то не то. Единственный момент, который нужно учитывать, заключается в том, что запускать сервер можно, только обладая на это правами (обычно правами администратора). Соответственно, если вы работаете под пользователем, не обладающим этими правами, то программа выдаст исключение. Для того чтобы все-таки запустить сервер, найдите исполняемый файл вашей программы и запустите его с правами администратора.

Обратите внимание, что в данном варианте работы сервер прекращает свое существование сразу после открытия, что совершенно неудобно. Поэтому добавим следующий код:

```
#region Output dispatchers listening
   foreach (Uri uri in host.BaseAddresses)
   {      Console.WriteLine("\t{0}", uri.ToString());   }
   Console.WriteLine();
   Console.WriteLine("Count and list of listening : {0}", host.ChannelDispatchers.Count);
   foreach (System.ServiceModel.Dispatcher.ChannelDispatcher dispatcher in host.ChannelDispatchers)
   {Console.WriteLine("\t{0}, {1}", dispatcher.Listener.Uri.ToString(), dispatcher.BindingName);
```

```
}
Console.WriteLine();
Console.WriteLine("Press <ENTER> to stop the host");
Console.ReadLine();
#endregion
```

Этот код выдает дополнительную информацию о вашем сервере и не останавливает его работу, пока пользователь не нажмет Enter.

Теперь разберемся с таким понятием WCF, как «конечные точки». Конечные точки — это места, где происходит обмен сообщениями (или просто их прием и передача), а также определяются все сведения, необходимые для этого обмена. Служба предоставляет одну или несколько конечных точек, а клиент создает конечную точку, совместимую с одной из конечных точек службы. Таким образом, конечные точки обеспечивают доступ клиентов к функциональным возможностям службы.

Конечная точка состоит из адреса, привязки, контрактов и поведения. Рассмотрим каждую составляющую подробнее.

Адрес конечной точки представлен классом EndpointAddress. Он однозначно определяет ее и указывает потенциальным клиентам на место расположения службы. Объект класса EndpointAddress содержит следующие важные свойства: свойство Uri, представляющее адрес службы, и свойство Identity, представляющее удостоверение безопасности службы и коллекцию необязательных заголовков сообщений.

Привязка конечной точки представлена абстрактным базовым классом Binding. Она задает способ связи клиента с конечной точкой:

- используемый транспортный протокол (например, TCP или HTTP);
- используемую в сообщениях кодировку (например, текст или двоичное кодирование);

- необходимые требования безопасности (например, безопасность сообщений SSL или SOAP).

В большинстве случаев клиенты могут использовать только одну из предусмотренных системой привязок.

Контракты показывают функциональные возможности, предоставляемые клиенту конечной точкой. В контракте задается форма сообщения, операции, которые могут быть вызваны клиентом, тип входных параметров или данных, требуемых для вызова операции, а также тип обработки или ответного сообщения, который может ожидать клиент.

Поведение конечной точки описывает различные аспекты ее работы. Его можно использовать для настройки локального поведения конечной точки службы.

На платформе .Net 4.0 конфигурирование конечных точек можно выполнять автоматически, на основе определенного множества базовых комбинаций свойств конечных точек. Но для лучшего понимания работы конечных точек мы будем конфигурировать их вручную.

Для работы нашей службы мы должны добавить поведение метаданных (Metadata behavior), конечные точки BasicHttpBinding нашей службы и конечные точки MEX, предоставляющие информацию о сервере клиентам.

Для описания поведения метаданных нашего проекта-сервера мы создаем конфигурационный файл app.config и прописываем в нем следующее:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
   <system.serviceModel>
      <behaviors>
         <serviceBehaviors>
            <behavior name="MyBehavior">
               <serviceMetadata httpGetEnabled="True"/>
```

```
            </behavior>
          </serviceBehaviors>
        </behaviors>
    </system.serviceModel>
</configuration>
```

Теперь мы добавим первую конечную точку, используя связь по протоколу http, используя BasicHttpBinding. Под тэгом <system.serviceModel> мы добавим тег <services /> и добавим конечную точку нашей службы, пока не заполняя ее атрибуты. В итоге мы получим следующее:

```
<services>
   <service name="">
      <endpoint
         address=""
         binding=""
         contract="" />
   </service>
</services>
```

Теперь перейдем к заполнению атрибутов. Первый из них name. В нем мы должны указать полное имя реализации нашей службы, то есть класса. В нашем случае это PhoneValidator, но по синтаксическим правилам платформы .Net его полное название включает пространство имен. Поэтому в итоге получаем PhoneService.PhoneValidator.

Следующий атрибут (адрес) мы не обязаны заполнять, так как базовый адрес хоста в нашем коде уже определяет его.

Значение атрибута привязки (binding) — просто basicHttpBinding.

В свойстве contract мы должны указать контракт, которым является наш интерфейс. Как и в случае с атрибутом name, необходимо ввести полное имя, в нашем случае: PhoneService.IPhoneValidator.

Теперь добавим конечную точку MEX, заполнив те же атрибуты (адрес, привязку и контракт). В адресе мы введем просто «mex», так как не можем указать для этой конечной точки тот же адрес, что и для конечной точки BasicHttpBinding. Привязка: mexHttpBinding. Контракт MEX-точки обязательно должен быть IMetadataExchange. Осталось только добавить поведение метаданных. Для этого переходим к тэгу объявления службы, рядом с ее именем добавляем атрибут behaviorConfiguration и указываем имя нашей конфигурации поведения: MyBehavior.

В итоге у вас должен получиться следующий код:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
   <system.serviceModel>
      <services>
         <service                              name="PhoneService.PhoneValidator"
behaviorConfiguration="MyBehavior">
            <endpoint
               address=""
               binding="basicHttpBinding"
               contract="PhoneService.IPhoneValidator" />
            <endpoint
               address="mex"
               binding="mexHttpBinding"
               contract="IMetadataExchange" />
         </service>
      </services>
      <behaviors>
         <serviceBehaviors>
            <behavior name="MyBehavior">
               <serviceMetadata httpGetEnabled="True"/>
            </behavior>
         </serviceBehaviors>
      </behaviors>
   </system.serviceModel>
</configuration>
```

Теперь займемся созданием клиентским приложением, чтобы у нас была возможность проверить работоспособность нашей службы.

Добавим новое консольное приложение в то же решение и назовем его ConsoleClient. Запустим наш хост от имени администратора и перейдем к заключительному шагу: созданию прокси-класса, находящегося между клиентом и службой. Этот класс генерируется для обеспечения интерфейса и не зависит от используемого протокола, что позволяет менять протокол в файле конфигурации без перекомпиляции всего приложения.

Есть два способа создания прокси-класса: либо с помощью средств Visual Studio, либо с помощью специальной утилиты. Второй способ более предпочтителен, поскольку Visual Studio генерирует дополнительные файлы с метаданными, которые могут оказаться разными на разных компьютерах. Поэтому лучше избегать такой ситуации, используя специальную утилиту svcutil.exe.

Утилита svcutil.exe поставляется вместе с Windows SDK. В результате ее работы получается два файла: код, реализующий прокси-класс на языке C#, и файл конфигурации для использования в клиенте.

Для начала найдите файл утилиты на своем компьютере и скопируйте его в папку с проектом ConsoleClient. Теперь запустите командную строку (cmd) и перейдите к месту расположения ConsoleClient, выполнив команду cd с атрибутом /d и путем к файлу (допустим, у вас проект находится на диске G в папке 11, тогда команда будет cd /d G:\11). Далее введите следующую команду:

```
svcutil http://localhost:8080/ /o:ServiceProxy.cs /config:App.Config /n:*,ConsoleClient
```

В качестве первого аргумента команде передается место расположения нашей службы, как указано в хосте (базовый адрес). Второй аргумент – имя выходного файла для создания прокси-класса. Третий аргумент указывает, что мы также хотим обновить конфигурационный файл приложения, а при его отсутствии – создать.

Последний аргумент – пространство имен для прокси-класса, совпадающее с именем приложения.

Выполнив команду, вернитесь в Visual Studio к своему приложению ConsoleClient и включите полученные файлы в проект. Для того чтобы это сделать, найдите в обозревателе решений App.Config и ServiceProxy.cs. Если вы их не видите там, то нажмите на иконку «ShowAllFiles» в верхней части обозревателя решений. Теперь новые файлы отображаются, и вы можете включить их в проект, нажав правой кнопкой мыши и выбрав Include in Project.

Теперь мы можем наконец обратиться к службе, послав запрос от клиента. Переходим к методу Main в классе Program нашего клиентского приложения. Теперь мы имеем доступ к прокси-классу, в данном случае его имя PhoneValidatorClient, то есть имя нашей службы с суффиксом Client.

В теле метода Main запустим цикл, в теле которого будем запрашивать номера телефонов для проверки, если будет введена пустая строка, то приложение закончит работу. Ниже создадим объект прокси-класса и с помощью него вызовем метод проверки телефона, в качестве параметра передав введенное пользователем значение. В итоге код будет выглядеть следующим образом:

```
static void Main(string[] args)
{   while (true)   {
Console.WriteLine("Enter a phone or press [ENTER] to quit...");
      string emailAddress = Console.ReadLine();
      if (string.IsNullOrEmpty(emailAddress))
         return;
      EmailValidatorClient svc = new EmailValidatorClient();
      bool result = svc.ValidateAddress(emailAddress);
      Console.WriteLine("Email address is valid : {0}", result);   }}
```

Мы рассмотрели реализацию простейшего клиент-серверного взаимодействия через http, используя технологию WCF.

3.2. Работа с электронной почтой

Нередко на практике возникают задачи по автоматической обработке электронной почты (прием-отправка). Рассмотрим технологии, используемые для этого. Работа с почтой осуществляется через определенные протоколы передачи данных (наборы соглашений логического уровня об интерфейсах передачи данных). Для отправки сообщений в сетях TCP/IP предназначен сетевой протокол SMTP (англ. Simple Mail Transfer Protocol — простой протокол передачи почты). Он используется для отправки почты от пользователей к серверам и между серверами для дальнейшей пересылки к получателю.

Данный протокол получил распространение в начале 80-х годов. До этого использовался протокол UUCP, основным недостатком которого была необходимость знания полного маршрута от отправителя до получателя и явного указания этого маршрута в адресе получателя либо наличия прямого коммутируемого или постоянного соединения между компьютерами отправителя и получателя.

Сервер SMTP представляет собой конечный автомат с внутренним состоянием. Клиент передает на сервер строку вида: «команда<пробел>параметры<перевод строки>». Сервер отвечает на каждую команду строкой, содержащей код ответа и текстовое сообщение, отделенное пробелом. Код ответа представляет собой число в диапазоне от 100 до 999, представленное в виде строки. Коды имеют следующие значения:

2XX — команда успешно выполнена;

3XX — ожидаются дополнительные данные от клиента;

4XX — временная ошибка, клиент должен произвести следующую попытку через некоторое время;

5XX — неустранимая ошибка.

Текстовая часть ответа носит справочный характер. Общение между клиентом и сервером обычно осуществляется через 25-й порт.

Рассмотрим пример обычной SMTP-сессии. В данном примере C: — клиент, а S: — сервер:

```
S: (ожидает соединения)
C: (подключается к порту 25 сервера)
S:220 mail.MyCompany.ltd ESMTP is glad to see you!//подключение успешно
C:HELO                    //начать работу
S:250 domain name should be qualified //укажите отправителя
C:MAIL FROM: user1name@company.ru //адрес отправителя
S:250 user1name@company.ru sender accepted //адрес принят
C:RCPT TO:user2@company.ltd  // адрес получателя
S:250 user2@company.ltd ok   // адрес принят
C:RCPT TO: user3@company.lttd  //еще адрес получателя
S:550 user3@company.lttd  unknown user account //адреса не существует
C:DATA // передача данных письма
S:354 Enter mail, end with "." on a line by itself //ожидаются данные,
                    //по окончании   ввода ожидается точка.
C:from: user1name@company.ru       //текст письма
C:to: user2@company.ltd                    //текст письма
C:subject: tema                            //текст письма
C:                                         //текст письма
C:Hi!                                      //текст письма
C:.                                        //текст письма
S:250 769947 message accepted for delivery//письмо принято для доставки
C:QUIT //клиент отключается
S:221 mail.company.tld ESMTP closing connection //сервер принял команду
S: (закрывает соединение)
```

В результате такой сессии письмо будет доставлено адресату user2@company.ltd, но не будет доставлено адресату user3@company.lttd, потому что такого адреса не существует.

Рассмотрим пример реализации отсылки электронного сообщения на языке C#. Во-первых, для доступа к необходимым классам подключим следующие пространства имен:

```
using System.Net.Mail;
using System.Net;
using System.Net.Mime;
```

Для доступа к серверу создадим объект класса SmtpClient, передав в качестве параметров в конструктор имя сервера и номер порта.

```
SmtpClient Smtp = new SmtpClient("smtp.mail.ru", 25);
```

Теперь укажем в свойстве Credentials логин-пароль для авторизации:

```
Smtp.Credentials = new NetworkCredential("user", "password");
```

Далее перейдем к формированию самого письма, используя объект класса MailMessage.

```
MailMessage Message = new MailMessage();
Message.From = new MailAddress("userFrom@mail.ru");//от кого
Message.To.Add(new MailAddress("userTo@mail.ru"));//кому
Message.Subject = "Проверка связи";//тема
Message.Body = "Сделано на Си шарп";//текст письма
```

Для прикрепления файла используем объект класса Attachment, добавив его потом в свойство-список объекта MailMessage:

```
string file = "F:\\Privet.doc";
Attachment attach = new Attachment(file, MediaTypeNames.Application.Octet);
Message.Attachments.Add(attach);
```

И собственно отправка письма:

```
Smtp.Send(Message);
```

Мы рассмотрели автоматическую отправку писем, теперь рассмотрим прием. Получение почтовых сообщений с сервера осуществляется посредством протокола POP3 (англ. Post Office Protocol Version 3 — протокол почтового отделения, версия 3). В данном протоколе предусмотрено 3 состояния сеанса:

- авторизация, когда клиент проходит процедуру аутентификации;
- транзакция, когда клиент получает информацию о состоянии почтового ящика, принимает и удаляет почту;

- обновление, когда сервер удаляет выбранные письма и закрывает соединение.

В таблице 3.1 приведены команды, поддерживаемые протоколом POP3.

Таблица 3.1. Команды протокола POP3

Команда	Описание	Аргументы	Ограничения	Возможные ответы
APOP [имя] [digest]	Команда служит для передачи серверу имени пользователя и зашифрованного пароля (digest).	[имя] – строка, указывающая имя почтового ящика. [digest] — хеш-сумма временной метки, связанной с паролем пользователя, вычисленная по алгоритму MD5. Временная метка получается при соединении с сервером.	Ее поддержка не является обязательной.	+OK maildrop has n message; -ERR password suplied for [имя] is incorrect.
USER [имя]	Передает серверу имя пользователя.	[имя] – строка, указывающая имя почтового ящика.	–	+OK name is a valid mailbox; -ERR never heard of mailbox name.
DELE [сообще-ние]	Сервер помечает указанное сообщение для удаления. Такие сообщения реально удаляются только после закрытия транзакции (закрытие происходит по команде QUIT или иногда по истечении времени, установленного сервером).	[сообщение] – номер сообщения.	Доступна после успешной идентификации.	+OK message deleted; -ERR no such message.

Продолжение табл. 3.1.

Команда	Описание	Аргументы	Ограничения	Возможные ответы
PASS [пароль]	Передает серверу пароль почтового ящика.	[пароль] – пароль для почтового ящика.	Работает после успешной передачи имени почтового ящика.	+OK maildrop locked and ready; -ERR invalid password; -ERR unable to lock maildrop.
LIST [сообще-ние]	Если был передан аргумент, сервер выдает информацию об указанном сообщении. Если аргумент не был передан, то сервер выдает информацию обо всех сообщениях, находящихся в почтовом ящике. Сообщения, помеченные для удаления, не перечисляются.	[сообщение] – номер сообщения (необязательный аргумент).	Доступна после успешной идентификации.	+OK scan listing follows; -ERR no such message.
NOOP	Сервер ничего не делает, всегда отвечает положительно. Команда используется для поддержки соединения с сервером при длительном бездействии.	–	Доступна после успешной идентификации.	+OK.
RETR [сообще-ние]	Сервер передает сообщение с указанным номером.	[сообщение] – номер сообщения.	Доступна после успешной идентификации.	+OK message follows; -ERR no such message.

Окончание табл. 3.1.

RSET	Этой командой производится откат транзакций внутри сессии. Например, если пользователь случайно пометил на удаление какие-либо сообщения, он может убрать эти пометки, отправив эту команду.	–	Доступна после успешной идентификации.	+OK.
STAT	Сервер возвращает количество сообщений в почтовом ящике плюс размер, занимаемый этими сообщениями на почтовом ящике.	–	Доступна после успешной идентификации.	+OK [количество] [размер].
TOP [сообще-ние] [коли-чество строк]	Сервер возвращает заголовки указанного сообщения, пустую строку и указанное количество первых строк тела сообщения.	[сообщение] – номер сообщения. [количество строк] – сколько строк нужно вывести.	Доступна после успешной идентификации.	+OK n octets; -ERR no such message.
QUIT	Закрытие соединения	–	–	+OK.

По аналогии с сервером SMTP рассмотрим пример сессии с сервером POP3:

S: <Сервер ожидает входящих соединений на порту 110>
C: <подключается к серверу>
S: +OK POP3 server ready <1896.697170952@dbc.mtview.ca.us>
C: APOP mrose c4c9334bac560ecc979e58001b3e22fb
S: +OK mrose's maildrop has 2 messages (320 octets)
C: STAT

```
S:   +OK 2 320
C:   LIST
S:   +OK 2 messages (320 octets)
S:   1 120
S:   2 200
S:   .
C:   RETR 1
S:   +OK 120 octets
S:   <сервер передает сообщение 1>
S:   .
C:   DELE 1
S:   +OK message 1 deleted
C:   RETR 2
S:   +OK 200 octets
S:   <сервер передает сообщение 2>
S:   .
C:   DELE 2
S:   +OK message 2 deleted
C:   QUIT
S:   +OK dewey POP3 server signing off (maildrop empty)
C:  <закрывает соединение>
S:  <продолжает ждать входящие соединения>
```

Теперь рассмотрим пример реализации получения писем на C#. Для начала подключаем namespace:

```
using System.Net.Mail;
using System.Net;
using System.Net.Mime;
using System.Net.Sockets;
```

Объявим несколько переменных: строку, в которую будем получать ответ сервера, строку для адреса отправителя, строку для темы письма и числовую переменную для хранения общего объема писем.

```
string response=null;
 string from = null;
 string subject = null;
 int totmessages=0;
```

Для работы с сервером нам потребуется объект класса TcpClient, в конструктор которого мы передадим адрес сервера и номер порта.

Так как при создании объекта сразу будет попытка установления соединения с сервером, то заключим код создания объекта в блок обработки возможных исключений.

```
TcpClient mailclient = null;
        try
        {
           mailclient = new TcpClient("pop3.mail.ru", 110);
        }
        catch (SocketException)
        {
           Console.WriteLine("Unable to connect to server");
           Console.ReadLine();
           return;
        }
```

С помощью объекта класса NetworkStream откроем поток обмена сообщения с сервером. Отправлять сообщения серверу мы будем путем записи их в поток методами экземпляра класса StreamWriter, получать сообщения – с помощью объекта класса StreamReader.

```
NetworkStream ns = mailclient.GetStream();
StreamReader sr = new StreamReader(ns, Encoding.GetEncoding("windows-
1251"));
StreamWriter sw = new StreamWriter(ns);
response = sr.ReadLine(); //открываем «диалог»
sw.WriteLine("User " + "username"); //Передаем имя пользователя
sw.Flush();
response = sr.ReadLine();
if (response.Substring(0, 3) == "-ER")//если сервер ответил «ошибка»
  {
Console.WriteLine("Unable to log into server");
Console.ReadLine();
return; // прерываем «диалог»
  }
sw.WriteLine("Pass " + "password"); //Передаем пароль
sw.Flush();
  try {// если авторизация прошла успешно
      response = sr.ReadLine(); //получим ответ от сервера
   }
   catch (IOException)
```

```
 {
 Console.WriteLine("Unable to log into server");
            Console.ReadLine();
            return; //иначе прерываем «диалог»
        }
if (response.Substring(0, 4) == "-ERR")//если сервер ответил «ошибка»
        {
            Console.WriteLine("Unable to log into server");
            Console.ReadLine();
            return; // прерываем «диалог»
        }
sw.WriteLine("stat"); //Посылаем запрос на статистику почтового ящика
        sw.Flush();
        response = sr.ReadLine();
        string[] nummess = response.Split(' ');
        totmessages = Convert.ToInt16(nummess[1]);
        if (totmessages > 0)
        {
       Console.WriteLine("you have " + totmessages + " messages");
        }
        else
        {
            Console.WriteLine("You have no messages");
        }

        for (int i = 1; i <= totmessages; i++)
        {
      sw.WriteLine("top "+i+" 0"); //читаем заголовки каждого письма
            sw.Flush();
            while (true)
            {
                response = sr.ReadLine();
                if (response == ".")
                    break;
                if (response.Length > 4)
                {
                    if (response.Substring(0, 5) == "From:")
                        from = response;
                    if (response.Substring(0, 8) == "Subject:")
                     subject = response;
                }
            }
            Console.WriteLine(i + " " + from + " " + subject);

        }
```

3.3. Работа приложения в сети Интернет

Помимо работы с электронной почтой, существует множество других задач, касающихся работы приложения в сети. Они очень разнообразны, и охватить их все просто невозможно. Поэтому в данном разделе мы рассмотрим основы работы приложений в сети, заострив внимание на решении классических, базовых «сетевых» задач [1].

Первая задача: получение имени хоста рабочего компьютера и преобразование его в IP-адрес. В этом случае для получения имени хоста рабочего компьютера вы должны использовать статический метод GetHostName класса Dns, расположенного в namespace System.Net. Метод возвратит вам строку – имя хоста. Для получения его IP-адреса используем метод GetHostAddresses того же класса. В результате своей работы метод возвращает массив IP-адресов, ассоциированных с хостом:

```
string hostname = Dns.GetHostName();
Console.WriteLine("Hostname: {0}", hostname);
IPAddress[] addresses = Dns.GetHostAddresses (hostname) ;
foreach (IPAddress addr in addresses)
{
Console.WriteLine(" IP Address: {0} ({1})",
addr.ToString(), addr.AddressFamily);
}
```

Вторая задача: выяснение доступности какого-либо компьютера в сети (сервера или сайта). Для этого вам необходимо обратиться к методу Send объекта класса Ping. Этот класс расположен в namespace System.Net.NetworkInformation:

```
Ping ping = new Ping();
PingReply pingreply = ping.Send("www.mail.ru");
Console.WriteLine("address: {0}", pingreply.Address);
Console.WriteLine("rountrip: {0}ms" , pingreply.RoundtripTime);
Console.WriteLine("status: {0}", pingreply.Status);
```

Задача третья: загрузка страницы или текстового файла из сети Интернет по протоколу HTTP. В данном случае вы должны обратиться к классу WebClient и его методу DownloadFile, который в качестве параметров принимает адрес веб-страницы для загрузки и имя выходного файла для выгрузки. Код будет следующим:

```
string url = "http://www.mail.ru";
string outputfile = @"G:/temp.html";
using (WebClient client = new WebClient())
{
try
{
client.DownloadFile(url, outputfile);
}
catch (WebException ex)
{
Console.WriteLine(ex.Message); Console.ReadLine();
}
```

В данном случае необходимо помнить, что приложение «повиснет», пока не загрузит все до конца. Поэтому здесь имеет смысл использовать технологию многопоточности, рассмотренную в разделе «Построение отчетов».

В настоящее время задачи подобного рода возникают все чаще. Приведем для примера краткое описание решения практической задачи, предлагаемой фрилансерам. В настоящее время возросла популярность Интернет-сайтов размещения объявлений, таких как Avito.ru, Slando.ru и прочих. На них размещается огромное количество объявлений в различных категориях и из различных регионов страны. При поиске какого-либо определенного товара (например, подержанного автомобиля) приходится ежедневно следить за появлением новых объявлений, что является достаточно утомительным. Исходя из этого актуальной является задача разработки системы мониторинга появления новых объявлений на сайте Avito.ru по введенным пользователем критериям [3].

Программное обеспечение должно выполнять следующие функции:

- выбор желаемого региона и категории объявлений;
- выбор временного интервала объявлений (объявления, появившиеся позднее какой-либо даты);
- поиск в получившемся списке объявлений по названию, цене.

Для решения задачи парсинга страницы в данном случае была использована альтернативная технология: методы библиотеки htmlAgilityPack.

HtmlAgilityPack – библиотека, распространяемая под лицензией Microsoft Public Licence, обеспечивающей ее свободное использование. Это гибкий анализатор HTML, который строит DOM-объекты, доступные для чтения / записи и поддерживает XPATH или XSLT. Причем, для работы с этой библиотекой знание XPATH или XSLT не требуется. HtmlAgilitiPack – это .NET-библиотека, позволяющая извлекать «из сети» файлы HTML, учитывающая имеющиеся в «реальном мире» некорректные HTML-файлы. Объектная модель этой библиотеки очень похожа на то, что предлагает System.Xml, но эта библиотека предназначена для HTML документов (или потоков).

Аббревиатура DOM расшифровывается как Document Object Model — «объектная модель документа». Это программный интерфейс, не зависящий ни от платформы, ни от языка и позволяющий программам получить доступ к содержимому HTML-, XHTML- и XML-документов. При использовании этой модели нет ограничений касательно структуры документа. Любой документ известной структуры с помощью DOM может быть представлен в виде дерева, каждый узел которого представляет собой элемент, атрибут, текстовый, графический или любой другой объект.

При использовании библиотеки htmlAgilityPack html-документ в текстовом виде преобразуется в DOM-объект, становятся доступны такие возможности, как LINQ и XPATH. При таком представлении данных не составляет труда получить необходимые данные из html-документа.

XPath (от английского XML Path Language) представляет собой язык запросов к XML-документу. Этот язык позволяет осуществить навигацию по DOM в XML. Он является стандартом консорциума W3C и разработан для осуществления доступа к частям документа XML в файлах трансформации XSLT.

LINQ (от английского Language Integrated Query) расшифровывается как запрос, интегрированный в язык. Это технология формирования SQL-подобных запросов, позволяющая использовать удобный и знакомый синтаксис для отправки запросов к любому набору данных, в частности XML-документу. Для выполнения запроса ему необходимо загрузить его содержимое в объект XDocument.

Получение необходимых данных проходит в 3 этапа. Первый этап – это загрузка первых web-страниц для каждой выбранной категории каждого региона. Второй этап – парсинг загруженных web-страниц, заполнение структур для хранения данных об объявлениях. При необходимости загружаются дополнительные web-страницы. Заключительный этап – вывод полученной информации на экран. Полученные из документа данные хранятся в структурированном виде списков объектов в памяти при работе программы. В соответствии с заданными пользователем требованиями выводятся объявления в виде списка. В списке указаны название объявления, время, дата и цена. Имеется возможность прямого перехода на страничку каждого объявления на сайте avito.ru .

Задача парсинга web-страниц является типовой задачей для применения многопоточности: происходят длительные вычисления, которые необходимо увести в фоновый режим. Если в приложениях Windows Forms запустить длительные вычисления в главном потоке, то он не сможет обрабатывать сообщения, поступающие от клавиатуры и мыши. В данном случае существует риск, что приложение будет помечено операционной системой как «Не отвечающее». При применении многопоточности главный поток продолжает работу, реагируя на события, в то время как фоновый поток выполняет вычисления. В C# для запуска фоновых потоков существует класс BackgroundWorker. Кроме возможности запуска какой-либо задачи в фоновом режиме, он предоставляет возможности информирования главного потока о промежуточном состоянии вычислений и об их окончании.

При завершении работы программы полученные данные выгружаются в xml-документ для ускорения работы при следующем запуске.

Четвертая задача: загрузка файлов на ftp-сервер. В данном случае мы также будем использовать методы класса WebClient:

```
string host = "ftp://ftp.Destination.ru";
string username = "mylogin";
string password = "mypassword";
string file = "MyFileForDownload.txt";
Uri uri = new Uri(host);
System.IO.FileInfo info = new System.IO.FileInfo(file) ;
string destFileName = host + "/" + info.Name;
try
{
WebClient client = new WebClient();
client.Credentials = new NetworkCredential(username, password);
byte[] reply = client.UploadFile(destFileName, file);
if (reply.Length > 0)
{
Console.WriteLine("Reply: {0}",
Encoding.ASCII.GetString(reply));
```

```
}
}
catch (WebException ex)
{
Console.WriteLine(ex.Message);
}
```

Мы рассмотрели технологии, применяемые для решения нескольких типовых задач по организации работы приложения в сети.

3.4. Работа с API Google.Маршруты

Все чаще появляются задачи, когда ваше приложение должно рассчитать расстояние или оценить время перемещения из одной точки в другую. Один из способов решения обозначенной проблемы – запрос к API маршрутов Google. Этот API представляет собой специальную службу, рассчитывающую расстояние между двумя точками на карте, а также вычисляющую время перемещения и прокладывающую маршрут, причем для разных путей перемещения (пешком, на общественном транспорте, на машине и т. д.).

Использование службы бесплатное, но существуют ограничения.

- 2500 запросов в день, причем запрос на автомобильный, пешеходный или велосипедный маршрут учитывается как один, а для общественного транспорта – как четыре.
- Запрос вместе с параметрами не должен превышать 2048 символов.
- Нельзя использовать данную службу без отображения карты, для которой был сделан запрос.
- В ответе службы содержится информация об авторских правах и предупреждения, которые должны каким-либо образом быть показаны пользователю.

Запрос к службе маршрутов имеет следующий вид:

http://maps.googleapis.com/maps/api/directions/output?parameters .

В данном случае параметр output отвечат за формат вывода. Он может принимать одно из двух значений: json и xml. После этого параметра идут остальные, как обязательные, так и нет. По стандарту адресов URL для разделения параметров используется знак амперсанда (&).

Рассмотрим список возможных параметров [5].

- origin – адрес или текстовое значение широты и долготы отправного пункта маршрута.
- destination – адрес или текстовое значение широты и долготы отправного пункта маршрута.
- sensor указывает, исходит ли запрос маршрута от устройства с датчиком местоположения. Допустимые значения – true или false.
- mode (значение по умолчанию – driving) указывает используемый способ перемещения для вычисления маршрута.
- waypoints задает массив путевых точек, направляющих маршрут через указанные пункты.
- alternatives – значение true указывает, что служба маршрутов может предложить в ответе несколько альтернативных путей.
- avoid указывает, что при вычислении маршрутов следует избегать либо tolls (платных дорог и мостов), либо highways (автомагистралей).
- units указывает единицы измерения для отображения результатов.
- region – код региона, указанный в виде двузначного числа ccTLD (домен верхнего уровня).
- arrival_time указывает время прибытия для маршрутов общественного транспорта в секундах относительно 00:00 1 января 1970 года в формате UTC.

Ответ службы маршрутов содержит два корневых элемента status и routes. В первом содержатся метаданные по запросу, второй же содержит массив маршрутов из исходной точки к пункту назначения. Поле status может содержать следующие значения [5]:

- OK указывает, что ответ содержит допустимый элемент result.
- NOT_FOUND означает, что по крайней мере для одного указанного пункта (исходный, пункт назначения или путевая точка) не удалось выполнить геокодирование.
- ZERO_RESULTS – указывает, что между исходной точкой и пунктом назначения не найдено ни одного маршрута.
- MAX_WAYPOINTS_EXCEEDED означает, что в запросе указано слишком много элементов массива waypoints. Разрешено использовать массив waypoints, содержащий не более 8 элементов, плюс исходный и конечный пункты. (Пользователи API Google Карт для организаций могут выполнять запросы, содержащие до 23 путевых точек.)
- INVALID_REQUEST – указывает, что запрос является недопустимым.
- OVER_QUERY_LIMIT – означает, что служба получила слишком много запросов от вашего приложения в разрешенный период времени.
- REQUEST_DENIED указывает, что служба отклонила запрос вашего приложения.
- UNKNOWN_ERROR означает, что обработка запроса маршрута невозможна из-за ошибки сервера. При повторной попытке запрос может быть успешно выполнен.

Маршруты, помещенные в массив routes, в свою очередь состоят из вложенных отрезков и шагов. Обратите внимание, что этот массив будет возвращен в любом случае, даже если не будет ни одного

маршрута из исходной точки в конечную. Элемент маршрута copyrights содержит обязательный для отображения текст об авторских правах, а warnings[] включает массив предупреждений, которые следует показать вместе с этим маршрутом.

В зависимости от числа путевых точек маршрут может содержать один или несколько массивов legs[], содержащих информацию об отрезках между двумя пунктами на заданном маршруте. Маршруты без путевых точек представляют собой единственный отрезок. Маршруты с путевыми точками содержат отрезки, соответствующие отдельным участкам пути. В каждом элементе legs могут содержаться следующие поля [5]:

- steps[] содержит массив шагов с информацией о каждом шаге на отрезке пути;
- distance обозначает общее расстояние, охватываемое этим отрезком, и представляет собой поле со следующими элементами:
    - value указывает расстояние в метрах;
    - text содержит удобочитаемое представление расстояния в единицах измерения, принятых в исходном пункте;
- duration обозначает общую продолжительность прохождения этого отрезка в виде поля со следующими элементами:
    - value указывает продолжительность в секундах;
    - text содержит удобочитаемое представление продолжительности;
- start_location содержит значения координат широты и долготы исходной точки этого отрезка;
- end_location содержит значения координат широты и долготы заданного пункта назначения этого отрезка;

- start_address содержит удобочитаемый адрес (обычно улицу и номер дома), отражающий значение start_location для этого отрезка;
- end_address содержит удобочитаемый адрес (обычно улицу и номер дома), отражающий значение end_location для этого отрезка.

Как было сказано выше, каждый отрезок состоит из последовательности элементов steps – шагов, каждый из которых является наименьшей неделимой единицей измерения маршрута, в которой может содержаться одна инструкция, например, «Повернуть налево на Уолл-стрит».

Каждый шаг, входящий в поле steps, может содержать следующие поля [5]:

- html_instructions содержит форматированные инструкции для этого шага, представленные в виде текстовой строки HTML;
- distance содержит расстояние, покрываемое в этом шаге до следующего шага;
- duration содержит время, которое обычно требуется для выполнения этого шага до следующего шага;
- start_location содержит местоположение конечной точки этого шага в виде единого набора полей lat и lng;
- end_location включает местоположение конечной точки этого шага в виде единого набора полей lat и lng;
- sub_steps содержит подробные инструкции, относящиеся к шагам, преодолеваемым пешком или на автомобиле в рамках маршрутов общественного транспорта. Вложенные шаги доступны только в том случае, если параметру travel_mode присвоено значение «transit». Массивы sub_steps и steps имеют одинаковый тип;
- transit_details содержит сведения, относящиеся к общественному транспорту. Это поле возвращается только в том случае, если параметру travel_mode присвоено значение «transit».

Рассмотрим пример получения автомобильных маршрутов между городами Торонто и Монреаль в формате xml. Строка запроса в данном случае будет следующей:

http://maps.googleapis.com/maps/api/directions/xml?origin=Toronto&destination=Montreal&sensor=false

Допустим, мы хотим написать консольное приложение на C# для выгрузки этой информации в файл. Для работы нам понадобится подключение к сети Интернет и функционал из следующих пространств имен: System.Net и System.IO. Сам код будет следующим:

```
WebClient Client = new WebClient();
using (Stream strm = Client.OpenRead("http://maps.googleapis.com/maps/api/directions/xml?origin=Toronto&destination=Montreal&sensor=false"))
{
StreamReader sr = new StreamReader(strm);
FileStream f = new FileStream("G:\\1.xml", FileMode.CreateNew);
StreamWriter sw = new StreamWriter(f);
sw.WriteLine(sr.ReadToEnd());
 Console.WriteLine("ok");
sw.Close();
f.Close();
}
```

В итоге в файл «G:\\1.xml» выгрузится следующая информация:


```
<?xml version="1.0" encoding="UTF-8"?>
<DirectionsResponse>
 <status>OK</status>
 <route>
  <summary>ON-401 E</summary>
  <leg>
   <step>
    <travel_mode>DRIVING</travel_mode>
    <start_location>
     <lat>43.6533103</lat>
     <lng>-79.3827675</lng>
    </start_location>
```

```
    <end_location>
     <lat>43.6557259</lat>
     <lng>-79.3837332</lng>
    </end_location>
    <polyline>
     <points>e`miGhmocNgBf@cBb@KBkGnB</points>
    </polyline>
    <duration>
     <value>28</value>
     <text>1 min</text>
    </duration>
    <html_instructions>Head &lt;b&gt;north&lt;/b&gt; on &lt;b&gt;Bay St&lt;/b&gt; toward &lt;b&gt;Hagerman St&lt;/b&gt;</html_instructions>
    <distance>
     <value>280</value>
     <text>0.3 km</text>
    </distance>
   </step>

. . .//несколько схожих блоков step

   <step>
    <travel_mode>DRIVING</travel_mode>
    <start_location>
     <lat>45.5101458</lat>
     <lng>-73.5525249</lng>
    </start_location>
    <end_location>
     <lat>45.5085712</lat>
     <lng>-73.5537674</lng>
    </end_location>
    <polyline>
     <points>muwtGfv|_MfC`BVNt@n@RNZVt@n@</points>
    </polyline>
    <duration>
     <value>40</value>
     <text>1 min</text>
    </duration>
    <html_instructions>Turn &lt;b&gt;right&lt;/b&gt; onto &lt;b&gt;Rue Notre-Dame E&lt;/b&gt;&lt;div style=&quot;font-size:0.9em&quot;&gt;Destination will be on the right&lt;/div&gt;</html_instructions>
    <distance>
     <value>200</value>
     <text>0.2 km</text>
    </distance>
```

```
    <maneuver>turn-right</maneuver>
   </step>

//общая информация
   <duration>
    <value>18787</value>
    <text>5 hours 13 mins</text>
   </duration>
   <distance>
    <value>542389</value>
    <text>542 km</text>
   </distance>
   <start_location>
    <lat>43.6533103</lat>
    <lng>-79.3827675</lng>
   </start_location>
   <end_location>
    <lat>45.5085712</lat>
    <lng>-73.5537674</lng>
   </end_location>
   <start_address>Toronto, ON, Canada</start_address>
   <end_address>Montreal, QC, Canada</end_address>
//общая информация

  </leg>
  <copyrights>Map data B©2013 Google</copyrights>
  <overview_polyline>
   <points>e`...OzHxF</points>
  </overview_polyline>
  <bounds>
   <southwest>
    <lat>43.6533103</lat>
    <lng>-79.3837332</lng>
   </southwest>
   <northeast>
    <lat>45.5101458</lat>
    <lng>-73.5525249</lng>
   </northeast>
  </bounds>
 </route>
</DirectionsResponse>
```


Получив в файле ответ сервиса, необходимо извлечь из него необходимые данные. Есть три способа разбора XML-документов:

средствами класса xmlDocument, средствами класса XmlTextReader или с помощью LINQ-запросов. Первый класс требует определенных затрат ресурсов, то есть при обработке больших файлов он будет работать медленно. Второй класс работает быстро, но имеет не столь богатый и удобный функционал. LINQ-запросы сочетают в себе производительность и удобство работы. Рассмотрим примеры работы всеми способами.

Начнем с класса xmlDocument. Он содержит методы для загрузки XML-документов из файлов, потоков или строк. Возьмем для примера отрезок нашего документа с общей информацией, оформив его в виде отдельного корректного xml-документа с единственным корневым элементом:

```
<DirectionsResponse>
<duration>
   <value>18787</value>
   <text>5 hours 13 mins</text>
  </duration>
  <distance>
   <value>542389</value>
   <text>542 km</text>
  </distance>
  <start_location>
   <lat>43.6533103</lat>
   <lng>-79.3827675</lng>
  </start_location>
  <end_location>
   <lat>45.5085712</lat>
   <lng>-73.5537674</lng>
  </end_location>
  <start_address>Toronto, ON, Canada</start_address>
  <end_address>Montreal, QC, Canada</end_address>
</DirectionsResponse>
```

Допустим, мы хотим извлечь оттуда общее время и протяженность поездки. Код будет таким:

```
class Program
```

```
   {
      const string sourceXml =
      "<DirectionsResponse>"+
"<duration> <value>18787</value> <text>5 hours 13 mins</text> </duration>"
+
  "<distance> <value>542389</value> <text>542 km</text> </distance>" +
  "<start_location><lat>43.65333</lat><lng>-79.3827</lng>  </start_location>"
+
  "<end_location>              <lat>45.5082</lat>              <lng>-73.5534</lng>
</end_location>" +
  "<start_address>Toronto, ON, Canada</start_address>" +
"<end_address>Montreal, QC, Canada</end_address>"+
"</DirectionsResponse>";

      static void Main(string[] args)
      {
         XmlDocument doc = new XmlDocument();
         doc.LoadXml(sourceXml);
         Console.WriteLine("duration: {0}",
         doc.GetElementsByTagName("text")[0].InnerText);
         Console.WriteLine("distance: {0}",
         doc.GetElementsByTagName("text")[1].InnerText);

         Console.ReadLine();
      }
   }
```

Обратите внимание, что для работы с этим классом необходимо подключить пространство имен System.Xml.

Теперь перейдем ко второму классу. Для его использования, кроме указанного выше, нужно подключить namespace System.IO. Код в данном случае будет следующим:

```
  class Program
   {
const string sourceXml =
      "<DirectionsResponse>"+
"<duration> <value>18787</value> <text>5 hours 13 mins</text> </duration>"
+
  "<distance> <value>542389</value> <text>542 km</text> </distance>" +
  "<start_location><lat>43.65333</lat><lng>-79.3827</lng>  </start_location>"
+
```

```
  "<end_location>          <lat>45.5082</lat>          <lng>-73.5534</lng>
</end_location>" +
  "<start_address>Toronto, ON, Canada</start_address>" +
"<end_address>Montreal, QC, Canada</end_address>"+
"</DirectionsResponse>";

      static void Main(string[] args)
      {
       bool first = true;
using (StringReader reader = new StringReader(sourceXml))
using (XmlTextReader xmlReader = new XmlTextReader(reader))
{
while (xmlReader.Read())
{
if (xmlReader.NodeType == XmlNodeType.Element)
{
if (xmlReader.Name == "text")
{
   xmlReader.Read();
   if (first) { Console.WriteLine("duration: {0}", xmlReader.Value); first = false; }
   else Console.WriteLine("distance: {0}", xmlReader.Value);
}}}
}
         Console.ReadLine();
      }
   }
```

Как мы видим, код в данном случае получился более сложным и объемным.

Теперь рассмотрим извлечение данных из документа с помощью технологии LINQ. Вскользь мы ее уже касались выше, поэтому здесь перейдем сразу к синтаксису. Основной оператор запроса from имеет следующий вид:

```
from <переменная диапазона> in <источник данных>
where <критерий отбора>
select <переменная>
```

Для работы нам потребуется подключить пространство имен System. Xml.Linq. Код извлечения данных будет следующим:

```
class Program
```

```
  {
const string sourceXml =
      "<DirectionsResponse>"+
"<duration> <value>18787</value> <text>5 hours 13 mins</text> </duration>"
+
  "<distance> <value>542389</value> <text>542 km</text> </distance>" +
  "<start_location><lat>43.65333</lat><lng>-79.3827</lng>  </start_location>"
+
  "<end_location>              <lat>45.5082</lat>              <lng>-73.5534</lng>
</end_location>" +
  "<start_address>Toronto, ON, Canada</start_address>" +
"<end_address>Montreal, QC, Canada</end_address>"+
"</DirectionsResponse>";

      static void Main(string[] args)
      {
       bool first = true;
         using (StringReader reader = new StringReader(sourceXml))
         {
            XDocument doc = XDocument.Load(reader);
            var chaptersWithHe = from chapter in doc.Descendants("text")
                                 select chapter.Value;
            foreach (var val in chaptersWithHe)
            {
               if (first) { Console.WriteLine("duration: {0}", val); first = false; }
                  else Console.WriteLine("distance: {0}", val);
            }
         }
        Console.ReadLine();          }
```

Единственным ощутимым минусом в работе с xml-ответом сервиса маршрутов будет необходимость самостоятельно проводить анализ документа с целью написания алгоритма извлечения необходимых данных, но так как выходная структура регламентирована, то алгоритм будет одним для всех запросов.

3.5. Работа с системой «Клиент-Банк»

Для удобной и оперативной работы своих клиентов многие банки предлагают воспользоваться автоматизированной системой дистанционного банковского обслуживания «Клиент-Банк:iBank2», разработанной ООО «БИФИТ» [8].

Система предназначена для осуществления операций с рублевыми и валютными счетами, оперативного получения выписок по лицевым счетам, а также обмена с банком письмами, ведения архива переданных в банк документов, копирования ранее созданных поручений и сохранения их как шаблонов для дальнейшего использования. Работа ведется через Web-браузер с любого компьютера, имеющего доступ в Интернет.

К основным преимуществам системы можно отнести экономию сил, времени и средств, затрачиваемых на поездку в банк; экономию времени, проводимого в ожидании обработки документов; возможность работы с банковскими счетами из офиса; высокую скорость документооборота.

Данная система не требует от клиента установки какого-либо дополнительного программного обеспечения, а вся информация хранится на защищенных серверах банка. На компьютере клиента осуществляется только лишь ее обработка. Теоретически организация может управлять своими счетами с любого компьютера, подключенного к сети.

Подключение к системе «Клиент—Банк» производится по следующей схеме.

Шаг 1. Клиент регистрируется на сервере «Клиент-Банк», и в результате на рабочем месте клиента создается электронно-цифровая подпись (ЭЦП), и распечатывается «Сертификат открытого ключа ЭЦП».

Шаг 2. Клиент получает у банка подготовленный договор на обслуживание по системе «Клиент-Банк».

Шаг 3. Клиент подписывает со своей стороны договор и «Сертификат открытого ключа ЭЦП» и направляет эти документы в Банк.

Шаг 4. Списание комиссии за подключение к системе согласно договору на обслуживание.

Шаг 5. Открытие доступа для работы через систему дистанционного банковского обслуживания.

Удобство работы с данной системой для разработчика заключается в том, что не надо работать с ней напрямую, соответственно, не нужно беспокоиться о защите данных в сети. В функционал системы iBank2 обязательно входит функция обмена документами с бухгалтерскими программами. Импорт и экспорт всех типов документов происходит через обмен файлами в текстовом или xml-формате. То есть клиент может загружать в клиентский АРМ платежные поручения, созданные бухгалтерскими программами, для передачи в банк и выгружать из клиентского АРМ выписки по счетам для работы с ними в бухгалтерских программах. Имена файлов обмена настраиваются в процедурах выгрузки и загрузки. Например, для взаимодействия с 1С по умолчанию используется: «1c_to_kl.txt» для выгрузки и «kl_to_1c.txt» для загрузки. Это обычные текстовые файлы в настраиваемой кодировке (Windows или DOS). Первая строка файла содержит специальную последовательность символов – внутренний признак файла обмена данными между 1С и системой электронного банкинга. Затем последовательно идут строки, содержащие необходимую служебную информацию, и строки, содержащие условия отбора документов (интервал дат, расчетные счета, виды документов). Пример содержимого файла приведен на рисунке 3.2 [8]. Мы видим, что информация организована следующим образом: существует ключ раздела, занимающий отдельную строку, или ключ данных, отделенный от значения знаком равенства. Алгоритм построения этой структуры всегда одинаков, она легко читаема как с машинной, так и с человеческой точек зрения.

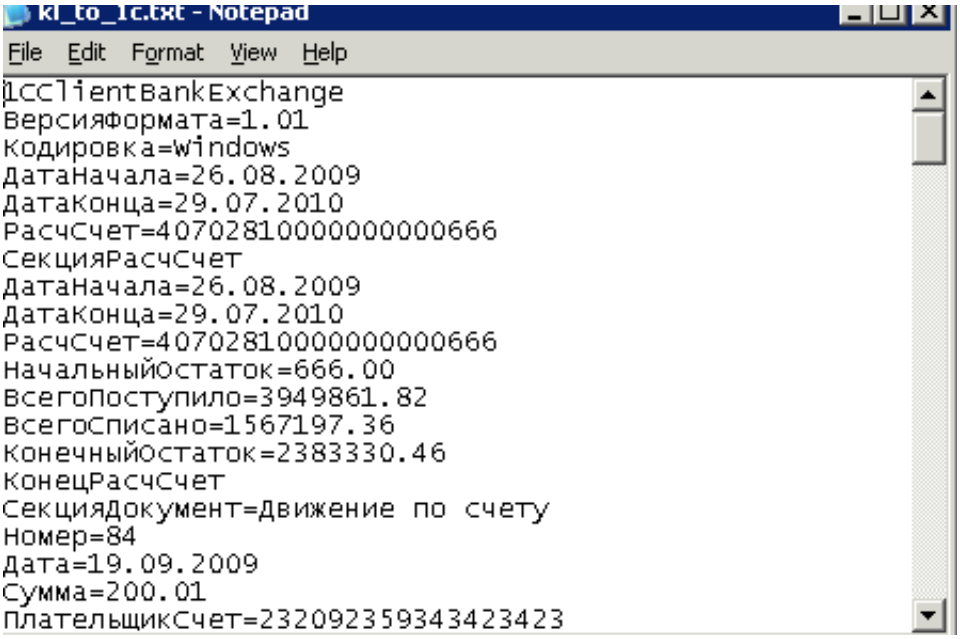

Рисунок 3.2. Пример выходного файла «Клиент-банка»

Таким образом, основная задача вашего приложения – корректно разобрать этот текстовый файл, выделив ключевые элементы, такие как СекцияДокумент, Сумма, ПлательщикСчет, Дата и так далее. Соотнесенную с этими элементами информацию необходимо занести в базу данных.

3.6. Работа с ОС и аппаратной частью

Иногда вам приходится корректировать функциональность своих приложений в зависимости от того, под какой операционной системой оно работает [1]. Получить информацию об операционной системе вам поможет класс System.Environment.OSVersion. Его объект содержит свойства Platform, ServicePack, Version и VersionString, на значения которых вы можете ориентироваться.

Самой распространенной задачей корректирования функциональности в зависимости от операционной системы является обеспечение корректности работы приложения как в 32-битовом, так и в 64-битовом окружении. Решение лежит в простой настройке конфигурации: в выборе целевой платформы в опциях построения

проекта. Так как сборки .Net содержат код, подвергающийся компиляции на этапе выполнения на текущей платформе, то по умолчанию выбирается вариант Any CPU. Такие сборки будут работать на любой архитектуре без перекомпиляции исходного кода. Но если вашей сборке требуется ссылаться на другую, откомпилированную под конкретную архитектуру, то ваша сборка также должна соответствовать этой архитектуре. То есть, если у вас есть 32-битовая dll и сборка, откомпилированная под Any CPU, то в 32-битовом окружении все будет работать без проблем, так как ваша сборка преобразуется в 32-битовую. А вот на 64-битовой операционной системе возникнут проблемы при взаимодействии сборки и библиотеки, так как у них будет разная разрядность. Выход в данном случае – компиляция сборки под 32-битовую систему, так как проблем при выполнении 32-битовых процессов в 64-битовых операционных системах нет.

Следующая задача взаимодействия приложения и операционной системы – мониторинг изменений разрешения монитора, режима питаний и прочее. Для решения данной проблемы можно воспользоваться функциональностью класса SystemEvents из пространства имен Microsoft.Win32. Данный класс содержит несколько статических событий, которые ваше приложение может прослушивать, таких как DisplaySettingsChanged, InstalledFontsChanged, PowerModeChanged и так далее. Единственной тонкостью при работе с событиями является необходимость освобождать дескрипторы событий по окончании работы приложения, иначе может произойти утечка памяти.

Помимо стандартного функционала .Net, для работы с операционной системой можно воспользоваться стандартными библиотеками, выпускаемыми Microsoft. Например, для доступа к функциональным возможностям Windows 7 можно воспользоваться

бесплатным пакетом Windows API Code Pack for Microsoft.NET Framework, доступным в библиотеке кода MSDN. В качестве примера предоставляемых им возможностей рассмотрим работу с файловыми диалогами. В данной библиотеке есть класс CommonOpenFileDialog, расположенный в пространстве имен Microsoft.WindowsAPICodePack.Dialogs, среди прочих предоставляющий возможность добавлять файл в список недавно открытых (свойство AddToMostRecentlyUsedList):

```
CommonOpenFileDialog OpenDialog = new CommonOpenFileDialog();
OpenDialog.AddToMostRecentlyUsedList = true;
OpenDialog.ShowDialog();
```

Этот же класс предоставляет возможность доступа к объектам, не входящим в файловую систему, например, к появившимся впервые в Windows 7 библиотекам (свойство AllowNonFileSystemItems):

```
CommonOpenFileDialog OpenDialog = new CommonOpenFileDialog();
OpenDialog. AllowNonFileSystemItems = true;

if (OpenDialog.ShowDialog()== CommonFileDialogResult.OK)
{
ShellObject so = OpenDialog.FileAsShellObject;
ShellLibrary lib = so as ShellLibrary;
if (lib != null)
{
textBoxInfo.AppendText(
"You picked a library: " + library.Name + ", Type: "
+ library.LibraryType.ToString());
foreach (ShellFileSystemFolder f in (ShellLibrary)so)
{
MessageBox.Show (f. Path.ToString()) ;
}
}
}
```

Теперь рассмотрим взаимодействие с аппаратной составляющей компьютера. Например, получение информации о количестве процессоров, объеме памяти и прочее. Эта информация может быть извлечена двумя способами. Первый – средствами самой платформы

.Net, второй – с помощью инструментария управления Windows (WMI, Windows Management Instrumentation). Разница заключается в объеме доступной информации. .Net предоставляет сведения через статические свойства класса Environment: MachineName, ProcessorCount, Is64BitOperatingSystem, Is64BitProcess, IsLittleEndian. Причем свойство ProcessorCount показывает логическое, а не физическое количество процессоров. Также этому классу недоступна информация о памяти. Для получения ее, а также получения физического количества процессоров мы должны обратиться ко второму способу. Для его использования нам необходимо подключить пространство имен System.Management и прописать следующий код:

```
ManagementObjectSearcher objects =
new ManagementObjectSearcher("SELECT * FROM
Win32_ComputerSystem");
ManagementObjectCollection coll = objects.Get();
foreach(ManagementObject obj in coll)
{
  Console.WriteLine(obj["NumberOfProcessors"].ToString());
}
```

Для получения информации о памяти код будет таким:

```
ManagementObjectSearcher objects =
new ManagementObjectSearcher(
"SELECT * FROM Win32_PhysicalMemory");
ManagementObjectCollection coll = objects.Get();
UInt64 total = 0;
foreach (ManagementObject obj in coll)
{
total += (UInt64)obj["Capacity"];
}
        Console.WriteLine(total);
```

Информацию о доступных дисплеях можно получить, используя код:

```
foreach (Screen screen in
System.Windows.Forms.Screen.AllScreens)
{
Console.WriteLine("Экран {0}", screen.DeviceName);
Console.WriteLine("Границы: {0}", screen.Bounds);
Console.WriteLine("Рабочая область: {0}",screen.WorkingArea);
Console.WriteLine("Глубина: {0}", screen.BitsPerPixel);
}
```

Информацию о текущем режиме питания можно получить из статических свойств класса PowerManager из рассмотренного ранее API Code Pack. Код в данном случае будет следующим:

```
bool _IsBatteryPresent = PowerManager.IsBatteryPresent;
string _PowerSource = PowerManager.PowerSource.ToString();
```

Теперь рассмотрим более сложные взаимодействия приложения и операционной системы, и начнем с журнала событий. В Microsoft Windows это стандартный способ для приложений и операционной системы централизованного хранения информации о важных событиях. Специальная служба сохраняет события от различных источников в едином журнале событий, а программный интерфейс (API) позволяет приложениям записывать в журнал информацию и просматривать существующие записи. На платформе .NET для этого API имеются специальные оболочки, например, класс EventLog.

Для работы с журналом событий вы должны создать источник информации для журнала, ассоциированный с вашим приложением, а затем выбрать или создать журнал, в который вы хотите записать сообщение. Первая операция выполняется только один раз, и для ее выполнения необходимы права администратора. Начиная с Windows Vista приложения не могут работать с правами администратора без дополнительного подтверждения этого, что сопровождается появлением диалогового окна, требующего подтверждения действий и, возможно, ввода пароля. Для создания источника журнала событий

вам в таком случае придется или запустить все приложение с правами администратора (что не является хорошим выходом), или дать приложению возможность запросить административные права только на выполнение этой функции. Рассмотрим реализацию второго варианта.

Первое, что нам необходимо сделать, – разместить на форме приложения кнопку, маркированную специальным узнаваемым значком – щитом, чтобы дать понять пользователю, что на выполнение функции потребуются административные права.

Стандартным функционалом платформы .Net отобразить пиктограмму на кнопке нельзя. Чтобы включить ее, необходимо отправить сообщение элементу управления Button (кнопка) средствами библиотеки WinAPI, то есть включить в наш проект так называемый неуправляемый код. Напомним, что управляемым кодом называется код, выполняющийся под управлением среды Common Language Runtime (CLR), которая реализует управление памятью, типами данных, межъязыковым взаимодействием, развертыванием (deployment) приложений. Код, выполняющийся вне этой среды, называется неуправляемым. Для того чтобы подключить к управляемому проекту неуправляемый код, нам необходимо воспользоваться платформенным вызовом. Платформенный вызов – это служба, обеспечивающая поиск и вызов экспортируемой в управляемый код неуправляемой функции из библиотеки динамической компоновки и в случае необходимости обеспечивающая передачу ее параметров через границы процесса (маршалинг).

Для вызова неуправляемой функции выполняются следующие действия. Служба платформенного вызова определяет местонахождение библиотеки, содержащей функцию, и при необходимости загружает ее в память. Необходимость возникает,

если библиотека ранее не была загружена. Затем определяется адрес вызываемой функции в памяти, значения ее параметров заносятся в стек, и, если нужно, выполняется маршалинг данных. После всего этого происходит передача управления неуправляемой функции.

Для того чтобы воспользоваться платформенным вызовом, необходимо выполнить следующие действия. Во-первых, определить, какую функцию и из какой библиотеки требуется вызвать. Во-вторых, создать класс для сохранения вызываемой функции и объявить в этом классе ее прототип. Затем в нужном месте кода через прототип вызвать функцию. Создание класса для размещения неуправляемой функции называется упаковкой неуправляемой функции в обертку управляемого класса. Функционал для программной реализации платформенного вызова становится доступным, если прописать пространство имен System.Runtime.InteropServices.

Теперь вернемся к созданию кнопки с пиктограммой UAC (User Account Control). Первый шаг, который вы должны сделать, это создать класс-обертку для функции неуправляемого кода. Пусть он будет называться Win32. Нам необходима функция SendMessage из библиотеки user32.dll. Мы описываем ее заголовок в классе один в один, как он описан в библиотеке, и перед этим заголовком помещаем атрибут DllImport, в параметрах которого указываем имя нужной нам библиотеки:

```
class Win32
  {  [DllImport("User32.dll")]
     public static extern IntPtr SendMessage(HandleRef hWnd, UInt32 Msg,
     IntPtr wParam, IntPtr lParam);

     public const UInt32 BCM_SETSHIELD = 0x0000160C;    }
```

Не забудьте прописать следующие namespace:

```
using System.Runtime.InteropServices;
using System.Security.Principal;
using System.Diagnostics;
```

Теперь мы можем создать пользовательский класс «Кнопка», наследника класса Button, в котором предусмотрено отображение пиктограммы:

```
public class UACButton : Button
{
private bool _showShield = false;

public bool ShowShield
{
get
{
return _showShield;
}
set
{
bool needToShow = value && !IsElevated();
if (this.Handle != IntPtr.Zero)
{
Win32.SendMessage(new HandleRef(this, this.Handle),
Win32.BCM_SETSHIELD,
new IntPtr(0), new IntPtr(needToShow ? 1 : 0));
_showShield = needToShow;
}
}
}

private bool IsElevated()
{
WindowsIdentity identity = WindowsIdentity.GetCurrent();
WindowsPrincipal principal = new WindowsPrincipal(identity);
return principal.IsInRole(WindowsBuiltInRole.Administrator);
}
public UACButton()
{
this.FlatStyle = FlatStyle.System;
}
}
```

Теперь на форму можно поместить кнопку, но она будет обычного типа Button. Для того чтобы назначить ей нашу функциональность, делаем следующее. Переходим в файл

Form1.Designer.cs и ищем там метод private void InitializeComponent(). В этом методе находим строчку

```
this.button1 = new System.Windows.Forms.Button();
```

и меняем ее на следующую:

```
this.button1 = new WindowsFormsApplication5.UACButton();
```

В данном примере считается, что класс вашей кнопки прописан в namespace WindowsFormsApplication5, и кнопка, на которую нужно поместить пиктограмму, называется button1. После того, как вы это сделали, в окошке Properties вашей кнопки должно появиться свойство ShowShield. Установите его в true. Если свойство сразу не появилось, сделайте Rebuild Solution.

Теперь нам нужно описать функциональность этой кнопки, так как появление пиктограммы еще не означает работу приложения с правами администратора. Более того, у запущенного приложения нельзя повысить его права, поэтому процесс придется перезапустить. Здесь у вас может возникнуть резонный вопрос осмысленности наших действий: если процесс придется перезапускать, то зачем все эти лишние действия? Ответ прост: мы разграничим функциональность и сделаем так, что в режиме администратора приложение сможет выполнить только одну функцию и закрыться. Для этого мы запустим новый процесс, передав в его основную функцию (Main) строковый аргумент, и при запуске приложения будем отслеживать, если строка внешних аргументов пуста, работаем в обычном режиме, если нет – выполняем действие в режиме администратора. Для этого мы переходим в файл Program.cs и исправляем метод Main так, чтобы он был похож на следующий:

```
[STAThread]
    static void Main()
    {
```

```
        string[] args = Environment.GetCommandLineArgs();
foreach (string arg in args)
{
if (string.Compare("-createEventSource", arg)==0)
{
CreateEventSource();
return;
}
}
        Application.EnableVisualStyles();
        Application.SetCompatibleTextRenderingDefault(false);
        Application.Run(new Form1());
    }
```

Теперь в кнопке напишем код запуска нового процесса с повышенными правами:

```
private void button1_Click(object sender, EventArgs e)
{         ProcessStartInfo startInfo = new ProcessStartInfo();
        startInfo.FileName = Application.ExecutablePath;
        startInfo.Arguments = "-createEventSource";
        startInfo.Verb = "AdminsRigth";
        try
        {
           Process proc = Process.Start(startInfo);
           proc.WaitForExit();
        }
        catch (Exception ex)
        {
           MessageBox.Show(
           "Error: " +
           ex.Message);
        }

}
```

Теперь мы можем в нашем классе Program написать код записи в журнал событий, не забыв подключить пространство имен System.Diagnostics:

```
public const string EvntLogName = "MyLog"; //Имя журнала
public const string EvntLogSource = "MySource"; //Имя источника
```

```
private static void CreateEventSource()
      {
//Создание источника информации
          if (!EventLog.SourceExists(LogSource))
          {

EventSourceCreationData data = new
EventSourceCreationData(EvntLogSource, EvntLogName);
             EventLog.CreateEventSource(data);
          }
//Запись данных
          using (EventLog log = new EventLog(Program. EvntLogName, ".",
Program. EvntLogSource))
          {
            log.WriteEntry("MyMessage", EventLogEntryType.Information, 1);
          }
      }
```

Чтение записей из журнала событий можно осуществить следующим образом:

```
using (EventLog log = new EventLog(Program.EvntLogName, ".",
Program.EvntLogSource) )
{
StringBuilder sb = new StringBuilder();

foreach (EventLogEntry entry in log.Entries)
{
sb.AppendFormat("{0}, {1} {2}) {3}",
entry.TimeGenerated, entry.InstanceId,
entry.EntryType, entry.Message);
sb.AppendLine();
}
MessageBox.Show(sb.ToString(),"My events");
}
```

Кроме журнала событий, есть еще несколько системных объектов, с которыми иногда приходится работать. К таким объектам можно отнести реестр Windows, содержащий информацию и настройки для аппаратного и программного обеспечения, профилей пользователей, предустановки и прочее. По сути реестр представляет собой дерево, состоящее из множества разделов, подразделов и

параметров реестра, которым сопоставлен набор вспомогательных файлов, содержащих резервные копии этих данных. Для работы с реестром используются классы Registry и RegistryKey из пространства имен Microsoft.Win32.

Пример кода, выводящего ключи из HKLM и ассоциированные им значения:

```
using (RegistryKey hklm = Registry.LocalMachine)
using (RegistryKey keyRun =
hklm. OpenSubKey (@" Software\Microsof t\Windows\CurrentVersion\Run") )
{

foreach (string valueName in keyRun.GetValueNames())
{
Console.WriteLine("Name: {0} = Value: {1}",
valueName, keyRun.GetValue(valueName));
}

}
```

Собственный ключ в реестре можно создать следующим образом:

```
using (RegistryKey software =
Registry.CurrentUser.OpenSubKey(@”Software", true))
using (RegistryKey myKeyRoot =
software.CreateSubKey(
"MyKeyNew",
RegistryKeyPermissionCheck.ReadWriteSubTree,
RegistryOptions.Volatile))
{
myKeyRoot.SetValue("NumberOfChapters", 2 8);
myKeyRoot.SetValue("SomeSomeSome",
Int64.MaxValue, RegistryValueKind.QWord);
}
```

Вызов OpenSubKey с параметром true дал нам возможность записывать в подключ. Значение «ложь» лишило бы нас этой возможности. Использование в CreateSubKey параметра

RegistryOptions.Volatile означает, что ключ должен быть удален при перезапуске компьютера.

Удаление записи из реестра осуществляется так:

```
software.DeleteSubKeyTree("MyKeyNew");
```

Теперь рассмотрим функционал, предназначенный для работы со службами Windows – специальными приложениями, выполняющимися независимо от статуса пользователя и, если настроено, запускаемыми автоматически при старте системы. Для управления службами в состав Windows входит специальный диспетчер: Service Control Manager – SCM. С его помощью можно управлять созданием, удалением, запуском и остановкой служб. Основное отличие службы от обычного приложения – возможность принимать сообщения от SCM.

Предусмотрено несколько режимов для служб: запрещен к запуску, ручной запуск, автоматический запуск при загрузке компьютера, автоматический отложенный запуск, автоматический запуск и невозможность для пользователя остановить службу.

Приведем пример нескольких стандартных служб. DHCP-клиент – регистрирует и обновляет IP-адреса и DNS-записи для этого компьютера. DNS-клиент – кэширует имена DNS (Domain Name System) и регистрирует полное имя данного компьютера. Автоматическое обновление – включает загрузку и установку обновлений Windows.

Для управления службами платформа .Net предоставляет функционал класса System.ServiceProcess.ServiceController. С его помощью и при наличии необходимых прав доступа можно управлять службами, как показано ниже:

```
ServiceController controller = new ServiceController (“MySomeService”);
controller.Start();
if (controller.CanPauseAndContinue)
{
```

```
controller.Pause();
controller.Continue();

}
controller.Stop();
```

Если же вы хотите оформить свое приложение в виде службы, то вам понадобится функционал класса System.ServiceProcess.ServiceBase. Рассмотрим пример реализации службы, выполняющий запись в файл информации о своем запуске, работе и остановке. Причем работа службы осуществляется в отдельном потоке. Для реализации вышеописанного мы должны определить собственный класс, наследуемый от ServiceBase, и переопределить в нем методы OnStart и OnStop, а также написать метод, выполняющийся в отдельном потоке работы службы:

```
public partial class MyService : ServiceBase
{

Thread _newThread;

bool _continue = false;

public MyService()
{
InitializeComponent ();
}

protected override void OnStart(string[] args)
{
_continue = true;
WriteString("Begin");
_newThread = new Thread(new ThreadStart(ThreadWork));
_newThread.Start();
}

protected override void OnStop()
{
_continue = false;
WriteString("End");
}
```

```
private void WriteString(string line)
{
using (FileStream fs = new FileStream(
@"D:\Temp\Output.log", FileMode.Append))
using (StreamWriter writer = new StreamWriter(fs))
{
writer.WriteLine(line);
}
}

private void ThreadWork()
{
while (_continue)
{
Thread.Sleep(5000);
WriteString(string.Format("{0} - running.",
DateTime.Now));
}
 }
}
```

Так как функционал платформы .Net позволяет реализовать несколько служб в одном выполняемом файле, то в функции Main необходимо прописать следующее. Во-первых, создать массив типа ServiceBase, который будет содержать ссылки на все запускаемые службы. Во-вторых, заполнить этот массив объектами и передать его в качестве параметра статическому методу Run класса ServiceBase:

```
static void Main()
{ ServiceBase[] RunServices = new ServiceBase[]{new MyService() };
ServiceBase.Run(RunServices); }
```

На этом описание базового функционала службы закончено. Теперь необходимо установить службу, воспользовавшись возможностями среды Visual Studio и утилитой InstallUtil.exe. Первое, что вам необходимо сделать, – создать класс-установщик. Для этого прейдите к конструктору объекта, представляющего службу, сделайте на нем щелчок правой кнопкой мыши и выберите в контекстном меню пункт Add Installer (Добавить установщик). Среда

автоматически создаст установщик, для использования которого вам необходимо открыть в Visual Studio командную строку с правами администратора, перейти в каталог, содержащий выполняемый файл службы, и написать команду:

```
InstallUtil /i MyService.exe
```

Удаление службы происходит практически так же, только вместо атрибута /i команде передается атрибут /u.

Контрольные вопросы к разделу 3

1. Перечислите основные классы, используемые для взаимодействия приложения с операционной системой.
2. Чем отличается служба Windows от обычного приложения?
3. Перечислите шаги, которые необходимо выполнить для запуска приложения с правами администратора.
4. Что такое система «Клиент-Банк»?
5. Какие ограничения накладываются на использование сервиса Google.Маршруты?
6. Перечислите классы, требуемые для реализации отправки электронных писем.
7. Перечислите классы, требуемые для реализации получения электронных писем.
8. Перечислите классы, требуемые для организации работы приложения в сети.
9. Что такое служба платформенного вызова?
10. Что такое технология WCF?

4. ПРИМЕРЫ ЗАДАЧ АВТОМАТИЗАЦИИ И ПРОЕКТИРОВАНИЯ ПРИЛОЖЕНИЯ

В данном разделе приводятся примеры задач автоматизации, которые могут быть решены средствами платформы .Net, а также алгоритм построения начального проекта приложения. Это обязательный этап, без которого написать приложение очень сложно. Однако стоит заметить, что приведенный алгоритм носит рекомендательный характер. Способы получения первоначального проекта могут быть другими.

4.1. Примеры задач автоматизации

Приведенные примеры могут быть использованы в качестве задач на курсовые проекты для направлений подготовки бакалавров «Программная инженерия» и «Прикладная информатика». В каждой задаче, помимо формулировки, содержатся рекомендации по количеству студентов, которые должны работать над реализацией приложения, а также по тому, какому конкретно направлению рекомендуется данная задача. Основной упор в задачах для направления «Программная инженерия» должен быть сделан на различные сетевые технологии и работу с сервисами. Для «Прикладной информатики» акцент необходимо сделать на расчеты и оформление стандартной экономической документации. Однако некоторые из приводимых задач являются комбинированными, рассчитанными на смешанные команды.

Система планирования автомобильного отдыха на Черном море

Направление: «Программная инженерия», количество студентов: 3.

Необходимо разработать приложение, подбирающее по запросу пользователя гостиницу, удовлетворяющую требованиям, на требуемом ему черноморском курорте, прокладывающее автомобильный маршрут и вычисляющее стоимость поездки.

Информацию о гостиницах рекомендуется брать с сайта otdih.nakubani.ru, а для прокладки маршрута использовать сервис Google.Maps. В базе данных необходимо хранить основную информацию о рассматриваемых гостиницах, периодически ее актуализируя. То есть основной поиск должен быть осуществлен по локальной базе данных, если дата актуальности информации устраивает пользователя. В результате своей работы приложение должно по запросу пользователя выводить информацию о трех наиболее подходящих вариантах и для окончательно выбранного варианта формировать отчет в форматах pdf, Ms Word, OpenOffice.writer с требуемой пользователю информацией (то есть выводимые в отчет характеристики гостиницы выбирает пользователь), а также информацию о стоимости и длительности поездки, разделяя время в пути и время отдыха. Кроме того, должна быть возможность вывода в OpenOffice.calc и MS Excel статистики о популярности тех или иных курортов и отдельных гостиниц. В Excel, помимо таблицы с данными, должна выводиться построенная по ним гистограмма. Данные для статистики также требуется хранить в базе данных. Предусмотреть возможность отправки сформированных отчетов на указанный электронный адрес.

Система планирования отдыха

Направление: «Программная инженерия», количество студентов: 2.

Необходимо создать программу, просчитывающую стоимость выбранного отдыха или же подбирающую оптимальный тур на введенную сумму. Система должна иметь обновляемую базу данных туров и соотнесенных с ними экскурсий. Загрузку информации о турах реализовать через Интернет. Основные характеристики каждого тура: список экскурсий, проезд, стоимость питания и проживания. У экскурсий должны быть баллы, отражающие их интерес. Пользователь может либо сам выбрать определенные экскурсии у

понравившегося тура для получения информации о стоимости отдыха. Либо пользователь может ввести количество времени и денег, которые он собирается потратить на отдых, а система подберет ему тур с наиболее интересным списком экскурсий, укладывающийся в стоимость. Результат работы – информация о трех наиболее подходящих вариантах и для окончательно выбранного варианта отчет в форматах pdf, Ms Word, OpenOffice.writer с требуемой пользователю информацией (то есть выводимые в отчет характеристики тура выбирает пользователь), а также информация о стоимости и длительности поездки, разделяя время в пути и время отдыха. Кроме того, должна быть возможность вывода в OpenOffice.calc и MS Excel статистики о популярности тех или иных туров и экскурсий. В Excel, помимо таблицы с данными, должна выводиться построенная по ним гистограмма. Данные для статистики также требуется хранить в базе данных. Предусмотреть возможность отправки сформированных отчетов на указанный электронный адрес.

Система подбора дизайна кухни

Направление: «Программная инженерия», количество студентов: 2.

Реализовать систему подбора дизайна кухни, обеспечивающую возможность для заданного помещения смоделировать расстановку мебели и бытовой техники, а также покрытие пола, потолка и стен выбранным материалом (плитка, обои). Программа должна обеспечить как минимум статический фронтальный просмотр каждой стены комнаты, но в идеале – трехмерный просмотр.

Информация о доступных мебельных блоках и бытовой технике, а также материалах для стен, пола и потолка должна храниться в базе данных вместе с изображением, а также редактироваться, добавляться и удаляться. Размещение модулей организовать методом их перетаскивания.

В результате работы должен формироваться отчет о полной стоимости выбранного пользователем оформления, а также должны формироваться бланки заказа на изготовление мебели, поставку обоев или плитки. Указанные документы должны быть в форматах pdf, MS Word и OpenOffice.writer. Должно быть реализовано хранение списка исполненных заказов с характеристиками клиента и формирование отчета о том, сколько заказов за определенный период и на какую сумму было сделано.

Статистические отчеты формировать в OpenOffice.calc и MS Excel, причем в последнем необходимо строить гистограмму динамики сумм заказов за период. Предусмотреть возможность отправки сформированных отчетов на указанный электронный адрес.

Система автоматизации работы продавца автомобилей

Направление: «Программная инженерия», количество студентов: 3.

Разработать программу, автоматизирующую деятельность по подбору подержанного автомобиля для перепродажи. Программа должна работать с постоянно пополняемой и обновляемой базой данных транспортных средств. Помимо ручного пополнения, данные должны собираться в базу автоматически «роботом» с сайтов avito.ru и auto.ru. Программа должна обеспечить возможность просмотра фото, если они были на сайте-поставщике информации, и перехода к самому объявлению при доступном подключении. Должна быть функция редактирования собранных роботом объявлений и функция автоматической подборки объектов по различным характеристикам для различных клиентов по их пожеланиям.

Информация о клиентах с их пожеланиями также хранится в базе данных и для каждого клиента должен автоматически составляться список рекомендованных ему объектов. Должна быть возможность оформления сделки на продажу того или иного объекта.

В результате формируется договор в форматах pdf, MS Word, OpenOffice.writer о вознаграждении за посреднические услуги.

Кроме того, должна быть возможность сформировать отчет о том, сколько было заключено договоров за определенный период времени и на какую сумму. Отчет выгружается в OpemOficce.calc и MS Excel. В последнем, помимо данных, должна быть гистограмма динамики сумм договоров за период. Предусмотреть возможность отправки сформированных отчетов на указанный электронный адрес.

Автоматизация работы точки продажи косметики

Направление: «Прикладная информатика», количество студентов: 2.

Необходимо разработать программу для автоматизации работы точки продажи косметики. Необходимо вести базу данных косметических средств и базу данных клиентов. Для каждого объекта необходимо выделить ключевые характеристики, на основании которых клиенту может быть подобран набор косметических средств (крема, шампуни и прочее). Каждое косметическое средство, помимо ключевых характеристик, должно иметь краткое текстовое описание и фотографию, получаемую либо с веб-камеры, либо путем загрузки фотографии с компьютера. Должна быть возможность формирования каталогов продукции, рекомендованной для каждого конкретного пользователя, в форматах pdf, OpenOffice.writer и MS Word. В каталоге должна содержаться информация о косметике и ее изображение. Предусмотреть возможность расчета скидок для постоянных клиентов по определяемым пользователем алгоритмам. Должна быть возможность реализации товара и пересчет его доступного количества. По результатам реализации должен формироваться бланк заказа в форматах MS Excel и OpenOffice.calc. Должна быть возможность просмотреть его, распечатать или отправить на электронную почту клиенту.

Система подбора оператора сотовой связи

Направление: «Программная инженерия», количество студентов: 2.

Необходимо разработать программу подбора наиболее выгодного оператора сотовой связи. Подбор может происходить двумя способами.

Первый способ: пользователь сам вводит ключевые параметры: количество отправляемых SMS в месяц, количество звонков на городские номера, на номера других операторов, количество телефонных номеров определенного оператора среди знакомых и т. д. Список ключевых параметров должен быть редактируемым.

Второй способ: анализ распечатки звонков за предыдущий месяц (рассмотреть детализации как минимум двух операторов). Должна быть возможность хранения, редактирования и добавления информации об операторе и его тарифах (причем с возможностью загрузки из сети Интернет). Итоговый отчет должен содержать три наиболее выгодных оператора и предполагаемую сумму на связь за месяц. Отчет сохранять в pdf, OpenOffice.writer и MS Word. Должна быть возможность распечатать его или отправить на указанный электронный адрес.

Система автоматизации работы консультативного центра

Направление: «Программная инженерия», количество студентов: 2.

Разработать систему автоматизации работы юридического консультативного центра. Система должна автоматически регистрировать вопросы клиентов, полученные ею по электронной почте, определять тип вопроса и закреплять его за исполнителем. Должна быть функция поиска подходящих ответов для каждого вопроса (как автоматически, так и вручную), исходя из схожести полученного вопроса и списка обработанных, хранящихся в базе данных.

Система должна предоставлять интерфейс для подготовки исполнителем ответа и автоматический учет времени на подготовку этого ответа (время считается с момента открытия исполнителем вопроса до отправки ответа на электронную почту). Должна быть функция занесения подготовленного ответа в архив и отправка по электронной почте клиенту. Исходя из тематики вопроса должна быть возможность оценки сложности вопроса и введение норматива времени для ответа.

Кроме того, должна быть функция формирования актов-отчетов (pdf,word,writer,calc) на оплату труда в зависимости от стоимости человеко-часа исполнителя и соответствие его времени нормативам. Система должна вести статистику по типам часто задаваемых вопросов и по эффективности работы исполнителей. Статистические отчеты должны выгружаться в Excel в виде таблиц и гистограмм.

Система контроля эффективности работы за компьютером

Направление: «Программная инженерия», количество студентов: 2.

Главной функцией системы является сбор, хранение и анализ статистики работы пользователей компьютеров под управлением операционной системы Windows.

Система должна обеспечивать возможность выполнения следующих функций.

1. Контроль за процессами (приложениями). Отслеживаются:
   - название процесса (приложения);
   - заголовок окна процесса;
   - время начала работы;
   - время завершения работы;
   - общая продолжительность работы процесса.
2. Контроль за активностью пользователей за компьютером.

Отслеживается интенсивность работы пользователя с клавиатурой и мышкой. Если пользователь не проявлял активности в

течение заданного интервала времени, то фиксируется простой компьютера. В результате формируются следующие статистические данные:

- общее время работы;
- время активности;
- время простоя.

3. Контроль за работой с файлами. Отслеживаются:
   - название файла;
   - путь к файлу;
   - время доступа к файлу.

4. Контроль за историей браузера.

5. Просмотр, печать отчета об активном времени работы для каждого пользователя. Формировать отчет можно в виде таблицы (Имя пользователя, активное время, время простоя и в каких приложениях он работал) в pdf, MS Word, OpenOffice.writer, OpenOffice.calc и MS Excel. Кроме того, необходимо иметь возможность отправить сформированный отчет на почту. Отчет в формате Excel проиллюстрировать диаграммами.

Работа должна состоять из службы-клиента, которая собирает статистику и посылает все данные на сервер, и сервера. Сервер полученную от всех клиентов статистику аккумулирует в базе данных и строит требуемые отчеты. Предусмотреть построения отчетов с разрезом не только по отдельным пользователям, но и по определенным группам (должность, отдел, пол, возраст).

Система автоматизации фирмы-грузоперевозчика

Направление: «Прикладная информатика» и «Программная инженерия», количество студентов: 2.

Необходимо реализовать систему, автоматизирующую деятельность фирмы, занимающейся грузоперевозками. Основной

функционал системы для грузоперевозок: рассчитать стоимость транспортировки в зависимости расстояния и веса груза, а также от дополнительных параметров (срочность доставки, ценность груза).

Для расчета расстояния целесообразно использовать сервис Google.Maps. Работа с заказчиками ведется по авансовой системе: заказ принимается по частичной предоплате, и полная оплата фиксируется после выполнения. Оплата ведется безналичными платежами, соответственно, необходимо предусмотреть загрузку платежей из системы клиент-банк. Заказ оформляется в виде договора (Word), счета на предоплату (pdf) и накладной (OpenOffice.writer). По завершении заказа печатается акт выполненных работ (pdf) и счет на оставшуюся сумму (pdf). Реализовать возможность отсылки документов на почту клиентов. Всю история заказов необходимо хранить в базе данных вместе с информацией о клиентах.

Должна быть возможность работы с информацией о клиенте и автоматическая выгрузка их реквизитов в обозначенные выше документы. Ведется также учет имеющихся на предприятии машин и сотрудничающих водителей. Необходимо обеспечить функции приема нового водителя, увольнение существующего, поиск водителя и редактирование информации о нем. Формирование отчета, кто из водителей сколько заработал за месяц по настраиваемому пользователем алгоритму (отчет – зарплатная ведомость в OpenOffice.calc). Кроме того реализовать возможность получения статистики о динамике выручки за период как в табличном виде, так и в виде графиков (MS Excel).

Система автоматизации фирмы, занимающейся пассажирскими перевозками

Направление: «Прикладная информатика» и «Программная инженерия», количество студентов: 2.

Программа должна реализовывать следующие функции: выбор адреса отправления и поиск ближайшей к этому адресу свободной машины в пределах определенного радиуса. Если свободная машина есть, то необходимо выбирать адрес пункта назначения и из расстояния между этими адресами, а также возможной скидки постоянному клиенту рассчитывать стоимость поездки и время, через которое машина подъедет. Необходимо реализовать функцию фиксации освобождения машины и ее текущее положение. Нужно реализовать функцию поиска клиента по номеру телефона и расчета суммы скидки по описываемому пользователем алгоритму.

Необходимо вести списки сотрудничающих водителей, а также обеспечить функции приема нового водителя, увольнение существующего, поиск водителя и редактирование информации о нем. Реализовать функцию формирования отчета, кто из водителей сколько заработал за месяц по настраиваемому пользователем алгоритму (отчет – зарплатная ведомость в pdf, MS Word и OpenOffice.writer).

Необходимо вести список транспортных средств предприятия и привлеченных водителей, отслеживая значимые даты, такие как окончание страховки и прочее, настраиваемое пользователем. При запуске системы необходимо проверять список машин на ключевые даты и за три дня до истечения срока отсылать сообщение на электронную почту водителя и ответственного за состояние машин на предприятии лица. Кроме того реализовать возможность получения статистики о динамике выручки за период как в табличном виде (OpenOffice.calc, MS Excel), так и в виде графиков (MS Excel). Реализовать возможность отправки сформированного отчета на электронную почту.

Система аттестации сотрудников

Направление: «Прикладная информатика» и «Программная инженерия», количество студентов: 2.

Необходимо создать универсальную программу для проведения тестирования сотрудников организаций.

Функционал: возможность загружать файлы с материалом по теме или же создавать файлы с темой внутри самой программы. Возможность загружать или создавать тесты и связывать их с разделами материала (возможность задавать, какие разделы должен пройти испытуемый, прежде чем перейти к тесту).

Вопросы должны быть разных типов: открытые, закрытые, без вариантов. Для каждого вопроса должно настраиваться количество правильных ответов. Организовать свободный переход между вопросами теста, а также ограничения времени тестирования.

Необходимо реализовать возможность проведения тестирования в клиент-серверном варианте с одновременным подключением к одному серверу нескольких тестируемых. Необходимо реализовать хранение списка тестируемых с оценками. Предусмотреть алгоритм расчета оценок для каждого теста по-разному.

Предусмотреть возможность высылки отчета-«электронного сертификата» (pdf, MS Word, OpenOffice.Writer) на почту тестируемого, а также списков с результатами (OpenOffice.calc) на указанный адрес руководителя отдела.

В файле результата необходимо выделять фамилии не прошедших тест (не набравших указанного для теста порога или не ответивших на ключевые вопросы теста).

Кроме того необходимо выводить статистический отчет (MS Excel) с информацией как в табличном виде, так и в виде графика об успеваемости по тестируемым, а также по прохождению тестов.

Система планирования расстановки рабочих мест и расчета их стоимости

Направление: «Прикладная информатика», количество студентов: 2.

Необходимо разработать приложение, выполняющее планировку расстановки рабочих мест в помещении.

В качестве исходных данных система получает план помещения с отмеченными запрещенными для расстановки участками и количество рабочих мест, которые необходимо расположить в данном помещении. В результате своей работы программа предлагает вариант размещения максимально возможного количества мест, соединенных в сеть.

Расстановка должна вестись с соблюдением масштаба и пропорций, а также ГОСТов. Кроме того должна быть предоставлена возможность ручной расстановки в помещениях столов, тумбочек, шкафов и ПЭВМ (АРМ).

В результате работы программа должна выдать отчет о количестве и стоимости требуемого оборудования, кабеля и мебели, а также бланки заказа на все. Отчеты выводятся в форматах MS Excel и OpenOffice.calc, а бланки заказов – в pdf, OpenOffice.writer и MS Word.

В заказе необходимо указать наименование, количество, цену и стоимость каждого заказываемого элемента, а также итоговую сумму всего заказа. Оборудование и мебель должны быть в разных бланках.

Всю информацию о мебели и сетевом оборудовании необходимо хранить в базе данных, обеспечив возможность ее редактирования, добавления и удаления.

Система учет затрат по подразделениям организации

Направление: «Прикладная информатика», количество студентов: 2.

Телефонную связь компании предоставляют три провайдера: «МирТелеком» (международные звонки), «ГородТелеком»

(междугородные звонки), «МестныйТелеком» (звонки по области и городу). У компании три «внешних» номера и 24 «внутренних» номера, разделенных между 6 подразделениями компании. Причем количество подразделений, их номерной состав может изменяться со временем. Каждый месяц провайдеры присылают компании детализацию по звонкам вида: внешний номер, куда звонили, дата, время, сумма. У компании есть своя АТС, которая в конце месяца формирует файл вида: номер департамента, куда звонили, дата, время.

Задача: сформировать отчет (calc,word,pdf,writer) по затратам на связь между подразделениями. Сформированный отчет высылается на почту финансовому директору. Каждому руководителю подразделения высылается уведомление о том, сколько его подразделением было потрачено на связь, с расшифровкой по внутренним номерам и кому этот номер принадлежит.

Кроме того необходимо видеть графики сравнения расхода на связь между подразделениями, годовое изменение расходов у каждого отдельного подразделения и у всех вообще. Графики должны быть на форме и в Excel. Можно просматривать отчеты за любой прошлый период.

Примечание: время провайдеров и время АТС может отличаться на две минуты.

Автоматизация работы биржи труда

Направление: «Программная инженерия», количество студентов: 2.

Есть резюме и запросы соискателей, есть вакансии и запросы работодателей. Необходимо сформировать для работодателя: отчет о предполагаемых работниках, подходящих под его требования. Для работника: список вакансий, подходящий под его навыки. Есть возможность добавления вакансии от компании, редактирования информации, добавление резюме от соискателя, редактирования резюме.

Кроме того должна быть возможность создания и рассылки электронных сообщений, а также выгрузка отчетов в форматах pdf, MS Word, OpenOfice.writer.

Предусмотреть добавление информации в базу данных не только ручным способом, но и путем автоматической загрузки из сети Интернет. Необходимо формировать статистические отчеты о наиболее востребованных специальностях, зарплатах по отраслям в форматах MS Excel (с диаграммами) и OpenOfice.calc.

Автоматизация работы отдела продаж

Направление: «Прикладная информатика», количество студентов: 2.

Необходимо разработать программу для автоматизации работы менеджера отдела продаж. Функции программы: хранение и добавление информации по клиентам, заполнение форм договоров и приложений. Формы договоров хранятся в файлах MS Word или OpenOffice.writer.

Необходимо заполнить их (дополнив информацией о клиенте), сохранить заполненную копию в текущем формате и перевести в формат pdf. Заполненные формы договоров можно просмотреть, распечатать или отправить клиенту по электронной почте для ознакомления.

Добавить функцию пакетной конвертации документов в формат pdf, а также функцию сборки нескольких файлов (причем разных форматов: jpg, pdf, word, writer) в один. Сформировать отчет в OpenOffice.calc и Excel о том, какой менеджер за определенный период сколько договоров заключил, на какую сумму и с кем. В Excel данные сопроводить диаграммой.

Автоматизация работы точки розничных продаж

Направление: «Прикладная информатика», количество студентов: 2.

Необходимо разработать программу для учета закупок и реализаций товаров. Ежедневно приходит файл о продажах и периодически – о закупках. Программа должна отслеживать, сколько номенклатуры осталось на складе и когда это количество приближается к некоему пороговому значению – формировать файлы заказов продукции у поставщиков и вместе с этим счета на оплату. Файл загрузок имеет произвольный формат, файлы с заказами продукции или счетами поставщикам – отчет в форматах pdf, Word,OpenOffice.writer. Должна быть возможность отправить сформированные файлы на электронную почту. Должна быть возможность добавить информацию о новом поставщике. Информация о новой номенклатуре грузится из файла закупок. Необходимо формировать статистические отчеты о популярности товаров и поставщиков в форматах Excel и OpenOffice.calc. В первом случае по данным построить диаграммы.

Автоматизация работы кафе с доставкой

Направление: «Прикладная информатика», количество студентов: 2.

Программа с графическим интерфейсом, интуитивно понятная для приема заказов. Работа строится следующим образом: клиент звонит, оператор выбирает уже существующего клиента, или заполняет контактную и адресную информацию на нового клиента.

Также он формирует заказ, смотрит общую сумму заказа и размер скидки. После оформления заказа распечатывается его бланк в форматах pdf, Word,OpenOffice.writer.

Новые клиенты, которые добавляются в эту программу, хранятся в базе данных вместе с историей их заказов. Клиенты связаны по реферальному принципу в три уровня: при заказе процент от суммы заказа в виде бонусных баллов падает на вышестоящего по иерархии, 7 %, 2 % и 1 %. Для этого каждому клиенту при

регистрации прикрепляется индивидуальный номер карты постоянного клиента.

При заказе нужна возможность расплатиться частично или полностью бонусными баллами. Программа должна уметь формировать отчеты о результате работы кафе за день-неделю-месяц-квартал, выводить на печать форму счета, показывать статистическую информацию о том, что наиболее часто и что наиболее редко заказывают. Статистические отчеты выгружать в форматах Excel и OpenOffice.calc. В первом случае по данным строить диаграммы.

Также необходимое условие – вопрос составного принципа продуктов – чтобы была возможность первоначально вносить в программу продукты: рис, лосось, сыр и т. д., – потом из этих продуктов можно было формировать предложение; к примеру, суши лосось: рис (21 грамм) + лосось (48 граммов) и т. д. – для будущей возможности проведения инвентаризации.

Основное требование – удобство поиска старого клиента по номеру телефона, имени и адресу и добавление нового, а также удобство формирования заказа, т. к. на это обычно очень мало времени.

Автоматизация сети торговых точек

Направление: «Прикладная информатика», количество студентов: 2.

Ваша фирма занимается реализацией товаров, у вас есть центральный офис со складом и сеть филиалов со своими складами. Товары хранятся на складе у каждого филиала, а туда отпускаются с общего склада.

Ведутся списки

Менеджеров (по филиалам)

Товаров

Поставщиков

Филиалов

Соответственно, есть возможность добавить менеджера, товар, поставщика.

Строятся отчеты в форматах pdf, Excel, OpenOffice.calc, Word,OpenOffice.writer:

Продаваемость товаров

Каких поставщиков определенных товаров берут больше

Продаваемость по филиалам

Результативность менеджеров

В Excel по данным строятся диаграммы.

Для центра: ежедневно рассылается в филиалы список товаров для пополнения склада (если это необходимо). Это количество определяется исходя из общего количества товаров на складе у филиала, заявок филиалов на товары. От филиалов ежедневно получают файлы об операциях вида: товар-поставщик-менеджер-продано количество и список заявок на товары, которые необходимо пополнить. Филиальная система сама мониторит, сколько осталось товаров, и формирует заявку на пополнение. Также заявку можно сформировать вручную.

Периодически загружаются от поставщиков прайсы. Каждый день центральная программа мониторит, сколько осталось товара на складе, и при достижении какого-то порога предлагает его заказать у лучшего поставщика. Лучший поставщик определяется по ценам и популярности у потребителей.

По менеджерам: у каждого менеджера есть плановый показатель продажи товара и фактический показатель продажи товара. Ежемесячно строится отчет результативности менеджеров (процент выполнения плана), и в зависимости от него менеджерам начисляется зарплата. Система ведет списки постоянных покупателей и использует систему скидок. Если покупатель хочет купить какой-либо товар, а филиал не может его продать, то он формирует заявку на

товар и вечером отсылает ее в центр вместе с остальными отчетами или смотрит, в каком филиале есть этот товар и либо сообщает об этом пользователю, либо формирует акт перенаправления и отсылает его этому филиалу, чтобы тот отослал ему требуемый товар на следующий день. Всю работу с электронной почтой и файлами полностью автоматизировать (программа сама получает почту и определяет, что с ней делать).

Автоматизация деятельности ателье по пошиву одежды

Направление: «Прикладная информатика», количество студентов: 2.

Ваша задача – реализовать программный продукт, который сможет помогать комбинировать одежду, подбирая образ. Пользователи должны иметь возможность сами загрузить изображения одежды, соответствующие оговоренному формату. То же самое может сделать администратор системы, для предоставления пользователю базы данных имеющихся вариантов, например, которые уже готовы и их можно купить или можно сшить. Кроме изображений, вы должны иметь сведения о размере, возможных цветах (соответственно, несколько изображений) и стоимости. Пользователь должен иметь возможность ввести свои параметры для того, чтобы система рассчитала его размер, и, соответственно, при подборе одежды это учитывалось. Программа должна иметь возможность автоматического подбора образов по заданным пользователем характеристикам. Одежда «надевается» на манекен, соответственно, вы должны иметь возможность загрузить в качестве лица манекена фотографию пользователя в оговоренном формате. Предусмотреть систему скидок постоянным клиентам и возможность просмотра, что было ими ранее приобретено. Необходимо иметь возможность распечатать счет на приобретение или пошив одежды (Word, writer, pdf), а также отчет о том, сколько одежды было продано и пошито за определенный период и на какую сумму (Excel, calc).

Автоматизация работы предприятия по покрытию кровли

Направление: «Прикладная информатика», количество студентов: 2.

Есть каталог кровельных покрытий и тротуарной плитки с фотографиями. Необходимо выбирать понравившуюся модель заказчику по его пожеланиям (материал, цвет и т. д.) и рассчитывать стоимость покрытия исходя из введенных размеров. Формировать прайс-лист (в OpenOffice.calc) с указанием, что есть на складе, в каком размере и по какой цене, а также документы-акты продажи (Word, pdf, writer). Строить отчеты с диаграммами в Excel о продажах за период. Реализовать отправку их на почту.

Подбор компьютера по прайсам компьютерных магазинов

Направление: «Программная инженерия», количество студентов: 2

Вы работаете с прайсами различных компаний («НИКС», «DNS» и любой другой на ваше усмотрение). Задача: сформировать самую дешевую комплектацию компьютера из их комплектующих по запросу пользователя. Комплектующие должны подходить друг другу (например, следить за тем, что если материнская плата не поддерживает SATA, чтобы SATA винчестера и не было). Пользователь может указать для каждого комплектующего предполагаемую цену, производителя и прочие характеристики. Так как «НИКС» работает с ценами в валюте, то необходимо взять из интернета самый свежий курс доллара или же, при отсутствии доступа к интернету, взять ранее. Обеспечить хранение курса валют за прошлые периоды и, соответственно, возможность обработки прайсов прошлых периодов с актуальным курсом. По желанию пользователя сформированный список комплектующих может быть отправлен к нему на электронную почту. Кроме того иметь возможность сравнения стоимости одних и тех же комплектующих у разных компаний. Иметь возможность, кроме выбора файла прайса на локальном компьютере, загружать его свежую версию из Интернета.

Предпочтительно реализовать возможность работать не только с прайсами, но и просто с сайтами компьютерных магазинов. Отчет о комплектации выгрузить в файлы типа pdf, Excel, OpenOffice.calc, Word, OpenOffice.writer.

Банковский калькулятор

Направление: «Программная инженерия», количество студентов: 2.

Калькулятор состоит из двух частей. Первая: калькулятор кредитов, позволяющий выбрать наиболее подходящий кредит для пользователя. Программа должна позволять осуществлять следующие действия.

1. Просмотреть список банков и добавить новый.
2. Добавить информацию о новом кредите.
3. Просмотреть список кредитов для каждого банка.
4. Выбрать наилучший кредит по двум условиям: либо как минимум суммы переплаты, либо по сумме ежемесячного платежа.
5. Сформировать и отправить на почту отчет о выбранном кредите в форматах pdf, Excel, OpenOffice.calc, Word, OpenOffice.writer.
6. Загрузить/выгрузить список банков-кредитов (пакетная загрузка данных из сети Интернет).

Вторая часть: калькулятор вкладов, позволяющий выбрать наиболее подходящий вклад для пользователя. Программа должна позволять осуществлять следующие действия.

1. Добавить информацию о новом вкладе.
2. Просмотреть список вкладов для каждого банка.
3. Выбрать наилучший вклад по двум условиям: либо как максимум суммы за заданный срок размещения денег, либо как минимум срока, по достижению которого сформируется требуемая сумма.

4. Сформировать и отправить на почту отчет о выбранном вкладе в форматах pdf, Excel, OpenOffice.calc, Word, OpenOffice.writer.

5. Загрузить/выгрузить список банков-вкладов (пакетная загрузка данных из сети Интернет).

Программа автоматизации работы абонентского отдела

Направление: «Прикладная информатика», количество студентов: 2.

С программой работает несколько менеджеров. Для каждого менеджера есть свой список клиентов с неким начальным балансом (долг или аванс). Ежемесячно вам поступает информация о том, каких услуг и на какую сумму потреблял клиент, а также сколько он заплатил.

Задача: для каждого клиента сформировать пакет документов (ваши отчеты (Word, pdf, writer)): акт, счет-фактура, счет и уведомление о задолженности, если на конец месяца клиент имеет отрицательный баланс.

Необходимо нумеровать документы общим списком, а не для каждого менеджера отдельно.

Программа должна позволять массово распечатать и/или отправить каждому на электронную почту.

Обеспечить возможность хранить архив документов и распечатывать/отправлять на почту отдельные документы за прошлый период.

Также обеспечить возможность рассылки всем или определенным клиентам определенных уведомлений по электронной почте: просто текста или прикрепленного файла. Ежемесячно выгружать отчет по клиентам с задолженностью (Excel, calc).

Учет заработной платы агентов по продажам

Направление: «Прикладная информатика», количество студентов: 2.

Необходимо создать программу учета заработной платы агентов по продажам услуг по сдельно-премиальной форме.

Суть работы агентов: они собирают заявки на услуги, как только по их заявке происходит реализация услуги (проплата от клиента), заявка засчитывается как фактическая.

Существует ежемесячный план по заявкам. Если агент не выполняет план, то его фактические заявки оцениваются по одному тарифу, если выполняет – по другому, плюс определенная сумма ежемесячной премии. Если агент выполняет план за квартал, то ему начисляется дополнительная премия. При этом агент должен отработать не менее 17 дней в месяц. Для тех, кто не отрабатывает это количество дней, стоимость заявки ниже.

Есть возможность получать отчет о наиболее и наименее результативных агентах. Если агент является результативным более чем 3 месяца от начала работы, он получает статус старшего агента и соответственно другую оплату.

Предусмотреть возможность ведения информации об агенте и заявках, а также привязку заявки к агенту.

По итогам месяца должен формироваться отчет в форматах pdf, Excel, OpenOffice.calc, Word,OpenOffice.writer с данными о том, сколько денег, какой агент должен получить.

Должна быть возможность просмотреть этот отчет, распечатать его или отослать на электронную почту.

Автоматизация работы приемной комиссии

Направление: «Прикладная информатика», количество студентов: 2.

Необходимо создать программу для учета абитуриентов вуза. Должна добавляться, редактироваться и храниться информация об абитуриентах, их результатах по ЕГЭ, а также информация о специальностях вуза, требуемых экзаменах на каждую специальность и количество мест на ней.

Программа должна сформировать три отчета: «первую волну», «вторую волну» и «третью волну». Каждый раз программа смотрит, сколько студентов зачисляется на специальности.

Зачисление происходит только среди студентов, принесших в приемную комиссию подлинники с наивысшими баллами по требуемым экзаменам. Если после «первой волны» остались еще места, то через некоторое время формируется следующий отчет и т. д. Отчет должен быть в файлах типа pdf, Excel, OpenOffice.calc, Word, OpenOffice.writer и содержать списки зачисленных на специальности студентов с баллами. Отчет можно просмотреть, распечатать или отправить на электронную почту.

4.2. Начальное проектирование приложений

Многие опытные программисты говорят: «Хорошая программа пишется на листочке». Это, действительно, так. Прежде чем начать создавать код, необходимо четко себе представлять, а что вы хотите сделать, ибо, как гласит «Третий закон Грира»: «Компьютерная программа делает то, что ты приказал ей сделать, а не то, что ты на самом деле хотел».

Исходя из этого, в создании любой хорошей программы неизбежен этап проектирования. Для больших систем, основанных на объектно-ориентированном подходе, этот этап заключается в определении функционала системы, определении классов, которые будут участвовать в реализации системы, а также в разработке различных сценариев действия системы. Понятно, что проект системы – это отдельная и весьма трудоемкая работа, которая, правда, не всегда выполняется от начала до конца. Более того, в условиях ограниченности времени на многих не очень серьезных проектах этап вовсе опускается. Для опытных разработчиков, решающих типовую задачу, это допустимо, но для начинающих программистов о проектировании забывать не следует. Мы рассмотрим техники

построения начального проекта приложения, который позволит четко представить, что конкретно необходимо написать для реализации системы. Как было сказано выше, для понимания системы нам необходимо описать три вещи: функционал, реализующие его классы, а также сценарии их взаимодействия. Для этого мы воспользуемся возможностями языка UML, а также техникой создания ER-диаграмм.

Unified Modeling Language (UML) представляет собой формальный искусственный язык графического моделирования. Его разработчиками являются Гради Буч, Ивар Якобсон и Джеймс Рамбо. Язык описывает правила построения различных видов диаграмм, которые используются при проектировании системы. Из всего их многообразия мы рассмотрим две: диаграмму вариантов использования (или диаграмму прецедентов) и диаграмму классов. Основное назначение первой диаграммы – определить, как именно система будет взаимодействовать с пользователем или другими системами. Иными словами, какие возможности предоставляет система внешнему миру, то есть ее основной функционал.

Диаграмма содержит два вида ключевых элементов: эктор и прецедент. Эктор – множество ролей, исполняемых взаимодействующей с системой сущностью. Графически этот элемент диаграммы изображается человечком. Прецедент – описание взаимодействия между эктором и системой через последовательность неких действий, приводящих к желаемому результату. Графически элемент изображается эллипсом.

Для пояснения рассмотрим построение диаграммы вариантов использования (или по-другому диаграммы прецедентов) для описания функционала системы проведения тестирования. Пусть необходимо создать универсальную программу для проведения тестирования сотрудников организаций. Система должна позволять загружать файлы с материалом по теме или же создавать файлы с

темой внутри самой программы. Возможность загружать или создавать тесты и связывать их с разделами материала (возможность задавать, какие разделы должен пройти испытуемый, прежде чем перейти к тесту). Необходимо реализовать возможность проведения тестирования и хранение списка тестируемых с оценками. Предусмотреть возможность высылки отчета-«электронного сертификата» в форматах pdf, MS Word, OpenOffice.Writer на почту тестируемого, а также списков с результатами в формате OpenOffice.calc на указанный адрес руководителя отдела. Кроме того необходимо выводить статистический отчет (MS Excel) с информацией как в табличном виде, так и в виде графика об успеваемости тестируемых, а также по прохождению тестов.

Первое, что мы должны сделать для построения диаграммы, – выделить из текста постановки задачи словосочетания «глагол+его дополнение»:

- загружать файлы с материалом;
- создавать файлы с темой;
- загружать тесты;
- создавать тесты;
- связывать тесты с материалом;
- проведение тестирования;
- хранение списка тестируемых с оценками;
- отсылка отчетов на почту;
- вывод статистического отчета.

Теперь мы должны превратить это в диаграмму, явно указав характер взаимодействия пользователя с системой для каждого действия. Результат представлен на рисунке 4.1.

Мы видим, что функции в списке и на диаграмме совпадают не полностью. Это объясняется тем, что в процессе проектирования всегда выделяются новые необходимые функции, скрытые в исходной

постановке задачи. Например, если говорится, что можно добавить тест, значит в любом случае в программе будет список этих тестов, и необходимо предусмотреть функции работы с ним. Кроме того необходимо учесть, что данная диаграмма является первым этапом проектирования, и дальнейшее ее уточнение неизбежно.

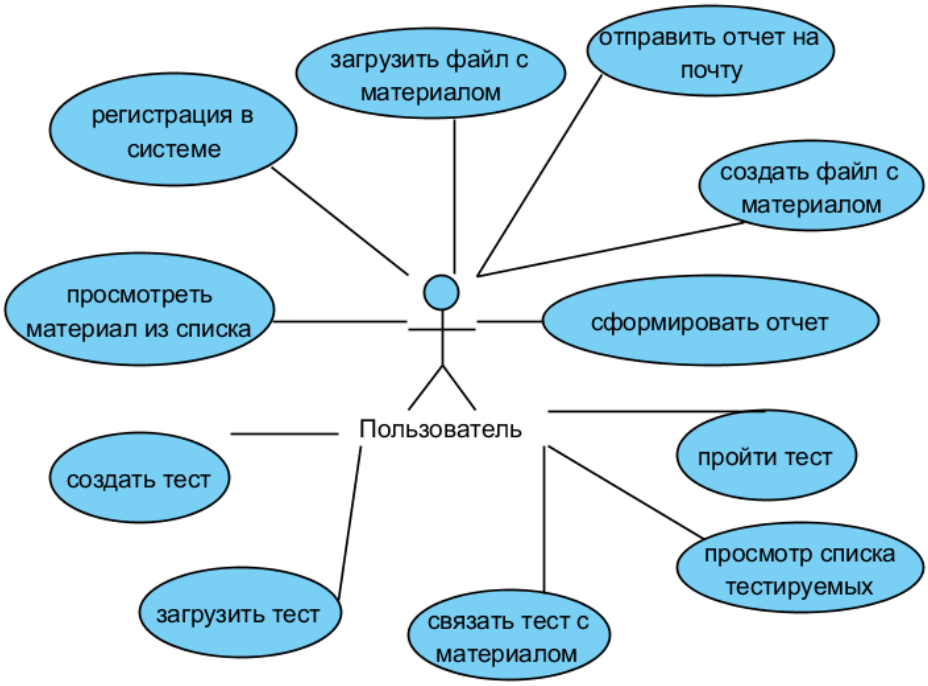

Рисунок 4.1. Диаграмма прецедентов.

После того как вы обозначили примерное множество функций системы, можно переходить к проектированию классов, обеспечивающих их реализацию.

В объектно-ориентированном подходе принято выделять три вида классов: сущности, управляющие и граничные классы. Рассмотрим их подробнее. Классы-сущности представляют собой абстракции реальной предметной области, обычно не зависят от окружения и могут использоваться в различных приложениях. Обычно это те классы, которые требуются системе для выполнения некоторых обязанностей. Их выделение из исходной постановки задачи начинается с построения списка существительных, с которыми

работает система и которых можно встретить в предметной области задачи. Для нашего примера это:

- материал;
- тест;
- вопрос;
- ответ;
- тестируемый;
- оценка;
- почта;
- отчет.

Теперь необходимо профильтровать полученный список. Класс-сущность обычно используется для долговременного хранения данных, имеет набор ключевых характеристик и участвует в реализации нескольких функций системы. Приведенные параметры отсекают два последних существительных из списка. Электронная почта в данном случае – всего лишь строка с адресом, являющаяся или характеристикой тестируемого, или неким вводимым значением. Отчет – срез данных, которые можно получить в любое время. Никакой самостоятельной смысловой нагрузки он не несет.

В большинстве случаев объекты классов-сущностей при работе программы хранятся в базе данных, поэтому после того, как мы выделили список классов-сущностей, необходимо определить их характеристики.

Материал: название, текст, список тестов.

Тест: название, список вопросов.

Вопрос: текст, список ответов.

Ответ: текст, пометка правильности.

Тестируемый: логин, пароль, список материалов, список тестов.

Оценка: тестируемый, тест, значение.

Теперь по этому описанию нам необходимо спроектировать базу данных, используя технику построения ER-диаграмм. Напомним, что это модель сущность-связь, позволяющая описывать концептуальную схему предметной области. С ее помощью мы упростим проектирование базы данных. Первое что нам необходимо сделать, – определить, какие между сущностями будут связи.

Из описания мы видим следующее: материал связан с тестом и тестируемым, тест – с вопросом, оценкой, тестируемым и материалом, вопрос связан с ответом, тестируемый – с материалом, оценкой и тестом, оценка связана с тестом и тестируемым. Теперь мы можем построить первую версию ER-диаграммы. Ее вид представлен на рисунке 4.2.

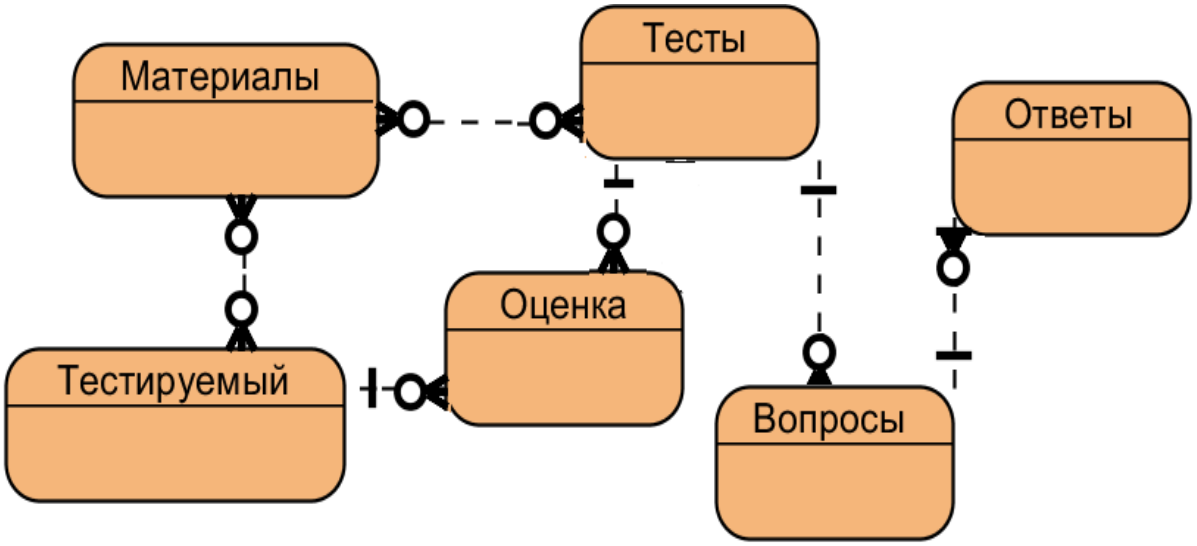

Рисунок 4.2. Первый этап построения ER-диаграммы

Мы видим, что на данной диаграмме убраны избыточные связи. Теперь нам необходимо уточнить характер оставшихся связей и связи типа «многие ко многим» привести к виду «один ко многим» с помощью дополнительных сущностей-связей. После того как мы это сделали, мы получим диаграмму, представленную на рисунке 4.3.

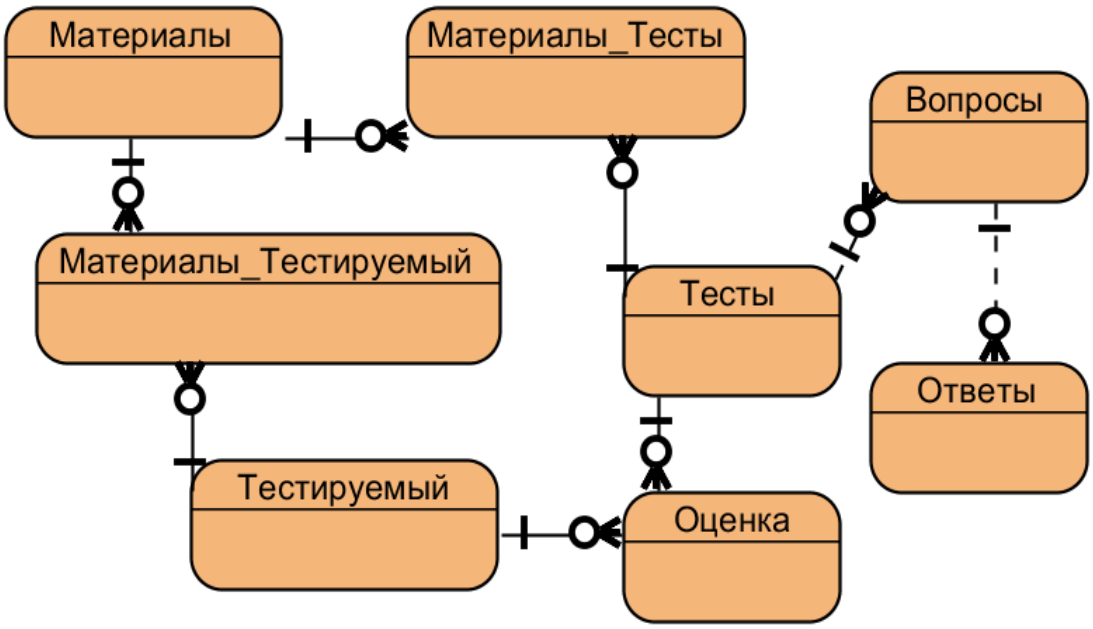

Рисунок 4.3. Финальная ER-диаграмма

Мы видим, что все связи имеют вид «один ко многим».

Теперь построим диаграмму классов-сущностей, в которой опишем их поля и методы. По правилам оформления диаграмм классов UML класс представляется в виде прямоугольника, имеющего три горизонтальные секции: имя, поля и методы. Связанные классы соединяются между собой линиями. Но так как классы-сущности в основном впоследствии будут представлены таблицами в базе данных, то раздел методов на данном этапе можно игнорировать. Итоговый вид диаграммы классов, отображающей сущности системы, представлен на рисунке 4.4. Обратите внимание на «искусственные» характеристики объектов: идентификаторы. Они необходимы для обеспечения связей таблиц в базе данных.

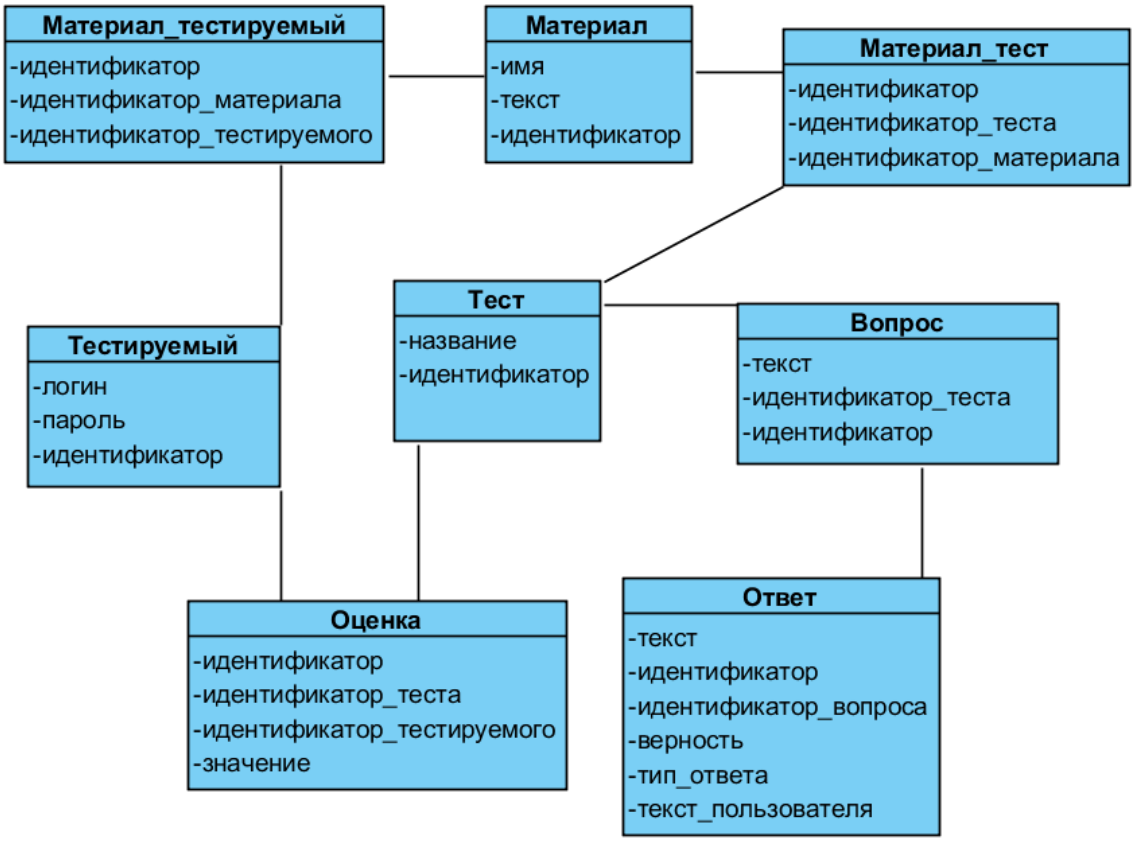

Рисунок 4.4. Финальная диаграмма классов-сущностей

Теперь перейдем к следующему виду классов, к управляющим классам. Их задача – реализация поведения прецедентов и определение его динамики. То есть по сути это то, что обеспечивает всю работу вашей программы, являясь ее ядром, внутренним механизмом. В противоположность классам-сущностям на диаграмме эти классы отображаются без полей, но с методами. Управляющие классы обычно выделяются следующим образом. Первым шагом проектируется по одному классу на каждый прецедент, а затем классы, манипулирующие одними и теми же сущностями или выполняющие схожие действия, объединяются в один.

Для нашего примера первоначальный список будет следующим:

- регистратор;
- загрузчик файла с материалом;
- загрузчик файла с тестом;

- создатель материала;
- создатель теста;
- построитель отчетов;
- тестировщик;
- редактор теста;
- редактор материала;
- почтовик;
- менеджер базы данных;
- класс для работы с тестируемыми.

Объединяя классы, получаем итоговый список:

- класс для работы с тестируемыми;
- класс для работы с тестами;
- класс для работы с материалами;
- построитель отчетов;
- почтовик;
- менеджер базы данных.

Теперь построим диаграмму классов, определив методы для каждого класса. Итоговый результат изображен на рисунке 4.5.

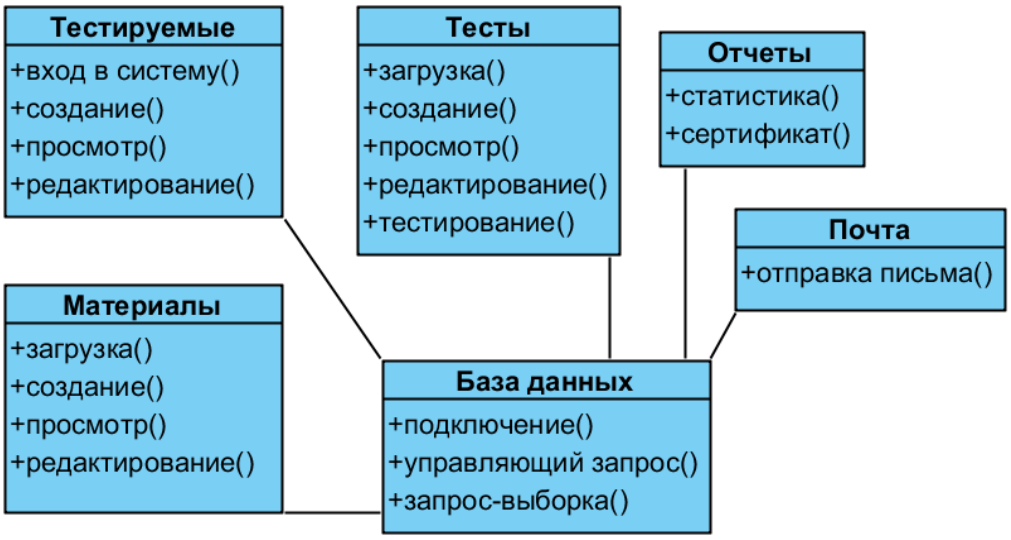

Рисунок 4.5. Финальная диаграмма управляющих классов

Теперь нам осталось рассмотреть последний вид классов – граничные классы. Их задача – обеспечение взаимодействия между окружающей средой и внутренними элементами системы. В некоторых источниках их называют интерфейсами, так как по сути они предоставляют интерфейс для пользователя или другой системы. Для проектирования этих классов мы изучаем методы управляющих классов и выделяем список тех, в которых требуется непосредственное участие пользователя или другой системы. Для каждого такого метода, как правило, проектируется отдельная форма, но затем универсальные формы объединяются в одну. Таким образом, итоговый список граничных классов для нашего примера будет следующим:

- основная форма с меню;
- вход в систему;
- регистрация в системе и редактирование информации;
- список тестируемых, материалов или тестов;
- загрузка материала или теста;
- создание, редактирование и просмотр материала;
- создание и редактирование теста;
- тестирование;
- форма окончания тестирования с выводом сертификата;
- отправка письма;
- подключение к базе данных.

Теперь ваша задача – определить первоначальную взаимосвязь форм. То есть какая форма, откуда будет вызываться. Для этого можно воспользоваться техникой построения диаграммы классов, связав взаимодействующие формы линиями.

Логика связей строится из соображений здравого смысла и удобства для пользователей. Каких-то правил или рекомендаций в

этом случае нет, разве что попробуйте представить, как бы с вашей точки зрения выглядела работающая программа.

Для нашего примера одним из возможных вариантов взаимодействия форм может быть вариант, представленный на рисунке 4.6. Рекомендуется для каждой формы написать примерный список визуальных элементов, которые должны быть на ней размещены. Но этот шаг можно опустить, так как в процессе реализации интерфейс в большинстве случаев все равно меняется.

Рисунок 4.6. Финальная диаграмма граничных классов

После того как вы получили списки классов сущностей, управляющих и граничных классов, вы можете переходить непосредственно к реализации вашей системы. Но необходимо учитывать, что это результаты только первичного проектирования, и в процессе работы над системой имеющаяся классовая модель может модифицироваться. При парной разработке проекта рекомендуется следующее деление: один разработчик работает с граничными

классами, второй – с классами-сущностями. Управляющие классы делятся поровну исходя из объема их функционала.

Контрольные вопросы к разделу 4

1. Перечислите диаграммы, используемые для начального проектирования приложений, и выполните с помощью них проектирование приложения для любой из представленных задач.
2. Перечислите основные элементы каждой из диаграмм.
3. Что такое класс-сущность?
4. Что такое граничный класс?
5. Что такое управляющий класс?

ЗАКЛЮЧЕНИЕ

Автоматизация бизнес-процессов организаций – одна из наиболее частых задач, стоящих перед специалистами ИТ-сферы. Конечно, бесспорным лидером в решении подобных проблем является 1С, но иногда его использование может быть ограничено или нерентабельно. Поэтому целесообразно знать возможности альтернативных технологий.

В данном пособии мы рассмотрели технологии автоматизации бизнес-процессов предприятий, совместимые с платформой .Net, а также примеры задач автоматизации и приемы объектно-ориентированного проектирования приложений. В ходе работы над пособием был изучен и переработан большой объем материала, а также изложены авторские разработки последних трех лет. Основной целью работы было освещение базовых и вместе с тем наиболее важных аспектов существующих технологий, которые студентам, возможно, придется использовать в своей будущей работе.

Пособие обобщает наиболее важную информацию о самых часто используемых технологиях, представляя ее в удобной для понимания и использования форме. Приведенные в библиографическом списке источники литературы ориентируют читателя на более подробное изложение некоторых вопросов.

Приведенные примеры задач атвоматизации представляют собой адаптации реальных производственных проектов. В ходе работы над ними студенты должны научиться разрабатывать программное обеспечение командой, строить проект системы и распределять обязанности по его реализации между собой. Рассмотренные примеры задач и алгоритмы решения некоторых из них позволяют студентам составить некоторое представление о том, что их может ожидать в реальной рабочей практике.

ГЛОССАРИЙ

Common Language Runtime (CLR) – среда выполнения программ на платформе .Net.

Граничные классы – разновидность стереотипов классов, используемая в объектно-ориентированном проектировании. Представляют собой классы, основной функцией которых является функция организации взаимодействия системы с окружающей средой (пользователем или другими системами).

Десериализация – процесс восстановления объекта из битовой последовательности (из файла).

Классы-сущности – разновидность стереотипов классов, используемая в объектно-ориентированном проектировании. Представляют собой сущности реального мира или элементы системы.

Многопоточность – технология запуска дополнительных потоков в рамках процесса приложения.

Неуправляемый код – программный код, работающий не под управлением CLR.

Технология отражения – технология, позволяющая организовывать динамическую работу с объектами в оперативной памяти.

Платформенный вызов – это служба, позволяющая управляемому коду вызывать неуправляемые функции, реализованные в библиотеках динамической компоновки.

Поток – независимый путь исполнения программы, способный одновременно выполняться с другими потоками.

Потокобезопасный код – программный код, лишенный неопределенностей при любых вариантах многопоточной работы.

Сериализация – процесс перевода структуры объекта в последовательность битов на диске (в файл).

Управляемый код – программный код, работающий не под управлением CLR.

Управляющие классы – разновидность стереотипов классов, используемая в объектно-ориентированном проектировании. Представляют собой классы, реализующие основной функционал системы.

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

CLI-сборка, 37
ComponentContext, 39
DataContext, 27
DataSet, 17
Desktop, 39
Document Object Model, 133
ER-диаграмма, 192
LINQ, 27
Mapping, 27
Portable Document Format, 83
ServiceManager, 39
Unified Modeling Language, 188
WCF-контракт, 112
WCF-служба, 110
WCF-хост, 110
Windows Communication Foundation, 109
XDispatchHelper, 41
XFrame, 41
XML Path Language, 133
Граничные классы, 196
Десериализация, 9
Диаграмма классов, 188
Диаграмма прецедентов, 188
Журнал событий, 153
Классы-сущности, 190
Клиент-Банк, 147
Конечная точка, 115
Многопоточность, 94
Основной поток, 104
Отражение, 47
Платформенный вызов, 155
Поток, 94
Прокси-класс, 111
Реестр Windows, 160
Сервер POP3, 127
Сервер SMTP, 121
Сериализация, 9
Синхронизация потоков, 97
Управляющие классы, 194
Фоновый поток, 104
Электронно-цифровая подпись, 148

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Ватсон, Б. C# 4.0 на примерах / Б. Ватсон. – СПб. : БХВ-Петербург, 2011. – 608 с. : ил.

2. Воронина, В. В. Разработка pdf-конвертера документов OpenOffice.writer / В. В. Воронина, Е. В. Дементьев // Вузовская наука в современных условиях : сборник материалов 47-й научно-технической конференции. Ч 2. – Ульяновск : УлГТУ, 2013. – С. 228–230.

3. Воронина, В. В. Разработка парсера сайта Avito.ru / В. В. Воронина, М. А. Липинский // Вузовская наука в современных условиях : сборник материалов 47-й научно-технической конференции. Ч 2. – Ульяновск : УлГТУ, 2013. – С. 219–221.

4. Джозеф, Р. LINQ: интегрированный язык запросов в C# 2008 для профессионалов / перевод с англ. / Р. Джозеф. – М. : ООО «И.Д. Вильямс», 2008. – 560 с. : ил.

5. API маршрутов Google. Документация для разработчиков. – https://developers.google.com/maps/documentation/directions/?hl=ru#StatusCodes (дата обращения: 28.08.2013).

6. Галушко, М. Маппинг классов на схему БД [Серия статей по SQL CE] / М. Галушко. – № 3. – http://wp7rocks.com/posts/details/2230 (дата обращения: 28.08.2013).

7. Жеребцов, А. LINQ и ADO.NET: различные подходы к организации доступа к данным / А. Жеребцов. – http://www.yavnauke.ru/blogs/linq-i-ado-net/linq-i-ado-net-razlichnye-podhody-k-organizaci-dostupa-k-danym.html (дата обращения: 28.08.2013).

8. Интеграция системы .iBank 2. с программой 1с. Руководство пользователя. Версия 2.0.22. – http://www.bifit.com/download/docs/Corporate_1C-Integration_Guide.pdf (дата обращения: 28.08.2013).

9. Кирюшкин, А. Работа с потоками в C# / А. Кирюшкин. – http://www.rsdn.ru/article/dotnet/CSThreading1.xml (дата обращения: 28.08.2013).

10. Материалы SQL-форума. – http://www.sql.ru/forum/415013/c-openoffice (дата обращения: 28.08.2013).

11. Материалы библиотеки msdn. – www.msdn.microsoft.com (дата обращения: 28.08.2013).

12. Материалы сайта «ITextSharp». – http://itextpdf.com (дата обращения: 28.08.2013).

13. Материалы свободной энциклопедии «Википедиа». – http://ru.wikipedia.org/wiki/ (дата обращения: 28.08.2013).

14. Материалы формуа «CyberForum». – http://www.cyberforum.ru/csharp-beginners/thread216373.html (дата обращения: 28.08.2013).

15. Материалы форума «gotdotnet». –http://www.gotdotnet.ru/forums/9/117093/ (дата обращения: 28.08.2013).

16. Молчанов, В. Работа с серверами автоматизации Word и Excel в Visual Studio .Net / В. Молчанов. – http://wladm.narod.ru/C_Sharp/componentbegin.html (дата обращения: 28.08.2013).

17. Чернов, Д. Работа с OpenOffice на C# / Д. Чернов. – http://life-sat.blogspot.ru/2007/06/openoffice-c-net.html (дата обращения: 28.08.2013).