# БАЗЫ ДАННЫХ

## для карманного персонального компьютера

# Pocket PC

Л.А. Родигин

УЧЕБНО-МЕТОДИЧЕСКОЕ
ПОСОБИЕ

КНОРУС

**Родигин Л.А.**

# БАЗЫ ДАННЫХ
# ДЛЯ КАРМАННОГО ПЕРСОНАЛЬНОГО КОМПЬЮТЕРА
# POCKET PC


**Практикум по прикладной информатике**

**Москва**

**2007**

Рецензенты:

Книга представляет собой учебное пособие по разработке и созданию баз данных на основе использования технологии Microsoft ADO.NET и языка C#.NET для карманных персональных компьютеров Pocket PC. Пособие предназначается в качестве методической помощи студентам, изучающим дисциплину «Базы данных», а также всем пользователям карманных персоналок и коммуникаторов, желающим расширить функциональность собственных SmartDevice.



**Содержание**

## ВВЕДЕНИЕ

Риторический вопрос – зачем нужен КПК - карманный персональный компьютер? Если без телефона, то – не знаю. А вот с телефоном, Интернетом, да еще и GPS, тогда понятно – все свое ношу и вожу с собой. Пакет операционной системы (ОС) Windows Mobile [8], а именно она превращает КПК в Pocket PC, имеет целый набор замечательных встроенных приложений, некоторые из которых Вам может и не всегда нужны, а некоторые насущные - отсутствуют. К числу последних относятся программы баз данных, предметная область которых настолько широка, что не имеет особого смысла дополнение состава ОС подобными приложениями. В настольных компьютерных системах предусмотрены пакеты программирования баз данных, но программирование в КПК изначально подвергнуто сомнению. Причем не столько из-за мощности самих устройств, сколько из-за сомнительной эргономики рабочего места. Хотя энтузиасты программирования непосредственно на КПК находятся. В общем же, программируются SmartDevice на настольных системах, а затем откомпилированные версии устанавливаются на КПК. Здесь возникает проблема соответствия версий настольного системного программного обеспечения и КПК, процесс тестирования приложений для КПК усложняется. Другой проблемой является урезанная функциональность прикладного программного обеспечения КПК по сравнению с настольными системами. Программированию SmartDevice в Microsoft уделяется отнюдь не главное внимание, поэтому в пакете MS Visual Studio 2003 (рассматриваемые в пособии примеры построены именно в данной среде) отсутствует целый ряд системных библиотек поддержки SmartDeviceApplication. Остается уповать только на сторонних разработчиков (например, OpenNetCF) и надеяться на дальнейшее развитие бизнеса Microsoft в данном направлении. Справедливости ради отметим, что в качестве компенсации отсутствия «привычных» средств программирования баз данных, Microsoft предлагает новую технологию доступа к данным - ADO.NET [2,3].

ADO.NET разделяет доступ к данным и манипуляции с ними. Соединенные классы, предоставляемые поставщиками данных ADO.NET, обеспечивают соединение с источником данных, выполнение команд и считывание результатов. Отсоединенные классы позволяют обращаться к данным и производить манипуляции с ними в автономном режиме, а затем синхронизировать изменения с соответствующим источником данных. В добавление к этому, XML тесно интегрированный в ADO.NET, дает возможность загружать данные, обращаться к ним и манипулировать ими, используя одновременно и XML, и отсоединенные классы.

С ростом популярности .NET и ADO.NET у разработчиков возникает множество вопросов о способах решения конкретных проблем и наиболее эффективной реализации решений. Пособие содержит решения для определенной категории проблем, примеры кода, демонстрирующего решение конкретной задачи. Все примеры кода функционально закончены, содержат комментарии, подробно разъяснены и апробированы в учебной практике. Для открытия и компиляции примеров понадобится Visual Studio.NET 2003 [4]. В книге опущен код, связанный с функционированием пользовательского интерфейса, а также код, автоматически генерируемый Visual Studio.NET.

Пособие имеет два приложения, в которых в виде справочника приведены конструкции языка C#.NET [5,7] и привязки элементов управления Windows [1,6] для приложений SmartDevice.

# 1. ADO.NET И ДОСТУП К ДАННЫМ

Технология ADO.NET предназначена для обеспечения доступа к данным в слабосвязанных n-уровневых архитектурах приложений, включая веб-службы [3]. ADO.NET позволяет обращаться к разнообразным источникам данных, как базы данных различных поставщиков (MS SQL Server, Oracle, MS Access), так и другие источники данных (MS Excel, Outlook, текстовые файлы). Для программирования баз данных **SmartDevice,** имеющих по определению ограниченную функциональность, наибольшее значение имеют текстовые форматы данных и данные отсоединенных наборов (DataSet). Набор данных не содержит информации об источнике данных, из которого он наполнялся. В результате можно сохранять набор данных как XML-документ и наполнять его данными как из XML-документа, так и текстового файла. Несмотря на это, каждый поставщик данных обязан предоставлять следующие отсоединенные классы:

- Connection – уникальный сеанс подключения к источнику данных;
- Command – команды выполнения SQL-операторов и процедур обработки данных;
- DataReader – последовательный доступ чтения потока результата запроса;
- DataAdapter – наполнитель отсоединенного набора (DataSet) и редактор источника данных в соответствии с изменениями, произведенными в отсоединенном наборе

Рассмотрим, как реализуется с помощью ADO.NET доступ к данным MS Access и MS Excel.

Задача темы: изучить методику и получить навыки в разработке функциональности приложений, использующих технологию ADO.NET для загрузки данных из файлов MS Access и MS Excel в форму C# для просмотра.

Разработка: Приложение «Connection» для автоматического подключения к базам данных в форматах mdb и xls.

Состав выполняемых функций:

- Файловые операции: импорт данных из файлов MS Access и MS Excel в форму C# через строку подсоединения для просмотра
- Алгоритмы: программное изменение свойств элемента Windows, предназначенного для просмотра данных
- Информационно-справочные: О программе

Наращивание функциональности не предусматривается.

Защита данных не предусматривается.

В качестве входных данных используются два тестовых файла - MS Access и MS Excel. Выходные данные программы – графическое отображение результатов в виде элемента табличного просмотра данных экранных форм для WindowsApplication

**Шаг 1. Организация дискового пространства и источников данных**

Необходимо создать каталог на диске, в который скопировать или создать заново базу данных MS Access и таблицу MS Excel. В нашем примере, каталог – ADO_XLS, в котором размещается приложение C# WindowsApplication. Файлы MS Access и Excel размещаются в каталоге \ADO_XLS\bin\Debug. В базе данных Access «Платежные поручения.mdb» присутствует таблица “ Контакты”  из двух полей: Код и Контакты, число записей в таблице – три, хотя это непринципиально. В файле Excel «Копия ADO_NET.xls» содержится лист «Содержание» с заполненным столбцом A и числом строк – 22, здесь диапазон строк имеет значение.

**Шаг 2. Тестируем подключение к источникам данных**

Подключение к источникам данных организуется на настольном компьютере с помощью различных драйверов. Для Access должен присутствовать драйвер Microsoft.Jet.OLEDB.4.0, а для Excel – Microsoft OLE DB Provider for ODBC Drivers. В MS Visual Studio 2003 рекомендуется предварительно проверить подключение к источникам данных с помощью этих драйверов. Для этого  в главной линейке меню выбираются опции: Сервис -> Подключение к базам данных -> Поставщики -> Microsoft.Jet.OLEDB.4.0

Provider -> Выбирается файл «Платежные поручения.mbd» -> Проверить подключение – для подключения к файлу Access. Аналогично – для Excel, только провайдер указывается уже другой.

Если тестирование успешно, то в обозревателе серверов (Вид -> Обозреватель серверов) будут видны файлы, к которым выполнено подключение - рис.1

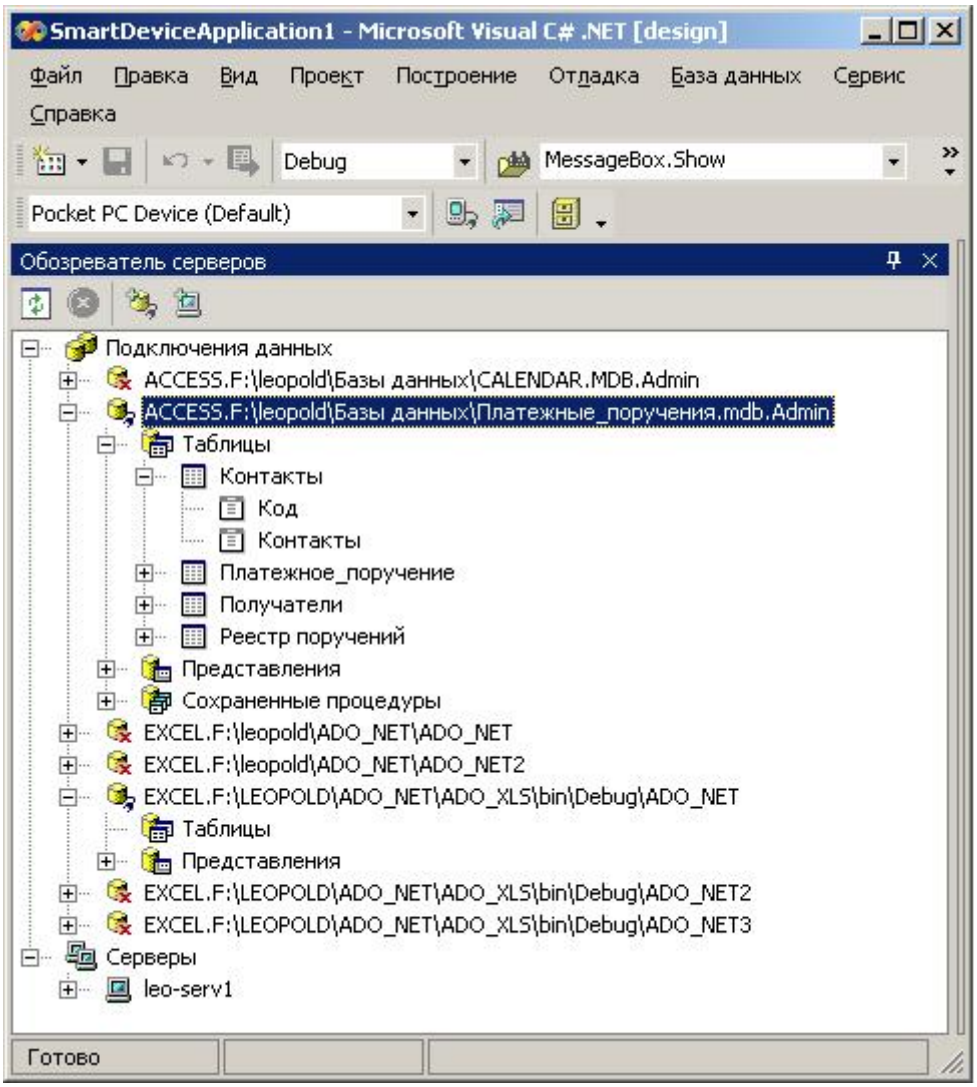


Рис.1 Подключение к источникам данных

Далее следует просмотреть свойства подключенных файлов и скопировать содержание строки подключения (ConnectionString) – рис.2. Дело в том, что разные провайдеры имеют разный синтаксис строки подключения, поэтому чтобы не гадать, точнее – чтобы не нарываться на исключения при тестировании приложения, лучше довериться генератору строки в MS Visual Studio.

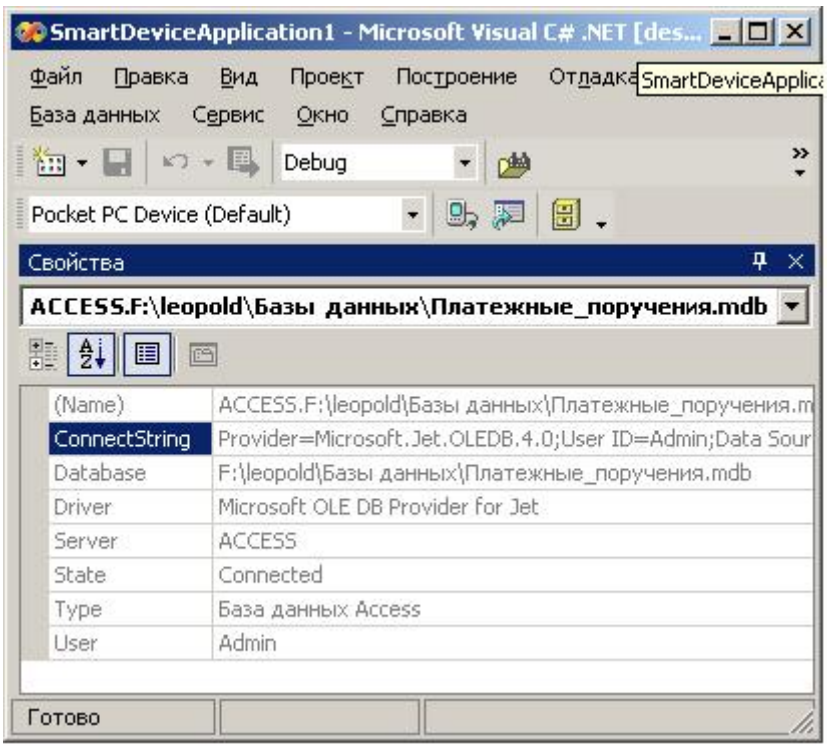


Рис.2 ConnectionString в окне просмотра свойств подключения к файлу

**Шаг 3. Разработка графического интерфейса приложения**

Для программы используются одна форма (Form1), на которой размещено окно DataGrid для просмотра данных таблиц, две кнопки подключения к источникам данных и Главное меню (mainMenu) с единственной опцией «О программе». Примерный дизайн которых показан на рис.3.

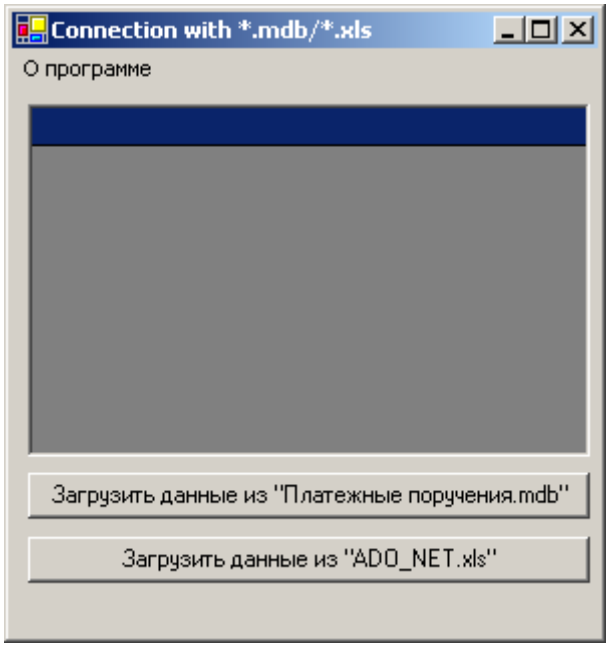


Рис.3

**Шаг 4. Расширяем состав библиотек формы 1**

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
/* добавляем библиотеки подключения к источникам данных
Отметим, что данные библиотеки не доступны для SmartDeviceApplication, поэтому мы и создаем WindowsApplication для настольной компьютерной системы */
using System.Data;
using System.Data.Odbc;
using System.Data.OleDb;
```

**Шаг 5. Создаем процедуру для кнопки подключения к данным файла Access**

```
private void button1_Click(object sender, System.EventArgs e)
{// Создаем соединение
System.Data.OleDb.OleDbConnection Коннект=new OleDbConnection();
// Задаем параметры строки соединения
Коннект.ConnectionString= "Data Source=Платежные_поручения.mdb;"+
"Provider=Microsoft.Jet.OLEDB.4.0";
/* А вот это копия строки соединения, сгенерированная редактором MS Visual Studio:
    Jet OLEDB:Global Partial Bulk Ops=2; Jet OLEDB:Registry Path=; Jet OLEDB:Database Locking Mode=1; Data Source= Платежные_поручения.mdb; Jet OLEDB:Engine Type=5; Provider=Microsoft.Jet.OLEDB.4.0; Jet OLEDB:System database=; Jet OLEDB:SFP=False; persist security info=False; Extended Properties=; Mode=Share Deny None; Jet OLEDB:Encrypt Database=False; Jet OLEDB:Create System Database=False; Jet OLEDB:Don't
```

```
Copy Locale on Compact=False; Jet OLEDB:Compact Without Replica Repair=False;  User ID=Admin; Jet OLEDB:Global Bulk Transactions=1
```

Можно видеть, что принципиально важными являются два параметра – источник данных и провайдер, остальные принимаются по умолчанию.

```
*/
/* Формируем SQL – запрос к данным таблицы «Контакты». В нашем случае выбираются все поля и записи таблицы
*/
OleDbDataAdapter da =
        new OleDbDataAdapter ("select * from контакты", Коннект);
// Создаем таблицу для данных
DataTable dt =new DataTable();
/* таблице присваиваем имя, чтобы программно определить свойства DataGrid
*/
dt.TableName="Контакты";
da.Fill(dt); // Заполняем таблицу данными из адаптера
//привязываем элемент просмотра данных программно
dataGrid1.TableStyles.Clear(); // чистим стиль элемента
// стиль грид делаем программно
DataGridTableStyle ts= new DataGridTableStyle();
// стиль будет разработан для таблицы «Контакты»
ts.MappingName="Контакты";
DataGridTextBoxColumn cs=new DataGridTextBoxColumn();
cs.MappingName="Код"; // для столбца «Код»
cs.HeaderText="Код"; // Заголовок столбца – тоже «Код»
cs.Width=30; // ширина столбца 30 пикселей
cs.NullText="";
ts.GridColumnStyles.Add(cs); // добавили в стиль столбцов
```

```
// форматируем второй столбец
cs=new DataGridTextBoxColumn();
cs.MappingName="Контакты"; // имя столбца «Контакты»
cs.HeaderText="Контакты";
cs.Width=240;
cs.NullText="";
ts.GridColumnStyles.Add(cs);
//  добавляем в стиль DataGrid
this.dataGrid1.TableStyles.Add(ts);
dataGrid1.DataSource=dt.DefaultView; // Смотрим, что получилось
}
```

**Шаг 6. Создаем процедуру для кнопки подключения к данным файла Excel**

```
private void button2_Click(object sender, System.EventArgs e)
{// создаем соединение с помощью драйвера ODBC
System.Data.Odbc.OdbcConnection Коннект=new OdbcConnection();
Коннект.ConnectionString= "Provider=MSDASQL.1;"+
        "DSN=Excel Files;"+
        "DBQ=Копия ADO_NET.xls";
/* А это копия строки подключения. Почувствуйте разницу:
Provider=MSDASQL.1;
  Persist Security Info=False;
  Extended Properties="DSN=Excel Files";
  DBQ=F:\LEOPOLD\ADO_NET\ADO_NET.xls;
  DefaultDir=F:\LEOPOLD\ADO_NET;
  DriverId=790;
  MaxBufferSize=2048;
  PageTimeout=5;
  Initial Catalog=F:\LEOPOLD\ADO_NET\ADO_NET
  */
```

```
/* обратите внимание – в SQL – запросе к листу книги «Содержание» указывается диапазон ячеек через знак $
*/
OdbcDataAdapter da =
        new OdbcDataAdapter ("select * from [Содержание$A1:A33]",Коннект);
// дальше все аналогично, как в процедуре доступа к файлу Access
DataTable  dt =new DataTable();
dt.TableName="Содержание";
da.Fill(dt);
//привязываем
dataGrid1.TableStyles.Clear();
// стиль грид делаем программно
DataGridTableStyle ts= new DataGridTableStyle();
ts.MappingName="Содержание";
DataGridTextBoxColumn cs=new DataGridTextBoxColumn();
cs.Width=270;
/* Здесь тонкость – в запросе не поддерживается точка, вместо нее генерируется #
*/
cs.MappingName="ADO#NET для SmartDevice:";
// а заголовок столбца поменяем на правильный
cs.HeaderText="ADO.NET для SmartDevice:";
cs.NullText="";
ts.GridColumnStyles.Add(cs);
this.dataGrid1.TableStyles.Add(ts);
dataGrid1.DataSource=dt.DefaultView;
}
```

Результат работы кнопок показан на рис. 4 и 5

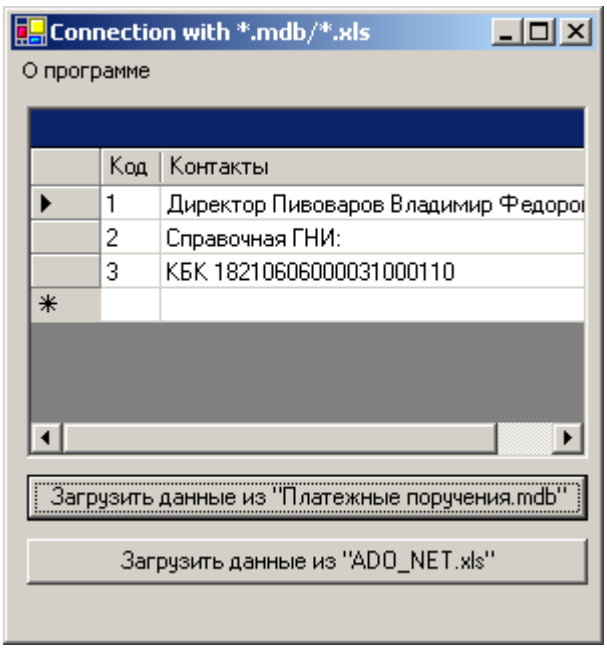


Рис.4 Загруженные данные из MS Access

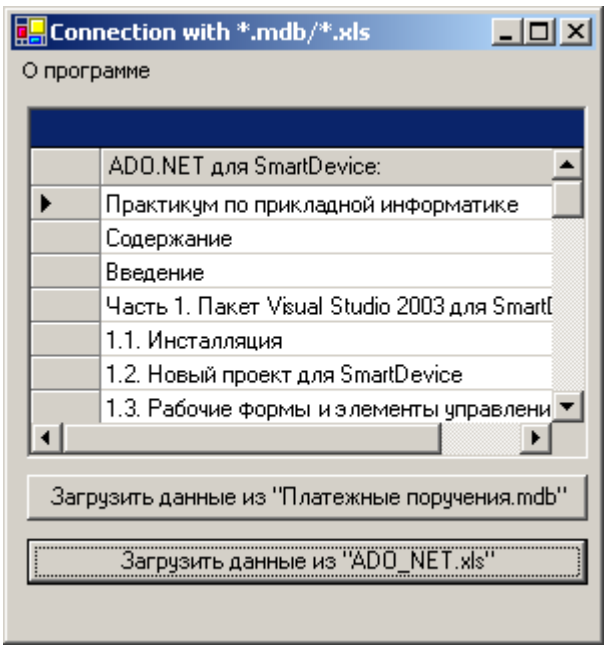


Рис.5 Загруженные данные из MS Excel

**Шаг 7. Создаем процедуру для опции главного меню «О программе»**

В процедуре выводится сообщение с информацией о литературных источниках, в которых представлена более подробная информация о подключении к источникам данных различных провайдеров

```
private void menuItem1_Click(object sender, System.EventArgs e) {
MessageBox.Show("Создание соединения с источником данных\n"+
        "и установка строки связи.\n"+
        "Байдачный С.С., NET Framework. М., \n"+
        "Солон-Пресс, 2004, с. 386\n"+
        "Гамильтон Б. ADO.NET для профессионалов \n"+
        "СПб., Питер, 2005, с.25","Подробнее...");
}
```

# 2. ТЕКСТОВОЙ ФОРМАТ ДАННЫХ ДЛЯ SMARTDEVICE

Для доступа к данным в текстовом файле используется поставщик OLE DB Jet, который загружает содержимое текстового файла в локальную таблицу и отображает ее содержимое в специально предназначенном элементе экранной формы. Локальные таблицы могут быть организованы по-разному, поэтому *невозможно определить все характеристики текстового файла через строку подключения.* Для решения этой проблемы используется дополнительный, расположенный в том же каталоге, что и текстовой файл, файл *.ini в котором отражается следующая информация о локальной таблице текстового файла:

- Имя файла:
- Формат файла;
- Имена полей, их длина и тип данных;
- Кодировка символов;
- Специальные преобразования типов данных

Отметим, что в строке (записи) файла *.ini «Формат файла» указываются разделители полей: запятая; нестандартный разделитель определенный пользователем; фиксированная длина поля или табулятор.

Что касается элементов экранных форм, предназначенных для отражения данных, то их существует несколько вариантов, привязку данных к которым рассмотрим ниже.

## 2.1. Ввод-вывод в текстовые файлы

Задача темы: изучить методику и получить навыки в разработке функциональности приложений, использующих загрузку данных из текстового файла в форму для редактирования, изменение и сохранение данных в текстовом файле.

Разработка: Приложение «Калькулятор цены автомобиля» для автоматического подсчета цены автомобиля в зависимости от комплектации.

Состав выполняемых функций:

- Файловые операции: открыть, сохранить, сохранить как, закрыть
- Алгоритмы: калькулятор цены автомобиля, редактор цен
- Информационно-справочные: О программе

Наращивание функциональности не предусматривается.

Защита данных предусматривается только от неквалифицированных действий пользователей.

В качестве входных данных используется отдельный текстовой файл по каждой марке автомобиля, структура данных включает одно поле, набор из 22 записей, тип данных определяется программно. Выходные данные программы – графическое отображение результатов в виде набора текстовых полей экранных форм для SmartDevice.

**Шаг 1. Структура данных**

В программе используется фиксированный набор данных текстовых полей (textBox), которые имеют следующие обозначения (label):

- Владелец автомобиля
- Телефон владельца
- Цена владельца
- Комплектация
- Базовая цена
- Усилитель руля
- Кондиционер
- Тонированные стекла
- Дисковые тормоза
- Сигнализация
- Фаркоп
- Противотуманные фары
- Диски легкосплавные
- Марка автомобиля

**Шаг 2. Разработка графического интерфейса**

Используется только одна форма Form1 (свойство Text = Урок 1. Калькулятор). В форме, в связи с ограниченностью экрана, размещается набор закладок tabControl1 с двумя закладками tabPage1 и tabPage2. Примерный дизайн закладок приведен на рис.6 и 7.

На закладке «Комплектация» (рис.1) размещаются 8 флажков (checkBox) маркировки состава комплектующих (изначально свойство флажка Checked = False) и кнопка (button) «OK», видимая только в режиме «Редактора» (изначально свойства кнопки Visible = False, Text = OK).

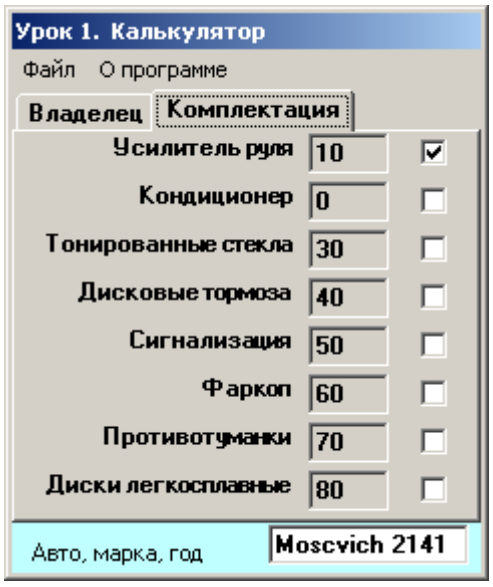


Рис.6

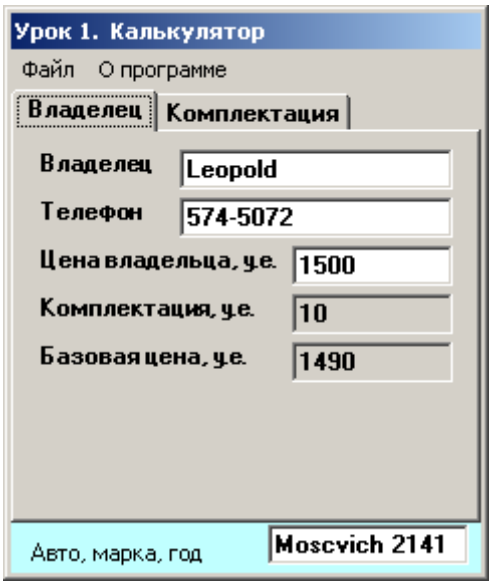


Рис.7

Невидимыми элементами управления графического интерфейса являются Диалоги открытия (openFileDialog1) и сохранения (saveFileDialog1) файла, Главное меню (mainMenu1), горизонтальная линейка которого имеет два пункта – «Файл» и «О программе» (видно на рис.1 и 2). Набор опций меню «Файл» включает: Открыть, Сохранить, Сохранить как, Редактор цен и Выход. Пункт меню «О программе» опций не имеет.

**Шаг 3. Расширяем состав системных библиотек**

```
using System;
using System.Drawing;
using System.Collections;
using System.Windows.Forms;
```

```
using System.Data;
// Добавляем библиотеку работы с файлами
using System.IO;
```

**Шаг 4. Включаем переменные для хранения результатов вычислений в public class Form1 : System.Windows.Forms.Form**

```
string file_name; // переменная хранения имени файла с данными
int appe_acc=0; // переменная редактируемой цены комплектации
int base_acc=0; // переменная базовой цены автомобиля
int comp_acc=0; // переменная цены комплектации автомобиля
```

**Шаг 5. Загружаем данные из файла (по умолчанию)**

```
public Form1()
{
// Required for Windows Form Designer support
InitializeComponent();
file_name="price.txt"; // имя файла, загружаемого по умолчанию
file_read(); // вызываем функцию чтения данных из файла
}
```

**Шаг 6. Создаем функцию чтения данных из текстового файла**

```
private void file_read()
{
FileStream fin;
try {
/* если файл доступен для чтения, то определяем переменную, в которой задаем имя открываемого файла */
fin=new FileStream(file_name,FileMode.Open);
        }
catch (IOException exc) {
MessageBox.Show ("Невозможно открыть файл \n"+ file_name,"Ошибка");
return;
        }
```

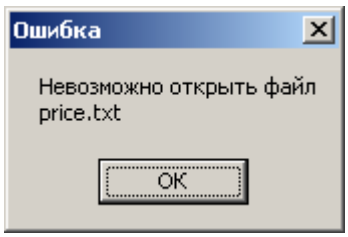


Рис.8

На этом шаге можно закрыть функцию фигурной скобкой «}» и откомпилировать программу. При запуске откомпилированной программы, так как файл price.txt не существует, то должно срабатывать исключение IOException и выдаваться сообщение – рис.3

**Шаг 7. Создаем процедуру сохранения файла**

Опция «Сохранить» (в нашем примере - menuItem4_Click) главного меню предназначена для сохранения данных в файле price.txt – создаем код процедуры:

```
private void menuItem4_Click(object sender, System.EventArgs e)
{
file_name="price.txt"; // задаем имя файла по умолчанию
file_write(); // вызываем функцию записи данных в файл
}
```

**Шаг 8. Создаем функцию записи структуры данных в файл**

```
private void file_write()
{// задаем поток
StreamWriter fin_out;
try {// проверяем доступность файла
fin_out=new StreamWriter(file_name);
        }
catch(IOException exc) {
MessageBox.Show ("Невозможно открыть файл.","Ошибка");
return;
        }
// считываем данные текстовых полей формы
```

```
foreach (Control t in this.Controls) {
if (t is TextBox) // условие: если элемент формы – текстовое поле
fin_out.Write(t.Text+"\r\n"); // пишем значения полей в файл
        }
fin_out.Close(); // закрываем файл
```

На этом шаге можно закрыть функцию фигурной скобкой «}» и откомпилировать программу. При запуске откомпилированной программы, активизировав опцию меню «Сохранить», на диске должен быть создан файл price price.txt, содержащий значения текстовых полей формы программы.

**Шаг 9. Добавляем в функцию file_write() данные состояния флажков checkbox**

Перед закрывающей функцию file_write() фигурной скобкой пишем:

```
//Теперь открываем файл для добавления
try{
fin_out=new StreamWriter(file_name,true);
        }
catch(IOException exc)  {
MessageBox.Show ("Невозможно открыть файл.","Ошибка");
return;
        }
// пишем состояние всех восьми флажков
string h1=this.checkBox1.CheckState.ToString();
fin_out.Write(h1.ToString()+"\r\n"); // дописываем в файл
string h2=this.checkBox2.CheckState.ToString();
fin_out.Write(h2.ToString()+"\r\n");
string h3=this.checkBox3.CheckState.ToString();
fin_out.Write(h3.ToString()+"\r\n");
string h4=this.checkBox4.CheckState.ToString();
fin_out.Write(h4.ToString()+"\r\n");
string h5=this.checkBox5.CheckState.ToString();
```

```
fin_out.Write(h5.ToString()+"\r\n");
string h6=this.checkBox6.CheckState.ToString();
fin_out.Write(h6.ToString()+"\r\n");
string h7=this.checkBox7.CheckState.ToString();
fin_out.Write(h7.ToString()+"\r\n");
string h8=this.checkBox8.CheckState.ToString();
fin_out.Write(h8.ToString()+"\r\n");
// считываем данные текстовых полей 1-й закладки
foreach (Control t1 in this.tabPage1.Controls) {
if (t1 is TextBox)
fin_out.Write(t1.Text.ToString()+"\r\n");
        }
// считываем данные текстовых полей 2-й закладки
foreach (Control t2 in this.tabPage2.Controls) {
if (t2 is TextBox)
fin_out.Write(t2.Text+"\r\n");
        }
fin_out.Close();// закрываем файл и выводим сообщение
MessageBox.Show("Данные сохранены в файле "+file_name,"Запись на диск");
} // закрывающая фигурная скобка
```

**Шаг 10. Дописываем функцию file_read()**

Теперь, когда данные в файле price.txt структурированы, перед закрывающей функцию file_read() фигурной скобкой пишем:

```
// Считываем значения из файла
StreamReader fstr_in=new StreamReader(fin);
string s="0"; // символ в строке
int i=0; // номер строки
//в переменную s считываются все символы, пока не кончится строка
while((s=fstr_in.ReadLine())!=null)   {
i=i+1;
```

```
if (i = = 1){this.textBox1.Text=s;} // Марка автомобиля в textBox1
if (i = = 2) {// первый флажок в checkBox1
        if (s = = "Unchecked") { this.checkBox1.Checked=false;}
        else {this.checkBox1.Checked=true;}
        }
if (i = = 3) {// второй флажок
        if (s = = "Unchecked") { this.checkBox2.Checked=false;}
        else {this.checkBox2.Checked=true;}
        }
if (i = = 4) {// третий флажок
        if (s = = "Unchecked") { this.checkBox3.Checked=false;}
        else { this.checkBox3.Checked=true;}
        }
if (i = = 5) {// четвертый флажок
        if (s = = "Unchecked") { this.checkBox4.Checked=false;}
        else { this.checkBox4.Checked=true;}
        }
if ( i = = 6) {// пятый флажок
        if (s = = "Unchecked") { this.checkBox5.Checked=false;}
        else { this.checkBox5.Checked=true;}
        }
if (i = = 7) {// шестой флажок
        if (s = = "Unchecked") { this.checkBox6.Checked=false;}
        else { this.checkBox6.Checked=true;}
        }
if (i = = 8) {// седьмой флажок
        if (s = = "Unchecked") {this.checkBox7.Checked=false;}
        else { this.checkBox7.Checked=true;}
        }
if ( i = = 9) {// восьмой флажок
```

```
        if (s = = "Unchecked") { this.checkBox8.Checked=false;}
        else { this.checkBox8.Checked=true;}
        }
/* Заносим данные в текстовые поля. Отметим, что порядок расположения и обозначения полей зависит от последовательности конструирования графического интерфейса. Поэтому, в нашем примере, обозначения могут не совпадать с автоматически создаваемыми элементами форм редактором Visual Studio */
if (i = = 10) {this.textBox14.Text=s;}
if (i = = 11)  {this.textBox13.Text=s;
/* преобразуем текстовой формат поля в числовой и заносим значение в переменную цены комплектации автомобиля */
comp_acc=int.Parse(this.textBox13.Text);
        }
if (i = = 12)  {this.textBox4.Text=s;}
if (i = = 13)  {this.textBox3.Text=s;}
if (i = = 14)  {this.textBox2.Text=s;}
if (i = = 15)  {this.textBox12.Text=s;}
if (i = = 16)  {this.textBox11.Text=s;}
if (i = = 17)  {this.textBox10.Text=s;}
if (i = = 18)  {this.textBox9.Text=s;}
if (i = = 19)  {this.textBox8.Text=s;}
if (i = = 20)  {this.textBox7.Text=s;}
if (i = = 21)  {this.textBox6.Text=s;}
if (i = = 22)  {this.textBox5.Text=s;}
        }// конец цикла
fstr_in.Close();
return;
} // закрывающая функцию фигурная скобка
```

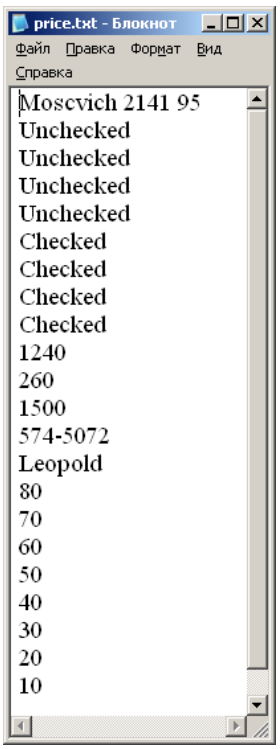


Рис.4

Теперь следует удалить или переименовать файл price.txt, откомпилировать и запустить программу. В textBox13 нужно внести цифру, чтобы избежать генерации исключения операционной системой, заполнить остальные поля и сохранить файл. Структуру (одно поле) и данные (22 записи) файла можно просмотреть в режиме блокнота – рис.4

**Шаг 11. Функция проверки цифровых полей**

Для того, чтобы наш калькулятор вычислял цену автомобиля необходимо чтобы пользователь вводил цифры в соответствующие поля. В противном случае программа будет прерываться, генерируя соответствующее исключение. Для проверки правильности ввода и корректировки ошибок создадим функцию Accountest()

```
private void Accountest()
{// проверяем цену продавца
if (!char.IsDigit(textBox4.Text[0])) {
textBox4.Text = "0"; // корректируем ошибки ввода – вводим в поля нули
MessageBox.Show("Введите цену владельца!","Ошибка");
return;
        }
base_acc=int.Parse(textBox4.Text);
// проверяем цифры для каждого each
foreach (Control NT in this.tabPage2.Controls)
if (NT is TextBox){
```

```
        if (!char.IsDigit(NT.Text.Trim()[0]))  {
        NT.Text = "0"; // корректируем ошибки ввода – вводим в поля нули
                }
        }
}
```

**Шаг 12. Процедуры пересчета цены**

Процедура пересчета цены срабатывает при каждом изменении состояния флажка комплектации – для всех восьми флажков процедуры одинаковые, за исключением привязки к элементам управления. Для первого флажка процедура:

```
private void checkBox1_CheckStateChanged_1(object sender, System.EventArgs e)
{
Accountest(); // вызваем функцию проверки правильности ввода
// считываем в переменную предыдущее значение цены комплектации
appe_acc=int.Parse(textBox5.Text);
if (this.checkBox1.Checked = = false){comp_acc=comp_acc-appe_acc ;}
else {comp_acc=comp_acc+appe_acc ;}
this.textBox13.Text=comp_acc.ToString();
this.textBox14.Text=(base_acc-comp_acc).ToString();
}
```

Для второго флажка меняется только строчка:

```
if (this.checkBox2.Checked = = false){comp_acc=comp_acc-appe_acc ;}
```

и так далее, до восьмого флажка включительно.

**Шаг 13. Защита данных базовой комплектации**

Файл price.txt в программе создается по умолчанию и содержит данные базовой комплектации для всех других вариантов файлов. Чтобы каждый раз не создавать данные в случае неквалифицированных или случайных действий пользователей, следует ограничить доступ к данным файла. Для этого всем полям на закладке «Комплектация» (Усилитель руля,  Кондиционер,

Тонированные стекла, Дисковые тормоза, Сигнализация, Фаркоп, Противотуманные фары, Диски легкосплавные) изначально устанавливается свойство ReadOnly=True. На закладке «Владелец» такое же свойство устанавливается для полей «Комплектация» и «Базовая цена», так как поля являются вычисляемыми и пользователю нет необходимости в доступе к этим полям.

Доступ для редактирования полей закладки «Комплектация» открывается опцией главного меню «Редактор цен», для которой создается процедура:

```
private void menuItem2_Click_1(object sender, System.EventArgs e)
{// блокируем флажки
this.checkBox1.Checked=false;
this.checkBox2.Checked=false;
this.checkBox3.Checked=false;
this.checkBox4.Checked=false;
this.checkBox5.Checked=false;
this.checkBox6.Checked=false;
this.checkBox7.Checked=false;
this.checkBox8.Checked=false;
// прячем флажки
this.checkBox1.Visible=false;
this.checkBox2.Visible=false;
this.checkBox3.Visible=false;
this.checkBox4.Visible=false;
this.checkBox5.Visible=false;
this.checkBox6.Visible=false;
this.checkBox7.Visible=false;
this.checkBox8.Visible=false;
// обнуляем комплектацию
comp_acc=0;
this.textBox13.Text=comp_acc.ToString();
```

```
this.textBox14.Text=(base_acc-comp_acc).ToString();
// показываем кнопку редактора
this.button1.Visible=true;
//отклываем текстовые поля для редактирования
this.textBox12.ReadOnly=false;
this.textBox11.ReadOnly=false;
this.textBox10.ReadOnly=false;
this.textBox9.ReadOnly=false;
this.textBox8.ReadOnly=false;
this.textBox7.ReadOnly=false;
this.textBox6.ReadOnly=false;
this.textBox5.ReadOnly=false;
}
```

После того, как данные на закладке «Комплектация» отдедактированы, закрываем к ним доступ кнопкой «ОК»:

```
private void button1_Click_1(object sender, System.EventArgs e)
{// делаем кнопку «ОК» невидимой
this.button1.Visible=false;
// поля - недоступными
this.textBox12.ReadOnly=true;
this.textBox11.ReadOnly=true;
this.textBox10.ReadOnly=true;
this.textBox9.ReadOnly=true;
this.textBox8.ReadOnly=true;
this.textBox7.ReadOnly=true;
this.textBox6.ReadOnly=true;
this.textBox5.ReadOnly=true;
// флажки - показываем
this.checkBox1.Visible=true;
this.checkBox2.Visible=true;
```

```
this.checkBox3.Visible=true;
this.checkBox4.Visible=true;
this.checkBox5.Visible=true;
this.checkBox6.Visible=true;
this.checkBox7.Visible=true;
this.checkBox8.Visible=true;
}
```

**Шаг 14. Диалог сохранения файла**

Вариантов цен и комплектаций может быть много, поэтому каждый вариант храним в отдельном файле. Для этого пишем процедуру опции «Сохранить как» (menuItem5_Click).

```
private void menuItem5_Click(object sender, System.EventArgs e)
{// в качестве имени файла используем значение поля
file_name=this.textBox1.Text;
saveFileDialog1.FileName=file_name;
if (saveFileDialog1.ShowDialog()==DialogResult.OK) {
file_name=saveFileDialog1.FileName;
FileStream fin_out;
try     {
        fin_out=new FileStream(file_name,FileMode.OpenOrCreate);
        fin_out.Close();
        }
catch(IOException exc)  {
        MessageBox.Show ("Невозможно открыть файл.","Ошибка");
        return;
        }
file_write(); // загружаем функцию записи данных в файл
        }
}
```

**Шаг 15. Диалог открытия файла**

По умолчанию всегда загружаются данные из файла price.txt – чтобы загрузить данные конкретной комплектации  из другого файла создаем процедуру опции главного меню «Открыть» (menuItem3_Click)

```
private void menuItem3_Click(object sender, System.EventArgs e){
if (openFileDialog1.ShowDialog()==DialogResult.OK) {
    file_name=openFileDialog1.FileName;
    file_read(); // загружаем функцию чтения данных из файла
    }
}
```

**Шаг 16. Закрываем приложение**

Одна из особенностей операционной системы для SmartDevice в том, что привычная кнопка закрытия приложения не удаляет программу из оперативной памяти. По сути приложение только сворачивается и делается не видимым. Чтобы освободить память пишем процедуру для опции главного меню «Выход» (menuItem7_Click)

```
private void menuItem7_Click(object sender, System.EventArgs e) {
Application.Exit();
}
```

Теперь можно установить свойство для формы Form1 ControlBox=False – выход из программы будет осуществляться использованием опции «Выход».

**Шаг 17. Авторские права**

Хорошим тоном является авторизация разработки – общественность должна знать своих героев. Пишем процедуру для опции «О программе»:

```
private void menuItem6_Click(object sender, System.EventArgs e){
MessageBox.Show("Калькулятор цены автомобиля \n (C)2006, \n Родигин Л.А.\n Задача: сохранить значения цен комплектующих в файле \n и подсчитать сумму выбранных значений из калькуляции.","Help");
}
```

## 2.2. Поиск в текстовых файлах

Задача темы: изучить методику и получить навыки в разработке функциональности приложений, использующих загрузку данных из текстового файла в форму для просмотра, поиска и выборки записей по ключу, добавление и сохранение данных в текстовом файле.

Разработка: Приложение «Поиск записей в текстовом файле» для автоматического подсчета средней зарплаты в выборке.

Состав выполняемых функций:

- Файловые операции: открыть, сохранить, закрыть
- Операции с записями: добавить, найти, выбрать
- Алгоритмы: подсчет средней зарплаты в выборке
- Информационно-справочные: О программе

Наращивание функциональности не предусматривается.

Защита данных предусматривается только от неквалифицированных действий пользователей. Редактирование – вручную с помощью редактора «Блокнот».

В качестве входных данных используется отдельный текстовой файл Сотрудники.txt в котором хранится база данных отдела кадров предприятия. Каждая строка файла содержит запись об одном сотруднике. Формат записи: Фамилия и инициалы – 18 символов, начиная с первой позиции; год рождения – 6 позиций; оклад – неограниченно. Тип данных определяется программно. Выходные данные программы – графическое отображение результатов в виде листбоксов и набора текстовых полей экранных форм для SmartDevice.

**Шаг 1. Разработка графического интерфейса**

Для программы используются три формы, примерный дизайн которых показан на рис.9, 10 и 11.

Главная форма (Form1) на рис.9 содержит элементы:

- 3 надписи (label) со свойствами Text: В текстовом файле Сотрудники.txt хранится база данных отдела кадров предприятия. Каждая стро-

ка файла содержит запись об одном сотруднике. Формат записи: фамилия и инициалы (18 позиций, фамилия должна начинаться с первой позиции), год рождения (6 позиций), оклад (неограничено); Программа по заданной фамилии выводит на экран сведения о сотруднике, подсчитывая средний оклад всех запрошенных сотрудников; (С)Родигин Л.А., 2006. Платформа MS Visual C#.Net 2003;

- главное меню (mainMenu) с двумя опциями: «База данных» и «Новая запись».

Форма «База данных» (Form2) на рис.10 содержит элементы:

- 5 надписей (label) со свойствами Text: Фамилия и инициалы; г.р. (год рождения); Оклад; Click для удаления (записи из выборки); Ср.оклад в выборке, у.е.;
- 2 листбокса (listBox) – для базы данных (верхний) и для выборки данных (нижний) из базы;
- 1 текстовое поле для ввода фамилии в качестве ключа поиска в базе данных;
- 1 кнопку «Найти и добавить» активизации поиска и выборки данных

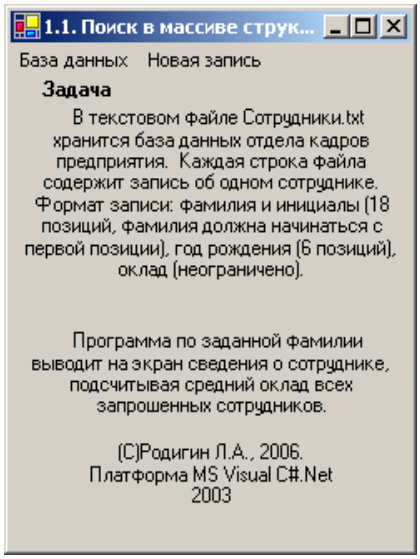


Рис.9 Главная форма

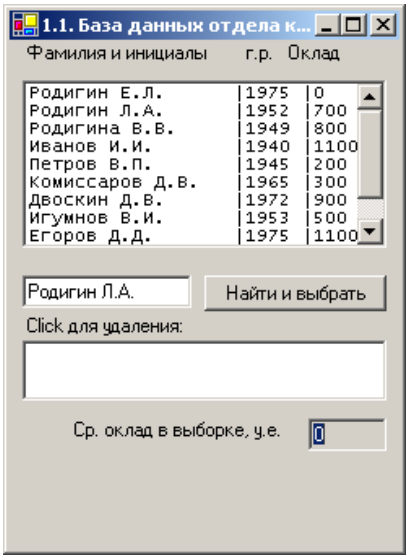


Рис.10 База данных

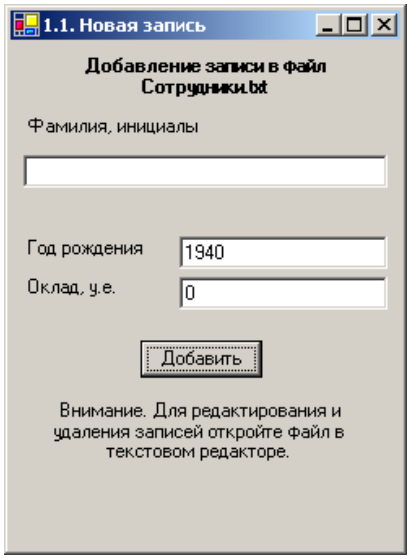


Рис.11 Новая запись

Форма «Новая запись» (Form3) на рис.11 содержит элементы:

- 5 надписей со свойствами Text: Добавление записи в файл Сотрудники.txt; Фамилия, инициалы; Год рождения; Оклад, у.е.; Внимание. Для редактирования и удаления записей откройте файл в текстовом редакторе;
- 3 текстовых поля (textBox);
- Кнопку «Добавить» для добавления записи в файл Сотрудники.txt

**Шаг 2. Открываем форму «База данных»**

Для опции меню «База данных» (menuItem1_Click) пишем:

```
private void menuItem1_Click(object sender, System.EventArgs e)
{// Открываем форму базы данных
Form DB=new Form2();
DB.Show(); // выводим форму на экран
}
```

**Шаг 3. Открываем форму «Новая запись»**

Для опции меню «Новая запись» (menuItem2_Click) пишем:

```
private void menuItem2_Click(object sender, System.EventArgs e)
{// Открываем форму ввода новой записи
Form RW=new Form3();
RW.Show();
}
```

**Шаг 4. Расширяем состав библиотек формы 2**

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
// добавляем библиотеку работы с файлами
using System.IO;
```

**Шаг 5. Включаем переменные для хранения результатов вычислений в public class Form2 : System.Windows.Forms.Form**

```
string s_new; // переменная для хранения данных нового сотрудника
int i_row=0; // число строк в выборке
int i_sum=0; // сумма выборки
```

**Шаг 6. Загружаем данные из файла (по умолчанию)**

```
public Form2() {
InitializeComponent();
// Тестируем наличие файла с данными и создаем файл для записи значений
FileStream fin;
Try {
fin=new FileStream("Сотрудники.txt",FileMode.Open);
        }
catch(IOException exc)  {
MessageBox.Show ("Невозможно открыть файл.","Ошибка");
return;
        }
// Считываем значения из файла
string s;
StreamReader fstr_in=new StreamReader(fin);
while((s=fstr_in.ReadLine())!=null)   {
listBox1.Items.Add(s); // заполняем верхний листбокс
          }
fstr_in.Close(); // закрываем файловый поток
}
```

**Шаг 7. Создаем функцию вычисления средней в выборке**

```
void AppListBox2() {
// вычисляем длину выбранной строки из листбокса
int i_len=s_new.Length;
string s_pay=s_new.Substring(25,i_len-25);
i_row=i_row+1; // считаем количество строк
// преобразуем текст в сумму и считаем сумму выборки
```

```
i_sum=int.Parse(s_pay)+i_sum;
float f_sum=i_sum/i_row; // считаем среднюю
textBox2.Text=f_sum.ToString(); // выводим среднюю в текстовое поле
}
```

**Шаг 8. Создаем процедуру выборки значения из верхнего листбокса в нижний по щелчку**

```
private void listBox1_SelectedIndexChanged(object sender, System.EventArgs e)
{
s_new=listBox1.SelectedItem.ToString();
listBox2.Items.Add(s_new); // добавляем запись в листбокс2
AppListBox2(); // вызваем функцию вычисления средней в выборке
}
```

**Шаг 9. Создаем процедуру удаления записей из выборки и пересчета средней по оставшимся в выборке записям**

```
private void listBox2_SelectedIndexChanged(object sender, System.EventArgs e)
{// если в выборке есть еще записи, которые можно удалить
if (listBox2.SelectedItem != null) {
// вычисляем длину выбранной строки из листбокса
int i_len=listBox2.SelectedItem.ToString().Length;
string s_pay=listBox2.SelectedItem.ToString().Substring(25,i_len-25);
// считаем количество строк
i_row=i_row-1;
// преобразуем текст в сумму и считаем сумму выборки
i_sum=-int.Parse(s_pay)+i_sum;
// считаем среднюю
float f_sum;
if (i_row = = 0) {f_sum=0;}
else    {f_sum=i_sum/i_row;}
textBox2.Text=f_sum.ToString();
// удаляем строки из listbox2
```

```
listBox2.Items.Remove(listBox2.SelectedItem.ToString());
        }
}
```

**Шаг 10. Создаем процедуру поиска при активизации кнопки**

```
private void button1_Click(object sender, System.EventArgs e) {
// вычисляем длину выбранной строки из TEXTбокса
int i_len=textBox1.Text.Length;
int i_row=listBox1.Items.Count+1;
//цикл просмотра списка верхнего листбокса
for(int i=0; i<listBox1.Items.Count;i++) {
// сравниваем значения текстбокса и листбокса
string s_name=listBox1.Items[i].ToString().Substring(0,i_len);
if (textBox1.Text==s_name)     {i_row=i;}
        }
if (i_row<=listBox1.Items.Count) {
s_new=listBox1.Items[i_row].ToString();
listBox2.Items.Add(s_new);
AppListBox2();
        }
else    {MessageBox.Show(textBox1.Text,"Не найдено");}
}
```

**Шаг 11. Расширяем состав библиотек формы 3**

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
// добавляем библиотеку работы с файлами
using System.IO;
```

**Шаг 12. Включаем константы размеров полей базы данных в public class Form3 : System.Windows.Forms.Form**

```
const int l_name=18;
const int l_year=6;
```

**Шаг 13. Создаем процедуру добавления записи в файл Сотрудники.txt для кнопки button1**

```
private void button1_Click(object sender, System.EventArgs e) {
// Проверка значений полей
if (textBox1.Text = = "") {
MessageBox.Show("Введите фамилию и инициалы сотрудника!","Ошибка");
return;
        }
// год рождения должен вводиться цифрами
if (!char.IsDigit(textBox2.Text[0])) {
textBox2.Text = "1940";
MessageBox.Show("Введите год рождения в поле 'г.д.'!","Ошибка");
return;
        }
if (!char.IsDigit(textBox3.Text[0])) {
textBox3.Text = "0";
MessageBox.Show("Введите цифру в поле 'Оклад'!","Ошибка");
return;
        }
/* Тестируем наличие файла с данными о сотрудниках и создаем файл для записи значений */
FileStream fin;
try {
fin=new FileStream("Сотрудники.txt",FileMode.OpenOrCreate);
fin.Close();  }
catch(IOException exc)  {
```

```
MessageBox.Show ("Невозможно открыть файл.","Ошибка");
return;}
//форматируем значения
string s_field1=textBox1.Text;
if (s_field1.Length<l_name) {
for (int i=s_field1.Length; i<l_name; i++)
 s_field1=s_field1+" ";
        }
string s_field2="|"+textBox2.Text; //  пользовательский разделитель данных!
if (s_field2.Length<l_year) {
for (int i=s_field2.Length; i<l_year; i++)
 s_field2=s_field2+" ";
        }
string s_field3="|"+textBox3.Text; //  пользовательский разделитель данных!
StreamWriter fin_out; // открываем файл для записи
try {fin_out=new StreamWriter("Сотрудники.txt", true);}
catch(IOException exc)  {
MessageBox.Show ("Невозможно открыть файл.","Ошибка");
return;}
string s=s_field1+s_field2+s_field3+"\r\n";
fin_out.Write(s);
fin_out.Close();
MessageBox.Show(s_field1+"\n"+s_field2+" г.р.\n"+s_field3+
" у.е.- оклад.","Запись добавлена");
// Обнуляем поля
textBox1.Text="";
textBox2.Text="1940";
textBox3.Text="0";
}
```

## 2.2. Поиск в текстовых файлах

Задача темы: изучить методику и получить навыки в разработке функциональности приложений, использующих загрузку данных из текстового файла в форму для просмотра, поиска и выборки записей по ключу, добавление и сохранение данных в текстовом файле.

Разработка: Приложение «Поиск записей в текстовом файле» для автоматического подсчета средней зарплаты в выборке.

Состав выполняемых функций:

- Файловые операции: открыть, сохранить, закрыть
- Операции с записями: добавить, найти, выбрать
- Алгоритмы: подсчет средней зарплаты в выборке
- Информационно-справочные: О программе

Наращивание функциональности не предусматривается.

Защита данных предусматривается только от неквалифицированных действий пользователей. Редактирование – вручную с помощью редактора «Блокнот».

В качестве входных данных используется отдельный текстовой файл Сотрудники.txt в котором хранится база данных отдела кадров предприятия. Каждая строка файла содержит запись об одном сотруднике. Формат записи: Фамилия и инициалы – 18 символов, начиная с первой позиции; год рождения – 6 позиций; оклад – неограниченно. Тип данных определяется программно. Выходные данные программы – графическое отображение результатов в виде листбоксов и набора текстовых полей экранных форм для SmartDevice.

**Шаг 1. Разработка графического интерфейса**

Для программы используются три формы, примерный дизайн которых показан на рис.9, 10 и 11.

Главная форма (Form1) на рис.9 содержит элементы:

- 3 надписи (label) со свойствами Text: В текстовом файле Сотрудники.txt хранится база данных отдела кадров предприятия. Каждая стро-

ка файла содержит запись об одном сотруднике. Формат записи: фамилия и инициалы (18 позиций, фамилия должна начинаться с первой позиции), год рождения (6 позиций), оклад (неограничено); Программа по заданной фамилии выводит на экран сведения о сотруднике, подсчитывая средний оклад всех запрошенных сотрудников; (С)Родигин Л.А., 2006. Платформа MS Visual C#.Net 2003;

- главное меню (mainMenu) с двумя опциями: «База данных» и «Новая запись».

Форма «База данных» (Form2) на рис.10 содержит элементы:

- 5 надписей (label) со свойствами Text: Фамилия и инициалы; г.р. (год рождения); Оклад; Click для удаления (записи из выборки); Ср.оклад в выборке, у.е.;
- 2 листбокса (listBox) – для базы данных (верхний) и для выборки данных (нижний) из базы;
- 1 текстовое поле для ввода фамилии в качестве ключа поиска в базе данных;
- 1 кнопку «Найти и добавить» активизации поиска и выборки данных

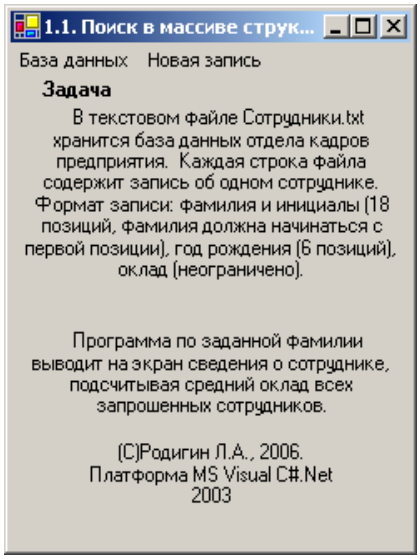


Рис.9 Главная форма

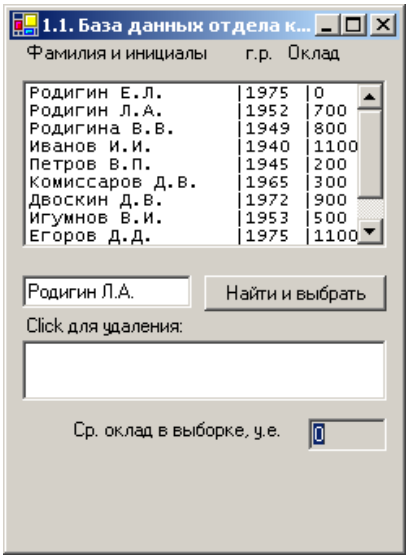


Рис.10 База данных

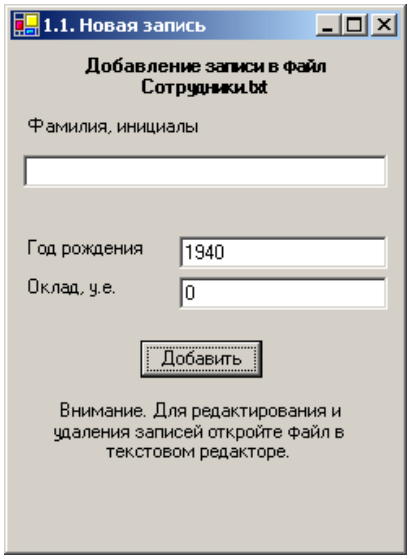


Рис.11 Новая запись

Форма «Новая запись» (Form3) на рис.11 содержит элементы:

- 5 надписей со свойствами Text: Добавление записи в файл Сотрудники.txt; Фамилия, инициалы; Год рождения; Оклад, у.е.; Внимание. Для редактирования и удаления записей откройте файл в текстовом редакторе;
- 3 текстовых поля (textBox);
- Кнопку «Добавить» для добавления записи в файл Сотрудники.txt

**Шаг 2. Открываем форму «База данных»**

Для опции меню «База данных» (menuItem1_Click) пишем:

```
private void menuItem1_Click(object sender, System.EventArgs e)
{// Открываем форму базы данных
Form DB=new Form2();
DB.Show(); // выводим форму на экран
}
```

**Шаг 3. Открываем форму «Новая запись»**

Для опции меню «Новая запись» (menuItem2_Click) пишем:

```
private void menuItem2_Click(object sender, System.EventArgs e)
{// Открываем форму ввода новой записи
Form RW=new Form3();
RW.Show();
}
```

**Шаг 4. Расширяем состав библиотек формы 2**

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
// добавляем библиотеку работы с файлами
using System.IO;
```

**Шаг 5. Включаем переменные для хранения результатов вычислений в public class Form2 : System.Windows.Forms.Form**

```
string s_new; // переменная для хранения данных нового сотрудника
int i_row=0; // число строк в выборке
int i_sum=0; // сумма выборки
```

**Шаг 6. Загружаем данные из файла (по умолчанию)**

```
public Form2() {
InitializeComponent();
// Тестируем наличие файла с данными и создаем файл для записи значений
FileStream fin;
Try {
fin=new FileStream("Сотрудники.txt",FileMode.Open);
        }
catch(IOException exc)  {
MessageBox.Show ("Невозможно открыть файл.","Ошибка");
return;
        }
// Считываем значения из файла
string s;
StreamReader fstr_in=new StreamReader(fin);
while((s=fstr_in.ReadLine())!=null)   {
listBox1.Items.Add(s); // заполняем верхний листбокс
          }
fstr_in.Close(); // закрываем файловый поток
}
```

**Шаг 7. Создаем функцию вычисления средней в выборке**

```
void AppListBox2() {
// вычисляем длину выбранной строки из листбокса
int i_len=s_new.Length;
string s_pay=s_new.Substring(25,i_len-25);
i_row=i_row+1; // считаем количество строк
// преобразуем текст в сумму и считаем сумму выборки
```

```
i_sum=int.Parse(s_pay)+i_sum;
float f_sum=i_sum/i_row; // считаем среднюю
textBox2.Text=f_sum.ToString(); // выводим среднюю в текстовое поле
}
```

**Шаг 8. Создаем процедуру выборки значения из верхнего листбокса в нижний по щелчку**

```
private void listBox1_SelectedIndexChanged(object sender, System.EventArgs e)
{
s_new=listBox1.SelectedItem.ToString();
listBox2.Items.Add(s_new); // добавляем запись в листбокс2
AppListBox2(); // вызваем функцию вычисления средней в выборке
}
```

**Шаг 9. Создаем процедуру удаления записей из выборки и пересчета средней по оставшимся в выборке записям**

```
private void listBox2_SelectedIndexChanged(object sender, System.EventArgs e)
{// если в выборке есть еще записи, которые можно удалить
if (listBox2.SelectedItem != null) {
// вычисляем длину выбранной строки из листбокса
int i_len=listBox2.SelectedItem.ToString().Length;
string s_pay=listBox2.SelectedItem.ToString().Substring(25,i_len-25);
// считаем количество строк
i_row=i_row-1;
// преобразуем текст в сумму и считаем сумму выборки
i_sum=-int.Parse(s_pay)+i_sum;
// считаем среднюю
float f_sum;
if (i_row = = 0) {f_sum=0;}
else    {f_sum=i_sum/i_row;}
textBox2.Text=f_sum.ToString();
// удаляем строки из listbox2
```

```
listBox2.Items.Remove(listBox2.SelectedItem.ToString());
        }
}
```

**Шаг 10. Создаем процедуру поиска при активизации кнопки**

```
private void button1_Click(object sender, System.EventArgs e) {
// вычисляем длину выбранной строки из TEXTбокса
int i_len=textBox1.Text.Length;
int i_row=listBox1.Items.Count+1;
//цикл просмотра списка верхнего листбокса
for(int i=0; i<listBox1.Items.Count;i++) {
// сравниваем значения текстбокса и листбокса
string s_name=listBox1.Items[i].ToString().Substring(0,i_len);
if (textBox1.Text==s_name)     {i_row=i;}
        }
if (i_row<=listBox1.Items.Count) {
s_new=listBox1.Items[i_row].ToString();
listBox2.Items.Add(s_new);
AppListBox2();
        }
else    {MessageBox.Show(textBox1.Text,"Не найдено");}
}
```

**Шаг 11. Расширяем состав библиотек формы 3**

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
// добавляем библиотеку работы с файлами
using System.IO;
```

**Шаг 12. Включаем константы размеров полей базы данных в public class Form3 : System.Windows.Forms.Form**

```
const int l_name=18;
const int l_year=6;
```

**Шаг 13. Создаем процедуру добавления записи в файл Сотрудники.txt для кнопки button1**

```
private void button1_Click(object sender, System.EventArgs e) {
// Проверка значений полей
if (textBox1.Text = = "") {
MessageBox.Show("Введите фамилию и инициалы сотрудника!","Ошибка");
return;
        }
// год рождения должен вводиться цифрами
if (!char.IsDigit(textBox2.Text[0])) {
textBox2.Text = "1940";
MessageBox.Show("Введите год рождения в поле 'г.д.'!","Ошибка");
return;
        }
if (!char.IsDigit(textBox3.Text[0])) {
textBox3.Text = "0";
MessageBox.Show("Введите цифру в поле 'Оклад'!","Ошибка");
return;
        }
/* Тестируем наличие файла с данными о сотрудниках и создаем файл для записи значений */
FileStream fin;
try {
fin=new FileStream("Сотрудники.txt",FileMode.OpenOrCreate);
fin.Close();  }
catch(IOException exc)  {
```

```
MessageBox.Show ("Невозможно открыть файл.","Ошибка");
return;}
//форматируем значения
string s_field1=textBox1.Text;
if (s_field1.Length<l_name) {
for (int i=s_field1.Length; i<l_name; i++)
 s_field1=s_field1+" ";
        }
string s_field2="|"+textBox2.Text; //  пользовательский разделитель данных!
if (s_field2.Length<l_year) {
for (int i=s_field2.Length; i<l_year; i++)
 s_field2=s_field2+" ";
        }
string s_field3="|"+textBox3.Text; //  пользовательский разделитель данных!
StreamWriter fin_out; // открываем файл для записи
try {fin_out=new StreamWriter("Сотрудники.txt", true);}
catch(IOException exc)  {
MessageBox.Show ("Невозможно открыть файл.","Ошибка");
return;}
string s=s_field1+s_field2+s_field3+"\r\n";
fin_out.Write(s);
fin_out.Close();
MessageBox.Show(s_field1+"\n"+s_field2+" г.р.\n"+s_field3+
" у.е.- оклад.","Запись добавлена");
// Обнуляем поля
textBox1.Text="";
textBox2.Text="1940";
textBox3.Text="0";
}
```

## 2.4. Сортировка записей в текстовом файле

Задача темы: изучить методику и получить навыки в разработке функциональности приложений, использующих данные текстового файла для просмотра и сортировки по шаблону.

Разработка: Приложение «Сортировка списка» для просмотра и сортировки текстового списка по шаблону.

Состав выполняемых функций:

- Файловые операции: открыть, сохранить, закрыть
- Алгоритмы: преобразование вводимого текста в верхний регистр, добавление и сохранение записей в файле, удаление записей из загруженного списка, сортировка записей загруженного списка по шаблону
- Информационно-справочные: нет

Наращивание функциональности не предусматривается.

Защита данных предусматривается только от неквалифицированных действий пользователей.

В качестве входных данных используется текстовой файл Список.txt. Выходные данные программы – графическое отображение результатов в листбоксах – для исходного и отсортированных списков, в форме для SmartDevice.

**Шаг 1. Разработка графического интерфейса**

Используется только одна форма Form1 с текстовым полем, кнопкой добавления записей в текстовой файл, кнопкой загрузки данных текстового файла в listBox1 и кнопкой сортировки и загрузки данных из listBox1 в listBox2. Примерный дизайн формы приведен на рис.13.

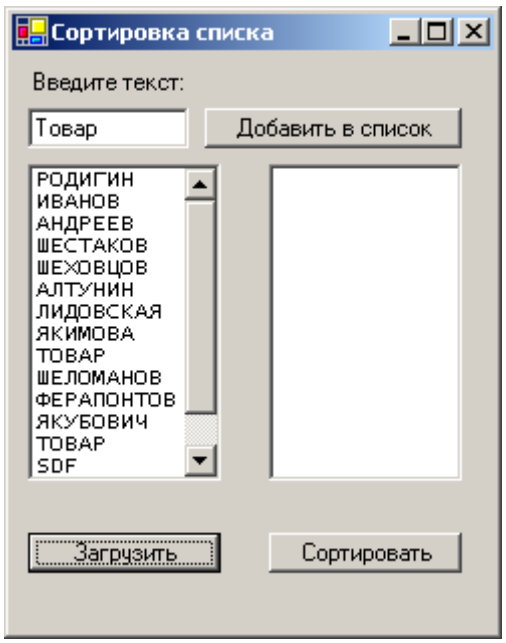


Рис.13

**Шаг 2. . Расширяем состав системных библиотек**

```
using System;
using System.Drawing;
using System.Collections;
using System.Windows.Forms;
using System.Data;
//добавляем библиотеку
using System.IO;
```

**Шаг 3. Тестируем наличие файла Список.txt в public Form1()**

```
InitializeComponent(); // после этой строки вставляем:
/* Тестируем наличие файла с данными о сотрудниках и создаем файл для
записи значений
*/
FileStream fin;
try {
fin=new FileStream("Список.txt",FileMode.OpenOrCreate);
        fin.Close();
        }
catch(IOException exc)  {
```

```
MessageBox.Show ("Невозможно открыть файл.","Ошибка");
return;
        }
```

**Шаг 4. Создаем процедуру ввода новых записей в файл**

Для кнопки «Добавить в список» пишем:

```
private void button1_Click_1(object sender, System.EventArgs e)
{// Проверяем пустое поле
if (textBox1.Text!=" ")    {// если поле не пустое
//Автоматически переводим текст в верхний регистр
string s=textBox1.Text.Trim().ToUpper()+"\r\n";
// открываем файл для записи
StreamWriter fin_out;
try     {
fin_out=new StreamWriter("Список.txt", true);
        }
catch(IOException exc)  {
MessageBox.Show ("Невозможно открыть файл.","Ошибка");
return;}
fin_out.Write(s); // записываем в файл новую строку
fin_out.Close();
MessageBox.Show(s,"Запись добавлена");
textBox1.Text=" "; // обнуляем текстовое поле
        }
else    {
MessageBox.Show("Введите текст!","Ошибка");
return;}
}
```

**Шаг 5. Создаем процедуру загрузки данных**

Для кнопки «Загрузить» пишем:

```
private void button3_Click(object sender, System.EventArgs e) {
```

```
FileStream fin;
try     {
fin=new FileStream("Список.txt",FileMode.Open);
        }
catch(IOException exc)  {
MessageBox.Show ("Невозможно открыть файл.","Ошибка");
return;}
// Считываем значения из файла
string s;
StreamReader fstr_in=new StreamReader(fin);
while((s=fstr_in.ReadLine())!=null)   {
listBox1.Items.Add(s.Trim().ToUpper());
        }
fstr_in.Close();
}
```

**Шаг 6. Создаем процедуру удаления записей из listBox1**

Для удаления записи из листбокса щелчком мыши пишем:

```
private void listBox1_SelectedIndexChanged(object sender, System.EventArgs e)
{// если листбокс не пустой
if (listBox1.SelectedItem != null) // удаляем запись
listBox1.Items.Remove(listBox1.SelectedItem.ToString());
}
```

**Шаг 7. Создаем процедуру сортировки по шаблону**

Для кнопки «Сортировать» пишем:

```
private void button2_Click(object sender, System.EventArgs e) {
listBox2.Items.Clear(); // чистим листбокс 2
if (listBox1.Items.Count!=0)     {/* если число сторк в листбокс 1 не равно 0,
создаем константу шаблона сортировки. Кстати, константу можно создать и в
другом месте программы.
*/
```

```
const string s_alfa="АБВГДЕЖЗИКЛМНОПРСТУФХЦШЩЭЮЯ";
// вычисляем длину выбранной строки из алфавита
for (int it=0;it<s_alfa.Length; it++)    {
string s_sym=s_alfa.Substring(it,1);
string s_name=s_sym;
// Вложенный цикл поиска повторений do-while
        do { // вложенный цикл просмотра списка листбокса
        for(int i=0; i<listBox1.Items.Count;i++)
// сравниваем значения алфавита и листбокса
        {s_name=listBox1.Items[i].ToString().Substring(0,1);
        if (s_sym==s_name) {
                int i_row=i;
                if (i_row<=listBox1.Items.Count)      {
                string s_new=listBox1.Items[i_row].ToString();
                listBox2.Items.Add(s_new);
                listBox1.Items.Remove(listBox1.Items[i_row].ToString());
                        }
                }
        }
}
while (s_sym==s_name & listBox1.Items.Count!=0);
        }
}
        else    {// если листбокс1 пустой, то
        MessageBox.Show("Введите текст!","Ошибка");
        return;}
}
```

## 2.4. Сортировка записей в текстовом файле

Задача темы: изучить методику и получить навыки в разработке функциональности приложений, использующих данные текстового файла для просмотра и сортировки по шаблону.

Разработка: Приложение «Сортировка списка» для просмотра и сортировки текстового списка по шаблону.

Состав выполняемых функций:

- Файловые операции: открыть, сохранить, закрыть
- Алгоритмы: преобразование вводимого текста в верхний регистр, добавление и сохранение записей в файле, удаление записей из загруженного списка, сортировка записей загруженного списка по шаблону
- Информационно-справочные: нет

Наращивание функциональности не предусматривается.

Защита данных предусматривается только от неквалифицированных действий пользователей.

В качестве входных данных используется текстовой файл Список.txt. Выходные данные программы – графическое отображение результатов в листбоксах – для исходного и отсортированных списков, в форме для SmartDevice.

**Шаг 1. Разработка графического интерфейса**

Используется только одна форма Form1 с текстовым полем, кнопкой добавления записей в текстовой файл, кнопкой загрузки данных текстового файла в listBox1 и кнопкой сортировки и загрузки данных из listBox1 в listBox2. Примерный дизайн формы приведен на рис.13.

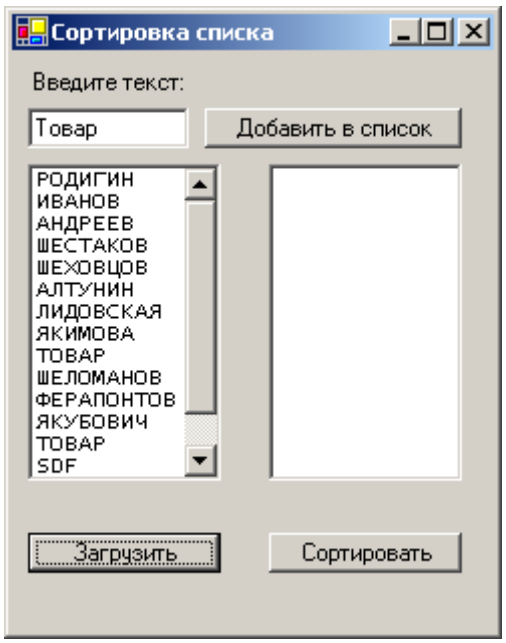


Рис.13

**Шаг 2. . Расширяем состав системных библиотек**

```
using System;
using System.Drawing;
using System.Collections;
using System.Windows.Forms;
using System.Data;
//добавляем библиотеку
using System.IO;
```

**Шаг 3. Тестируем наличие файла Список.txt в public Form1()**

```
InitializeComponent(); // после этой строки вставляем:
/* Тестируем наличие файла с данными о сотрудниках и создаем файл для
записи значений
*/
FileStream fin;
try {
fin=new FileStream("Список.txt",FileMode.OpenOrCreate);
        fin.Close();
        }
catch(IOException exc)  {
```

```
MessageBox.Show ("Невозможно открыть файл.","Ошибка");
return;
        }
```

**Шаг 4. Создаем процедуру ввода новых записей в файл**

Для кнопки «Добавить в список» пишем:

```
private void button1_Click_1(object sender, System.EventArgs e)
{// Проверяем пустое поле
if (textBox1.Text!=" ")    {// если поле не пустое
//Автоматически переводим текст в верхний регистр
string s=textBox1.Text.Trim().ToUpper()+"\r\n";
// открываем файл для записи
StreamWriter fin_out;
try     {
fin_out=new StreamWriter("Список.txt", true);
        }
catch(IOException exc)  {
MessageBox.Show ("Невозможно открыть файл.","Ошибка");
return;}
fin_out.Write(s); // записываем в файл новую строку
fin_out.Close();
MessageBox.Show(s,"Запись добавлена");
textBox1.Text=" "; // обнуляем текстовое поле
        }
else    {
MessageBox.Show("Введите текст!","Ошибка");
return;}
}
```

**Шаг 5. Создаем процедуру загрузки данных**

Для кнопки «Загрузить» пишем:

```
private void button3_Click(object sender, System.EventArgs e) {
```

```
FileStream fin;
try     {
fin=new FileStream("Список.txt",FileMode.Open);
        }
catch(IOException exc)  {
MessageBox.Show ("Невозможно открыть файл.","Ошибка");
return;}
// Считываем значения из файла
string s;
StreamReader fstr_in=new StreamReader(fin);
while((s=fstr_in.ReadLine())!=null)   {
listBox1.Items.Add(s.Trim().ToUpper());
        }
fstr_in.Close();
}
```

**Шаг 6. Создаем процедуру удаления записей из listBox1**

Для удаления записи из листбокса щелчком мыши пишем:

```
private void listBox1_SelectedIndexChanged(object sender, System.EventArgs e)
{// если листбокс не пустой
if (listBox1.SelectedItem != null) // удаляем запись
listBox1.Items.Remove(listBox1.SelectedItem.ToString());
}
```

**Шаг 7. Создаем процедуру сортировки по шаблону**

Для кнопки «Сортировать» пишем:

```
private void button2_Click(object sender, System.EventArgs e) {
listBox2.Items.Clear(); // чистим листбокс 2
if (listBox1.Items.Count!=0)     {/* если число сторк в листбокс 1 не равно 0,
создаем константу шаблона сортировки. Кстати, константу можно создать и в
другом месте программы.
*/
```

```
const string s_alfa="АБВГДЕЖЗИКЛМНОПРСТУФХЦШЩЭЮЯ";
// вычисляем длину выбранной строки из алфавита
for (int it=0;it<s_alfa.Length; it++)    {
string s_sym=s_alfa.Substring(it,1);
string s_name=s_sym;
// Вложенный цикл поиска повторений do-while
        do { // вложенный цикл просмотра списка листбокса
        for(int i=0; i<listBox1.Items.Count;i++)
// сравниваем значения алфавита и листбокса
        {s_name=listBox1.Items[i].ToString().Substring(0,1);
        if (s_sym==s_name) {
                int i_row=i;
                if (i_row<=listBox1.Items.Count)      {
                string s_new=listBox1.Items[i_row].ToString();
                listBox2.Items.Add(s_new);
                listBox1.Items.Remove(listBox1.Items[i_row].ToString());
                        }
                }
        }
}
while (s_sym==s_name & listBox1.Items.Count!=0);
        }
}
        else    {// если листбокс1 пустой, то
        MessageBox.Show("Введите текст!","Ошибка");
        return;}
}
```

## 2.6. Сортировка массива

Задача темы: изучить методику и получить навыки в разработке функциональности приложений, использующих сортировки текстовых данных. В отличие от предыдущей задачи, в которой сортировались текстовые строки, состоящие только из цифр, здесь ставится задача сортировки строки из букв.

Разработка: Приложение «Сортировка » для сортировки массива переменных типа Char с использованием функции сортировки, определенной в отдельном классе.

Состав выполняемых функций:

- Файловые операции: нет
- Алгоритмы: Quick-сортировка
- Информационно-справочные:нет

Наращивание функциональности не предусматривается.

Защита данных не предусматривается.

В качестве входных данных используется текстовая строка. Выходные данные программы – графическое отображение результатов в виде текстового поля и листбокса в форме для SmartDevice.

### Шаг 1. Разработка графического интерфейса

Используется одна форма с двумя текстовыми полями – одно для ввода (указываем свойство Text= Исходный массив), другое и листбокс – для вывода отсортированного массива. Предусматриваются кнопка запуска процедуры сортировки «Сортировать» и кнопка очистки поля ввода «Очистить». Примерный дизайн показан на рис. 15.

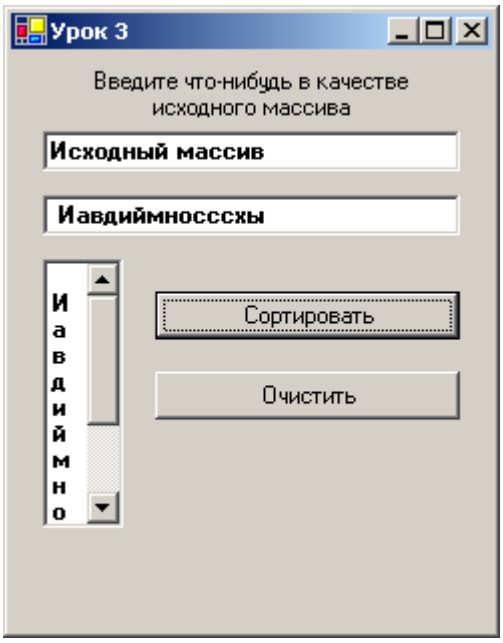


Рис.15

**Шаг 2. Добавляем в проект новый класс**

Новому классу присваиваем имя Quicksort (хотя это и необязательно – можно использовать имя по умолчанию Class1)

**Шаг 3. Создаем функцию сортировки qs в public class Quicksort**

Так как функция сортировки массива определена в отдельном классе, то для того, чтобы сделать ее доступной из другого класса используем определение static. Вместо public Quicksort() пишем::

```
static void qs(char[] items,int left,int right) {
int i,j;
char x,y;
i=left; j=right;
x=items[(left+right)/2];
do { while((items[i]<x)&&(i<right)) i++;
        while((x<items[j])&&(j>left)) j--;
        if(i<=j) {y=items[i];
                items[i]=items[j];
                items[j]=y;
                i++; j--;
                }
        }
while(i<=j);
if(left<j) qs(items,left,j); // рекурсивный вызов функции сортировки Квик
if(i<right) qs(items,i,right);
}
```

**Шаг 4. Создаем функцию qsort для рекрсивного вызова функции сортировки qs в public class Quicksort**

```
public static void qsort(char[] items)
{//вызывается функция  qs с параметрами исходного массива Char
qs(items,0,items.Length-1);
}
```

### Шаг 5. Создаем процедуру запуска сортировки

Для кнопки «Сортировать» класса Form1 пишем:

```
private void button1_Click(object sender, System.EventArgs e) { /* Так как string это сумма char, то нужно выполнить конвертацию посимвольно */
string CharValidate =this.textBox1.Text; /* задаем строковую переменную, в которую считываем значения из текстового поля и преобразуем строку в массив кодов символов */
char[] a=CharValidate.ToCharArray();
int i;
string s_old="";
for(i=0;i<a.Length;i++)
s_old=s_old+a[i].ToString();
textBox1.Text=s_old;
Quicksort.qsort(a); //вызов функции - срабатывает подсказка редактора!
string s_new="";
for(i=0;i<a.Length; i++)
s_new=s_new+a[i].ToString();
textBox2.Text=s_new; // выводим результат сортировки в textBox2
// а теперь запускаем цикл построчного заполнения листбокса
for(i=0;i<a.Length; i++)
listBox1.Items.Add(a[i]);
}
```

### Шаг 6. Создаем процедуру очистки результатов

Для кнопки «Очистить» пишем:

```
private void button2_Click(object sender, System.EventArgs e) {
textBox1.Text="Исходный массив";
textBox2.Text="Отсортированный массив";
listBox1.Items.Clear();
}
```

## 2.7. Добавление-удаление записей

Задача темы: изучить методику и получить навыки в разработке приложений для управления базами данных в текстовом формате.

Разработка: Приложение «Корзина покупателя» для автоматического подсчета стоимости отобранных товаров в корзину (тележку) в универсаме.

Состав выполняемых функций:

- Файловые операции: открыть, сохранить, закрыть
- Алгоритмы: работа с записями базы - добавление, удаление, сортировка, поиск, фильтрация; калькуляция стоимости корзины
- Информационно-справочные: О программе

Наращивание функциональности не предусматривается.

Защита данных только от неквалифицированных действий пользователя.

В качестве входных данных используется текстовой файл «Товары.txt». Выходные данные программы – графическое отображение результатов в виде текстовых полей, листбоксов и форм для SmartDevice.

### Шаг 1. Структура данных

Структура данных генерируется изначально в программе и включает текстовые поля «Товар», «Производитель», «Количество», «Цена». Вид текстового файла с фрагментом заполненной базы с данными о товарах в редакторе «Блокнот» показан на рис. 16.

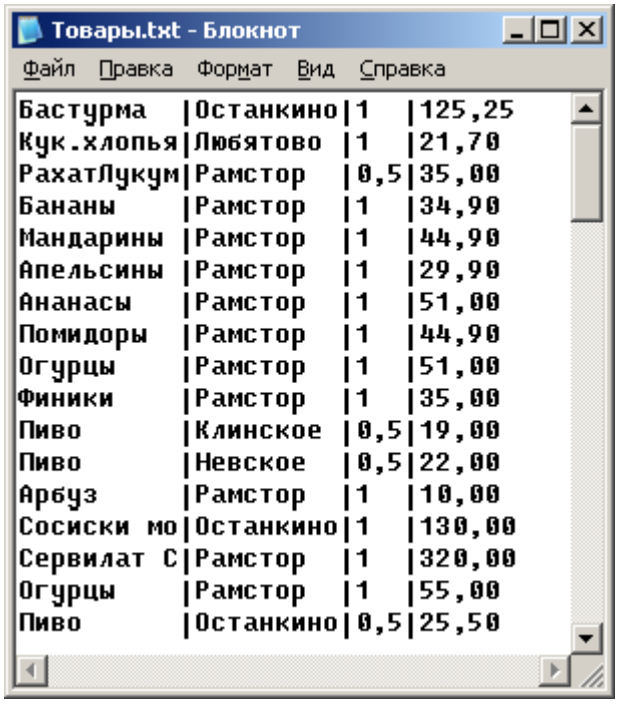


Рис. 16

## Шаг 2. Разработка графического интерфейса

Графический интерфейс включает две формы: «Корзина покупателя» (Form1) – рис. 17, 18, 19 и «Удаление записи» (Form2).- рис. 20. Добавляем в проект вторую форму.

В форме «Корзина покупателя» (свойства MaximizeBox=False, MinimizeBox=False) размещены главное меню с единственной опцией «О программе» и TabControl с тремя закладками: «Редактор», «Товары», «Корзина».

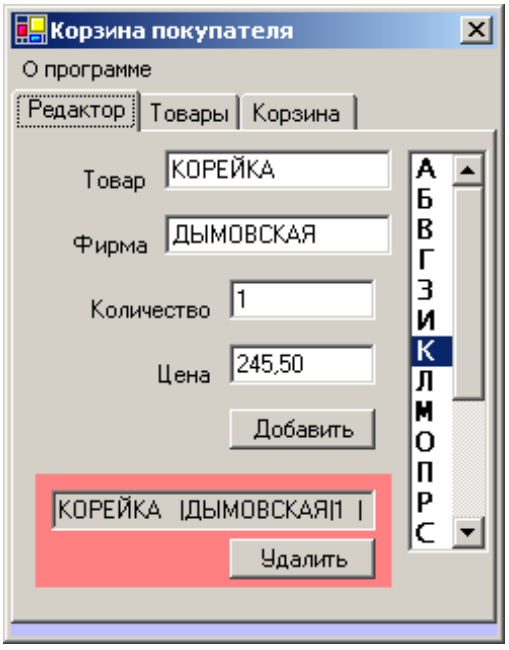


Рис.17

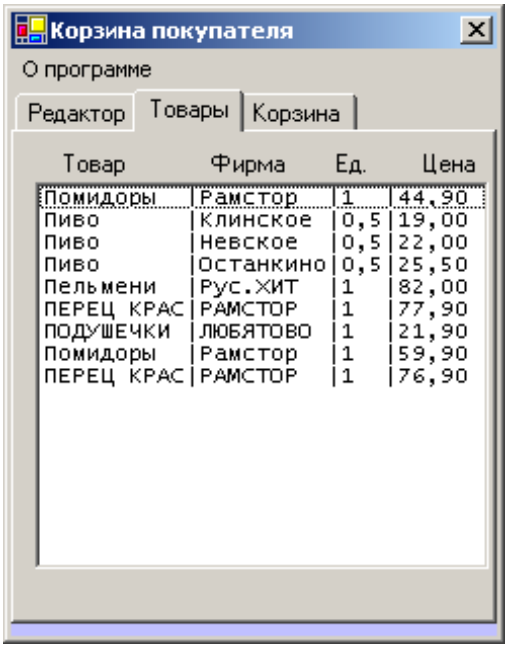


Рис.18

На закладке «Редактор» размещены 4 текстовых поля (textBox) с надписями (label) «Товар», «Фирма», «Количество», «Цена»; Поле для удаления записи (Свойство ReadOnly = True); Кнопка удаления записи «Удалить» и Листбокс для поиска товара – примерный дизайн показан на рис.17.

На закладке «Товары» размещен листбокс для просмотра базы данных и 4 надписи над столбцами листбокса: «Товар», «Фирма», «Количество», «Цена» – примерный дизайн показан на рис.18.

На закладке «Корзина» размещен листбокс выбранных из базы товаров и 4 надписи над столбцами листбокса: «Товар», «Фирма», «Количество», «Цена». Еще одна надпись «ВСЕГО, у.е.» размещена рядом с текстовым по-

лем результатов калькуляции стоимости корзины – примерный дизайн показан на рис.19.

В форме «Удаление записи» (Свойство ControlBox=False) размещены 3 надписи «Вы действительно хотите удалить эту запись?», «Чтение…» (Свойство Visible = False), «Запись…» (Свойство Visible = False); текстовое поле вывода значения удаляемой записи (Свойство ReadOnly = True); кнопка «Да» и кнопка «Нет»; Листбокс для поиска удаляемой записи в списке (Свойство Visible = False); ProgressBar для индикации перезаписи файла списка товаров. Примерный дизайн видимых элементов показан на рис.20.

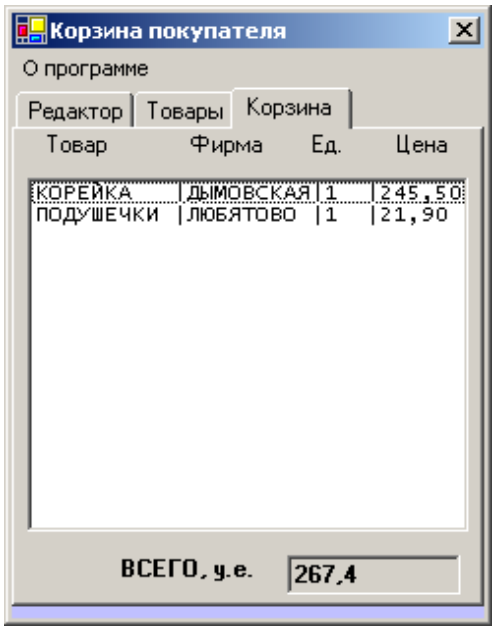


Рис.19

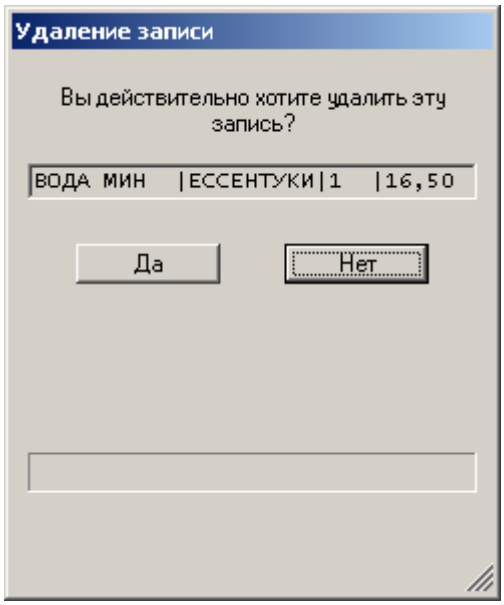


Рис.20

**Шаг 3. Добавляем библиотеки в оба класса форм**

```
using System;
using System.Drawing;
using System.Collections;
using System.Windows.Forms;
using System.Data;
//добавляем библиотеку
using System.IO;
```

**Шаг 4. Объявляем общие переменные в public class Form1 : System.Windows.Forms.Form**

```
// объявляем глобальную переменную для использования в форме 2
public static string ss;
float i_sum=0; // стоимость товара в корзине
int L_goods=10; // длина поля «Товар»
int L_firma=10; // длина поля «Фирма»
int L_unit=4; // Длина поля «Количество»
string s_new; // строка обозначения товара
```

**Шаг 5. Для public Form1(), загружаемой первой по умолчанию**

Тестируем наличие файла базы данных - дописываем:

```
InitializeComponent();
// если файла с данными о ценах нет, то создаем файл для записи значений
FileStream fin;
try {
fin=new FileStream("Товары.txt",FileMode.Open);
        }
catch(IOException exc) {
MessageBox.Show ("Невозможно открыть файл.","Ошибка");
return;}
// Считываем значения из файла
string s;
StreamReader fstr_in=new StreamReader(fin);
while((s=fstr_in.ReadLine())!=null) {
listBox1.Items.Add(s); // грузим базу в листбокс
}
fstr_in.Close(); // и закрываем поток
// вызываем функцию для заполняем листбокса поиска товара
appe_listbox3(); // вообще-то функции пока нет
```

**Шаг 6. Создаем функцию алфавита appe_listbox3**
**из имеющихся названий товара**

```
private void appe_listbox3() {
listBox3.Items.Clear(); // для вывода результата используем listBox3
const string s_alfa="АБВГДЕЖЗИКЛМНОПРСТУФХЦЧШЩЭЮЯ";
string s,s_name,s_sym;
// вычисляем длину выбранной строки из алфавита
for (int it=0;it<s_alfa.Length; it++) {
s_sym=s_alfa.Substring(it,1);
// Тестируем наличие файла с данными о ценах
FileStream fin;
try {
fin=new FileStream("Товары.txt",FileMode.Open);
        }
catch(IOException exc) {
MessageBox.Show ("Невозможно открыть файл.","Ошибка");
return;}
// бежим по файлу
StreamReader fstr_in=new StreamReader(fin);
while ((s=fstr_in.ReadLine())!=null ) {
s_name=s.Substring(0,1);
        if (s_name==s_sym) {
        listBox3.Items.Add(s_name);
        s_sym=""; // обнуляем, чтобы избежать повторов
                }
        }
fstr_in.Close();
}
```

### Шаг 7. Добавляем запись в базу данных

Для кнопки «Добавить» в закладке «Редактор» создаем процедуру:

```
private void button1_Click(object sender, System.EventArgs e) {
// Проверка значений полей
if (textBox2.Text=="") {
MessageBox.Show("Введите товар!","Ошибка");
return;}
if (textBox3.Text=="") {
MessageBox.Show("Введите фирму!","Ошибка");
return;}
if (textBox4.Text=="") {
MessageBox.Show("Введите количество!","Ошибка");
return;}
if (!char.IsDigit(textBox5.Text[0])) {
textBox5.Text = "0,00";
MessageBox.Show("Введите цену!","Ошибка");
return;}
// Тестируем наличие файла с данными
FileStream fin;
try {
fin=new FileStream("Товары.txt",FileMode.OpenOrCreate);
fin.Close();  }
catch(IOException exc) {
MessageBox.Show ("Невозможно открыть файл.","Ошибка");
return; }
//форматируем значения
string s_field1=textBox2.Text;
if (s_field1.Length<L_goods) {
        for (int i=s_field1.Length; i<L_goods; i++)
        s_field1=s_field1+" ";
```

```
        }
else { //обрезаем строку
        int L=s_field1.Length;
        s_field1=s_field1.Remove(L_goods,L-L_goods);
        }
string s_field2="|"+textBox3.Text; // вставляем разделитель полей
if (s_field2.Length<L_firma) {
        for (int i=s_field2.Length; i<L_firma; i++)
        s_field2=s_field2+" ";
        }
else { //обрезаем строку
        int L=s_field2.Length;
        s_field2=s_field2.Remove(L_firma,L-L_firma);
        }
string s_field3="|"+textBox4.Text;
        if (s_field3.Length<L_unit) {
        for (int i=s_field3.Length; i<L_unit; i++)
        s_field3=s_field3+" ";
        }
else { //обрезаем строку
        int L=s_field3.Length;
        s_field3=s_field3.Remove(L_unit,L-L_unit);
        }
string s_field4="|"+textBox5.Text;
// Записываем значения в файл
StreamWriter fin_out;
try {
fin_out=new StreamWriter("Товары.txt", true);
        }
catch(IOException exc) {
```

```
MessageBox.Show ("Невозможно открыть файл.","Ошибка");
return;}
string s=s_field1+s_field2+s_field3+s_field4+"\r\n";
fin_out.Write(s);
fin_out.Close();
MessageBox.Show(s_field1+"\n"+s_field2+"\n"+s_field3+"\n"+
        s_field4,"Запись добавлена");
// Обнуляем поля
textBox2.Text="";
textBox3.Text="";
textBox4.Text="";
textBox5.Text="0,00";
// заполняем листбокс3
appe_listbox3();
}
```

**Шаг 8. Теперь можно программировать процедуру листбокс на закладке «Редактор» для выборки товара из базы**

```
private void listBox3_SelectedIndexChanged(object sender, System.EventArgs e)
{//выборка групп товаров из файла
listBox1.Items.Clear();
s_new=listBox3.SelectedItem.ToString();
string s;
FileStream fin;
try {
fin=new FileStream("Товары.txt",FileMode.Open);
        }
catch(IOException exc) {
MessageBox.Show ("Невозможно открыть файл.","Ошибка");
return;}
// бежим по файлу
```

```
StreamReader fstr_in=new StreamReader(fin);
while ((s=fstr_in.ReadLine())!=null ) {
if (s_new.Substring(0,1)==s.Substring(0,1)) {
        listBox1.Items.Add(s); // если нашли, то добавляем в листбокс
                }
        }
fstr_in.Close();
//переключаемся на закладку "База данных" для просмотра выборки
this.tabControl1.SelectedIndex=1;
}
```

**Шаг 9. А теперь из выборки отбираем товар в корзину щелчком мыши**

```
private void listBox1_SelectedIndexChanged(object sender, System.EventArgs e)
{//Выбираем строки из листбокса 1
s_new=listBox1.SelectedItem.ToString();
listBox2.Items.Add(s_new);
this.textBox6.Text=s_new;
this.tabControl1.SelectedIndex=2; // смотрим корзинку
edit_row(); // вызываем функцию подсчета стоимости товара в корзинке
} // хотя этой функции пока нет  - напишем.
```

**Шаг 10. Создаем функцию подсчета стоимости товара в корзине**

```
// заполняем поля редактора и считаем корзину покупателя
private void edit_row() {
// вычисляем длину выбранной строки из листбокса
int i_len=s_new.Length;
string s_goods=s_new.Substring(0,L_goods);
string s_firma=s_new.Substring(L_goods+1,L_firma-1);
string s_unit=s_new.Substring(L_goods+L_firma+1,L_unit-1);
string s_price=s_new.Substring(L_goods + L_firma + L_unit + 1,
        i_len-(L_goods + L_firma + L_unit+1));
// преобразуем текст в сумму и считаем сумму выборки
```

```
i_sum=float.Parse(s_price)+i_sum;
textBox1.Text=i_sum.ToString();
textBox2.Text=s_goods;
textBox3.Text=s_firma;
textBox4.Text=s_unit;
textBox5.Text=s_price;
}
```

**Шаг 11. Создаем процедуру удаления записи из корзины по щелчку мыши**

```
private void listBox2_SelectedIndexChanged_1
        (object sender, System.EventArgs e) {
if (listBox2.SelectedItem != null) {
// вычисляем длину выбранной строки из листбокса
int i_len=listBox2.SelectedItem.ToString().Length;
string s_pay = listBox2.SelectedItem.ToString().Substring(L_goods + L_firma +
        L_unit+1, i_len-(L_goods+L_firma+L_unit+1));
// преобразуем текст в сумму и считаем сумму выборки
i_sum=float.Parse(textBox1.Text)-float.Parse(s_pay);
textBox1.Text=i_sum.ToString();
// удаляем строки из listbox2
listBox2.Items.Remove(listBox2.SelectedItem.ToString());
        }
}
```

**Шаг 12. Создаем процедуру вызова формы диалога удаления записи**

Для кнопки «Удалить» в закладке «Редактор» пишем:

```
private void button3_Click(object sender, System.EventArgs e) {
ss=this.textBox6.Text;
if (ss = = "") {
MessageBox.Show("Нет записи для удаления!","Ошибка");
return;}
```

```
//Создаем место в памяти для объекта Форма2
Form Form_Del=new Form2();
// позиционируем форму 2 в форме 1
Form_Del.Parent=this.Parent;
// Вызываем как модальную форму
Form_Del.ShowDialog();
}
```

**Шаг 13. В текстовое поле Form2 переносим значение выбранной для удаления записи**

```
public Form2()
{
InitializeComponent();
this.textBox1.Text=Form1.ss;
}
```

**Шаг 14. Если в Form2 удалять запись раздумали**

```
//Для кнопки «Нет» в форме 2 пишем:
private void button2_Click(object sender, System.EventArgs e) {
this.Close();
}
```

**Шаг 15. Если в Form2 запись решили удалить**

```
// Для кнопки «Да» в форме 2 пишем:
private void button1_Click(object sender, System.EventArgs e) {
this.label2.Visible=true;
//Считываем значения из файла в листбокс 1
// Тестируем наличие файла с данными о ценах
FileStream fin;
try {
fin=new FileStream("Товары.txt",FileMode.Open);
        }
catch(IOException exc) {
```

```
MessageBox.Show ("Невозможно открыть файл.","Ошибка");
return;}
// Считываем значения из файла
string s;
StreamReader fstr_in=new StreamReader(fin);
while((s=fstr_in.ReadLine())!=null) {
listBox1.Items.Add(s);
progressBar2.Value=progressBar2.Value+1;
}
fstr_in.Close();
progressBar2.Value=0;
this.label2.Visible=false;
this.label3.Visible=true;
// Стираем все в файле
StreamWriter fin_out;
try {
fin_out=new StreamWriter("Товары.txt", false);
        }
catch(IOException exc) {
MessageBox.Show ("Невозможно открыть файл.","Ошибка");
return;}
fin_out.Write("");
fin_out.Close();
//переписываем файл
try {
fin_out=new StreamWriter("Товары.txt", true);
        }
catch(IOException exc) {
MessageBox.Show ("Невозможно открыть файл.","Ошибка");
return;}
```

```
if (listBox1.Items.Count!=0)    {
string s_sym=this.textBox1.Text;
string s_name;
for(int i=0; i<listBox1.Items.Count;i++) {
        s_name=listBox1.Items[i].ToString();
        progressBar2.Value=progressBar2.Value+1;
        if (s_sym!=s_name) {
        fin_out.Write(s_name+"\r\n");
                }
        }
fin_out.Close();
this.Close();
        }
}
```

**Шаг 16. Последний штрих – о программе**

Для опции главного меню формы 1 пишем:

```
private void menuItem1Click(object sender, System.EventArgs e) {
MessageBox.Show("Калькулятор потребительской корзины в универсаме,"+
" чтобы не ошибиться на кассе и удачно выбрать товар по соотношению"+
" цена/качество.\n"+
"!!! Рекомендую копию базы Товары.txt хранить и на ПК, и на FlashDisk.\n"+
"Платформа C#.Net FrameWork v1.1 (C)Родигин Л.А. 02-2006",
"О программе");
}
```

# 3. ФОРМАТ XML ДЛЯ SMARTDEVICE

Существуют различные мнения относительно важности XML в разработке программного обеспечения. Компания Microsoft разработала собственную стратегию NET, направленную на использование XML и Web-служб в качестве «прослойки», обеспечивающей такое взаимодействие между компаниями, которое не зависит от использующихся в них операционных систем и механизмов хранения данных. Поскольку XML— это данные, а ADO.NET— механизм работы с данными в .NET, тот факт что ADO.NET поддерживает работу с XML-документами наравне с остальными данными, выглядит вполне естественно. В этом аспекте библиотека ADO.NET является тесно интегрированной с инфраструктурой XML .NET.

ADO.NET-стратегию Microsoft в отношении XML необходимо рассматривать с двух сторон: с одной стороны, Microsoft стремится предоставить те же инструменты для доступа к XML-содержимому, которые используются для доступа к информации, хранящейся в базе данных, с другой — обеспечить в ADO.NET средства преобразования информации базы данных в формат XML. Отметим, что название ADO.NET не совсем корректное, так как ADO.NET не является прямым "наследником" ADO.

Поскольку ADO.NET - всего лишь механизм работы с данными в .NET, то справедливым представляется утверждение, что XML представляет собой просто еще один формат данных.

Во времена ADO/OLE DB для обработки информации из некоего источника данных (например, базы данных) необходимо было создать поставщик OLE DB, который довольно сложен и запутан (вспомните синтаксис ConnectionString разных поставщиков, рассмотренный в части 1). В .NET для доступа к информации из источника, отличного от базы данных, можно либо создать управляемый поставщик, либо воспользоваться преимуществами тесной интеграции ADO.NET с XML.

Ядро такой интеграции - класс DataSet. Можно загружать данные и схему из XML-потока или документа в объект DataSet и наоборот, перманентно сохранять или сериализовать данные или схему из объекта DataSet в XML-поток или документ. ADO синхронизирует объект DataSet с объектом XmlDataDocument. Данные можно модифицировать одновременно, используя любой из классов; все изменения, произведенные с помощью одного из классов, моментально отражаются в другом классе.

Эта глава посвящена поддержке XML в ADO.NET для карманных персональных компьютеров. Вынужден сразу оговориться, что также как и для текстовых форматов, так и для XML возможности использования ADO.NET реализованы для КПК не в полной мере по сравнению с настольными системами. Поддержка XML в .NET обеспечивается интегрированными классами в пяти пространствах имен:

- System.Xml - содержит классы, обеспечивающие стандартизированную поддержку для обработки XML;
- System.Xml.Schema - содержит классы, обеспечивающие стандартизированную поддержку схем на языке XML Schema Definition Language (XSD);
- System.Xml.Serialization - содержит классы, осуществляющие сериализацию объектов в XML-докумснты или потоки;
- System.Xml.XPath - содержит классы, осуществляющие синтаксический разбор и вычисление XPath;
- System.Xml.Xsl - содержит классы, поддерживающие XSL - преобразования.

Чтобы описать модификацию, которые претерпел набор данных, Microsoft предложила XML–формат DiffGram, идентифицирующий текущую и исходную версии данных, позволяя точно воссоздавать содержимое набора данных. Этот формат, благодаря своей XML-сущности, независим от платформы и приложения, однако не имеет широкого употребления или средств интерпретации вне приложений Microsoft .NET [3]

### 3.1. Создание и управление объектами ADO.NET

Задача темы: изучить методику и получить навыки в разработке приложений для управления базами данных в гипертекстовом формате XML.

Разработка: Приложение «Привязка данных к элементам управления Windows» для просмотра данных XML в Windows-форме.

Состав выполняемых функций:

- Файловые операции: открыть, сохранить, закрыть
- Алгоритмы: просмотр и прокрутка записей базы в текстовых полях, списках и таблицах; просмотр отдельной записи
- Информационно-справочные: нет

Наращивание функциональности не предусматривается.

Защита данных – нет.

В качестве входных данных используются программно сформированные объекты. Выходные данные программы – графическое отображение результатов в виде текстовых полей, списков и таблиц для SmartDevice.; файл DataSet.xml

**Шаг 1. Разработка графического интерфейса**

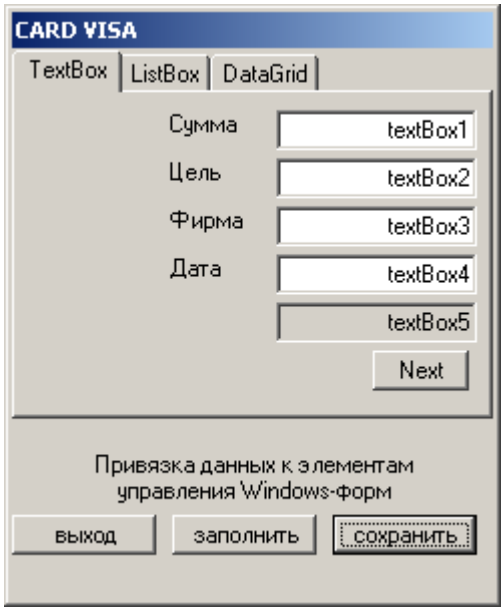


Рис.21

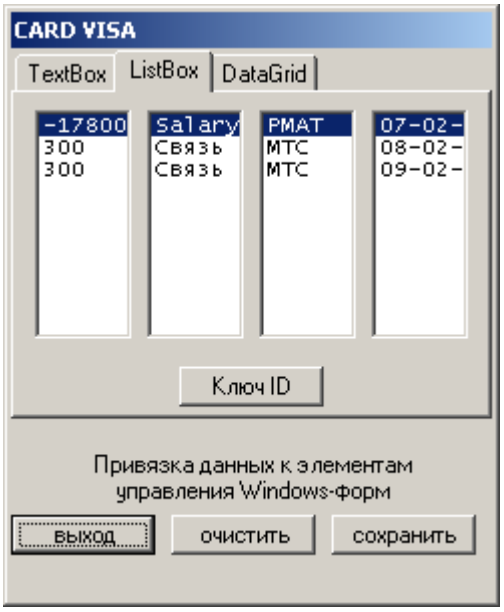


Рис.22

Графический интерфейс строится с использованием одной формы с надписью (label) «Привязка данных к элементам управления Windows-форм»;

тремя кнопками: «Выход», «Заполнить» (изначально свойство Visible =True), «Очистить» (кнопка совмещена с кнопкой «Заполнить и изначально свойство Visible = False, а при загруженных данных в элементы просмотра свойство меняется на Visible =True), «Сохранить»; TabControl из трех закладок – примерный дизайн показан на рис.21, 22, 23.

Закладка TextBox содержит 4 надписи: «Сумма», «Цель», «Фирма», «Дата»; 5 полей, свойство одно из которых ReadOnly=True; кнопку прокрутки записей «Next» - рис.21.

Закладка ListBox содержит 4 листбокса для вывода записей таблицы и кнопку просмотра идентификационного номера записи «Ключ ID» - рис.22

Закладка DataGrid содержит только элемент для просмотра таблиц DataGrid – рис. 23

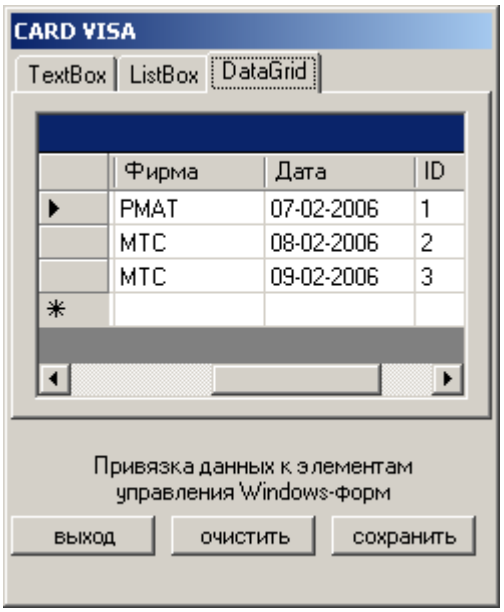


Рис.23

**Шаг 2. Расширяем состав библиотек**

Добавляем библиотеки работы с файлами и с форматами XML

```
using System;
using System.Drawing;
using System.Collections;
using System.Windows.Forms;
using System.Data;
using System.IO; // добавляем библиотеку работы с файлами
using System.Xml; // добавляем библиотеку работы с форматами XML
```

**Шаг 3. Добавляем переменные в состав public class Form1 : System.Windows.Forms.Form**

int i=0; // номер строки для вывода в textbox

**Шаг 4. Создаем процедуру заполнения – очистки элементов просмотра XML – данных**

Для кнопки «Заполнить» («Очистить») пишем:

```
private void button2_Click(object sender, System.EventArgs e) {
DataSet dataSet=new DataSet(); // создаем переменную dataSet
// добавляем таблицу в dataSet
DataTable dataTable=dataSet.Tables.Add("CardVISA");
// программно создаем структуру - добавляем столбцы
dataTable.Columns.Add("Сумма",System.Type.GetType("System.String"));
dataTable.Columns.Add("Цель",System.Type.GetType("System.String"));
dataTable.Columns.Add("Фирма",System.Type.GetType("System.String"));
dataTable.Columns.Add("Дата",System.Type.GetType("System.String"));
// для идентификации записей создаем поле с типом данных
Int64dataTable.Columns.Add("ID",System.Type.GetType("System.Int64"));
// программируем объекты записей
object[] a1={"-17800","Salary","PMAT", "07-02-2006","1"};
object[] a2={"300","Связь","МТС", "08-02-2006","2"};
object[] a3={"300","Связь","МТС", "09-02-2006","3"};
// программно заполняем таблицу, добавляя строки
dataTable.Rows.Add(a1);
dataTable.Rows.Add(a2);
dataTable.Rows.Add(a3);
// привязываем текстовые боксы с указанием i-й строки вывода
BindingContext[dataTable].Position=i;
// привязываем листбокс
listBox1.DataSource=dataTable;
listBox1.DisplayMember="Сумма";
```

```
listBox1.ValueMember="ID";
listBox2.DataSource=dataTable;
listBox2.DisplayMember="Цель";
listBox3.DataSource=dataTable;
listBox3.DisplayMember="Фирма";
listBox4.DataSource=dataTable;
listBox4.DisplayMember="Дата";
// привязываем датагрид
dataGrid1.DataSource=dataTable;
button2.Visible=false; // прячем кнопку «Заполнить»
button3.Visible=true; // показываем кнопку «Очистить»
}
```

**Шаг 5. Создаем процедуру прокрутки записей на закладке «TextBox»**

Для кнопки «Next» закладки «TextBox» пишем:

```
private void button4_Click(object sender, System.EventArgs e) {
// дублируем функцию кнопки «Заполнить»
DataSet dataSet=new DataSet();
DataTable dataTable=dataSet.Tables.Add("CardVISA");
dataTable.Columns.Add("Сумма",System.Type.GetType("System.String"));
dataTable.Columns.Add("Цель",System.Type.GetType("System.String"));
dataTable.Columns.Add("Фирма",System.Type.GetType("System.String"));
dataTable.Columns.Add("Дата",System.Type.GetType("System.String"));
dataTable.Columns.Add("ID",System.Type.GetType("System.Int64"));
object[] a1={"-17800","Salary","PMAT", "07-02-2006","1"};
object[] a2={"300","Связь","MTC", "08-02-2006","2"};
object[] a3={"300","Связь","MTC", "09-02-2006","3"};
dataTable.Rows.Add(a1);
dataTable.Rows.Add(a2);
dataTable.Rows.Add(a3);
// крутим i
```

```
i=i+1;
if (i = =3)
i=0;
BindingContext[dataTable].Position=i;
// чистим текстбоксы
foreach (Control t in this.tabPage1.Controls) {
        if (t is TextBox) {
                t.DataBindings.Clear();
                t.Text="";}
        }
// привязываем текстовые боксы
textBox1.DataBindings.Add("Text",dataTable,"Сумма");
textBox2.DataBindings.Add("Text",dataTable,"Цель");
textBox3.DataBindings.Add("Text",dataTable,"Фирма");
textBox4.DataBindings.Add("Text",dataTable,"Дата");
textBox5.DataBindings.Add("Text",dataTable,"ID");
}
```

**Шаг 6. Создаем процедуру вывода номера записи для закладки «ListBox»**

Для кнопки «Ключ ID» пишем:

```
private void button5_Click(object sender, System.EventArgs e) {
MessageBox.Show(String.Format("{0}",listBox1.SelectedValue),"Ключ ID");
}
```

**Шаг 7. Создаем процедуру очистки элементов просмотра данных**

Для кнопки «Очистить» пишем:

```
private void button3_Click(object sender, System.EventArgs e) {
foreach (Control t in this.tabPage1.Controls) { // чистим текстбоксы
        if (t is TextBox)  {
                t.DataBindings.Clear();
                t.Text="";}
        }
```

```
// чистим листбоксы
listBox1.DataSource=dataTable;
listBox1.Items.Clear();
listBox2.DataSource=dataTable;
listBox2.Items.Clear();
listBox3.DataSource=dataTable;
listBox3.Items.Clear();
listBox4.DataSource=dataTable;
listBox4.Items.Clear();
// чистим грид
dataGrid1.DataSource=dataTable;
dataGrid1.Text="";
button2.Visible=true; // показываем кнопку «Заполнить»
button3.Visible=false; // прячем кнопку «Очистить»
}
```

**Шаг 8. Создаем процедуру сохранения таблицы в формате XML**

Для кнопки «Сохранить» пишем:

```
private void button6_Click(object sender, System.EventArgs e) {
FileStream fin_out; // задаем поток
try  {// если файл не существует
fin_out=new FileStream("dataSet.xml",FileMode.OpenOrCreate);
fin_out.Close();
        }
catch(IOException exc)  {
MessageBox.Show ("Невозможно открыть файл.","Ошибка");
return;}
StreamWriter xml_out;
try {
xml_out=new StreamWriter("dataSet.xml");
        }
```

```
catch(IOException exc) {
MessageBox.Show ("Невозможно открыть файл.","Ошибка");
return;}
// форматируем xml
XmlTextWriter xmlWriter=new XmlTextWriter(xml_out);
// способ форматирования
xmlWriter.Formatting=Formatting.Indented;
xmlWriter.Indentation=3;
DataSet dataSet=new DataSet();
DataTable dataTable=dataSet.Tables.Add("CardVISA");
dataTable.Columns.Add("Сумма",System.Type.GetType("System.String"));
dataTable.Columns.Add("Цель",System.Type.GetType("System.String"));
dataTable.Columns.Add("Фирма",System.Type.GetType("System.String"));
dataTable.Columns.Add("Дата",System.Type.GetType("System.String"));
dataTable.Columns.Add("ID",System.Type.GetType("System.Int64"));
object[] a1={"-17800","Salary","PMAT", "07-02-2006","1"};
object[] a2={"300","Связь","MTC", "08-02-2006","2"};
object[] a3={"300","Связь","MTC", "09-02-2006","3"};
dataTable.Rows.Add(a1);
dataTable.Rows.Add(a2);
dataTable.Rows.Add(a3);
dataSet.WriteXml(xmlWriter,XmlWriteMode.IgnoreSchema);
xmlWriter.Close();
MessageBox.Show("Файл записан в формате XML","Сохранение");
}
```

**Шаг 9. Создаем процедуру выхода из программы**

Для кнопки «Выход» пишем:

```
private void button1_Click(object sender, System.EventArgs e) {
Application.Exit();
}
```

### 3.2. Добавление-удаление записей

Задача темы: изучить методику и получить навыки в разработке приложений для манипулирования записями в базах данных формата XML.

Разработка: Приложение «CARD VISA» для просмотра, добавления, редактирования, удаления записей и математической обработки данных движения средств на пластиковой карточке.

Состав выполняемых функций:

- Файловые операции: открыть, сохранить, сохранить как, закрыть
- Алгоритмы: просмотр и прокрутка записей базы в текстовых полях, таблицах; просмотр отдельной записи; фильтрация и группмировка записей, подсчет сумм расходов по операциям и остатка средств на карточке
- Информационно-справочные: О программе

Наращивание функциональности не предусматривается.

Защита данных – от неквалифицированных действий пользователя.

В качестве входных данных используются файлы в формате XML. Выходные данные программы – графическое отображение результатов в виде текстовых полей и таблиц для SmartDevice; файлы в формате XML

#### Шаг 1. Разработка графического интерфейса

Для разработке используется одна форма со свойством ComboBox=False, tabControl с двумя закладками «База данных» и «Редактор».

На закладке «База данных» размещены надпись «Остаток» с текстовым полем (свойство RaedOnly=True), кнопка поиска «Найти» и таличный элемент DataGrid.

На закладке «Редактор» размещены надписи «Сумма», «Цель», «Фирма», «Дата» с соответствующими полями (для поля даты свойство ReadOnly=True); поле для вывода значения идентификационного номера записи со свойством ReadOnly=True; заголовочная надпись «День-Месяц-Год» с тремя комбобоксами (коллекции комбобоксов: день – от 01 до 31; месяц –

от 01 до 12; год  - на 10 лет вперед, например, от 07 до 17); кнопки «Фильтр ЦЕЛЬ», «Фильтр ФИРМА», «Показать все», «Добавить», «Удалить», «Заменить».

Примерный дизайн показан на рис. 24 и 25

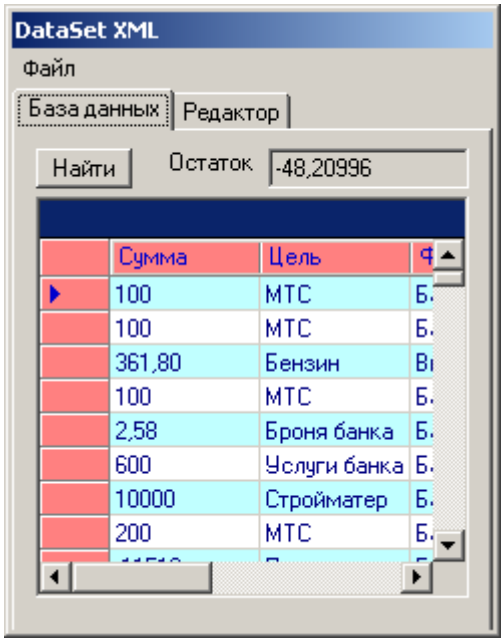


Рис.24

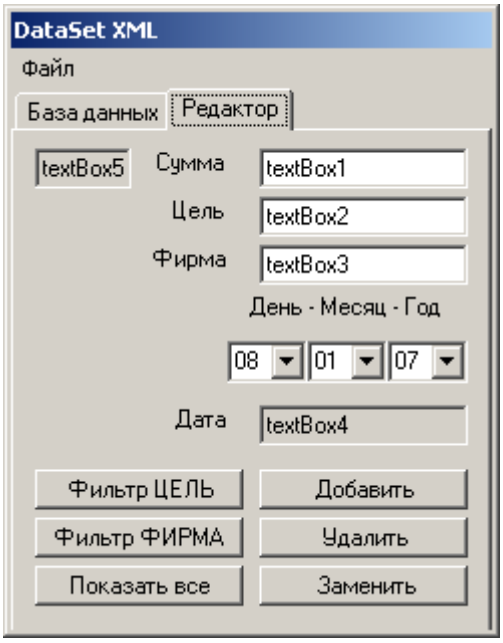


Рис.25

Невидимые элементы управления: mainMenu с опциями «Открыть», «Сохранить как» и «Выход»; openFileDialog и saveFileDialog. - для диалогов свойство Filter = Документы XML|*.xml

**Шаг 2. Добавляем библиотеки**

```
using System;
using System.Drawing;
using System.Collections;
using System.Windows.Forms;
using System.Data;
//
using System.IO;
using System.Xml;
```

**Шаг 3. Добавляем переменные в public class Form1 : System.Windows.Forms.Form**

```
// объявляем переменную имени файла
```

```
string file_name;
// глобальный датасет
DataSet dataSet,dataSet1;
//глобальный дататейбл
DataTable dataTable, dataTable1;
int my_grid=0;// статус грида, если 0, то грузим редактор, если 1, то - фильтр
string my_fuel, my_service;// переменные запроса с группировкой
// переменные полей
string my_summa,my_cell, my_firma,my_ID;
// переменная номера строки
System.Int64 my_Int64;
//переменная для удаления строки, чтобы не нарушать коллекцию foreach
DataRow remove_row;
DataRow newRow;
Int64 my_maxID=0;
// установка начального значения текущей даты
string my_day="01";
string my_mon="01";
string my_yea="01";
// фильтр
string my_filt, my_chan;
```

**Шаг 4. Дополняем public Form1()**

```
{
InitializeComponent();
file_name="dataSet.xml"; // тестовый файл загрузки по умолчанию
file_read(); // вызов функции чтения данных из файла
// устанавливаем системную дату для облегчения ввода дат пользователем
DateTime curDate = new DateTime();
curDate=DateTime.Now;
my_day=curDate.ToString("dd.MM.yyyy").Substring(0,2);
```

```
my_mon=curDate.ToString("dd.MM.yyyy").Substring(3,2);
my_yea=curDate.ToString("dd.MM.yyyy").Substring(8,2);
// грузим дату в комбобоксы
this.comboBox1.Text=my_day;
this.comboBox2.Text=my_mon;
this.comboBox3.Text=my_yea;
}
```

**Шаг 5. Создаем функцию file_read()**

Фунция открывает файл для чтения и грузит данные в Датагрид, затем рекурсивно вызывает функцию подсчета остатка средств на карточке

```
private void file_read() {
FileStream fin;
try {
fin=new FileStream(file_name,FileMode.Open);
fin.Close();
        }
catch(IOException exc) {
MessageBox.Show ("Невозможно открыть файл \n"+file_name,"Ошибка");
return;}
// Считываем значения из файла
XmlTextReader xml_in=new XmlTextReader(file_name);
dataSet=new DataSet();
dataTable=dataSet.Tables.Add("CardVISA");
// добавляем столбцы
dataTable.Columns.Add("Сумма",System.Type.GetType("System.String"));
dataTable.Columns.Add("Цель",System.Type.GetType("System.String"));
dataTable.Columns.Add("Фирма",System.Type.GetType("System.String"));
dataTable.Columns.Add("Дата",System.Type.GetType("System.String"));
dataTable.Columns.Add("ID",System.Type.GetType("System.Int64"));
dataSet.ReadXml(@xml_in);
```

```
xml_in.Close(); // закрываем входящий поток
// сначала сортируем строки по столбцу ID
DataView sortedView=new DataView(dataTable);
// обратная сортировка
sortedView.Sort="ID DESC";
// привязываем
dataGrid1.DataSource=sortedView;
balance(); // вызываем функцию пересчета остатка
}
```

**Шаг 6. Создаем функцию подсчета остатка средств на карточке balance()**

```
private void balance() {
float my_balance=0;
foreach (DataRow row in dataTable.Rows) {// цикл построчного пересчета
my_summa=(row["Сумма"].ToString());
my_balance=my_balance+float.Parse(my_summa); // нарастающий итог
        }
textBox6.Text=my_balance.ToString(); // показываем результат
}
```

**Шаг 7. Создаем процедуру открытия файла**

Если файл DataSet.xml отсутствует или ведет расчеты по другой карточке для опции главного меню «Открыть» пишем:

```
// читаем файл
private void menuItem2_Click(object sender, System.EventArgs e) {
if (openFileDialog1.ShowDialog()==DialogResult.OK) {
        file_name=openFileDialog1.FileName;
        file_read();
        }
}
```

**Шаг 8. Создаем процедуру сохранения файла**

Для опции главного меню «Сохранить как» пишем:

```
private void menuItem3_Click(object sender, System.EventArgs e) {
saveFileDialog1.FileName=file_name;
if (saveFileDialog1.ShowDialog()==DialogResult.OK) {
        file_name=saveFileDialog1.FileName;
        FileStream fin_out;
        try {
        fin_out=new FileStream(file_name,FileMode.OpenOrCreate);
        fin_out.Close();
                }
        catch(IOException exc) {
        MessageBox.Show ("Невозможно открыть файл.","Ошибка");
        return;}
        StreamWriter xml_out;
        try {
        xml_out=new StreamWriter(file_name);
                }
        catch(IOException exc) {
        MessageBox.Show ("Невозможно открыть файл.","Ошибка");
        return;}
// форматируем xml
        XmlTextWriter xmlWriter=new XmlTextWriter(xml_out);
        xmlWriter.Formatting=Formatting.Indented;
        xmlWriter.Indentation=3;
// принимаем изменения в dataSet !!!
        this.dataTable.AcceptChanges();
// выводим в xml
        dataSet.WriteXml(xmlWriter,XmlWriteMode.IgnoreSchema);
        xmlWriter.Close();
        MessageBox.Show("Файл записан в формате XML","Сохранение");
        }
```

```
}
```

**Шаг 9. Выходим из программы**

Для опции главного меню «Выход» пишем:

```
private void menuItem4_Click(object sender, System.EventArgs e) {
Application.Exit(); // что не очень хорошо, если изменения еще не записаны
}
```

**Шаг 10. Создаем функцию считывания данных из полей закладки «Редактор»**

```
private void appe_row() {
// заполняем текстбох даты
my_day=this.comboBox1.Text;
my_mon=this.comboBox2.Text;
my_yea=this.comboBox3.Text;
this.textBox4.Text=my_day+"-"+my_mon+"-"+my_yea;
// вычисляем длину строки
int my_len=textBox1.Text.Length;
if (my_len>=2) {
 // проверяем минус, если вводится отрицательное число
        if (textBox1.Text.Substring(0,1)=="-") {
        // проверяем вторую цифру
                if (!char.IsDigit(textBox1.Text[1])) {
                MessageBox.Show("Введите отрицательную сумму!","Ошибка");
                return;}
                }
        else
        // проверяем первую
        if (!char.IsDigit(textBox1.Text[0])) {
        MessageBox.Show("Введите сумму!","Ошибка");
        return;}
        // если длина больше 2-х
```

```
        else
        if (!char.IsDigit(textBox1.Text[0])) {
        MessageBox.Show("Введите сумму!","Ошибка");
        return;}
        }
}
```

**Шаг 11. Создаем процедуру добавления записи в базу**

Для кнопки «Добавить» закладки «Редактор» пишем:

```
private void button1_Click(object sender, System.EventArgs e) {
appe_row(); // считываем значение полей закладки «Редактор»
// вычисляем последнее значения
foreach (DataRow row2 in dataTable.Rows) {
string my_IDD=(row2["ID"].ToString());
my_Int64=Int64.Parse(my_IDD);
if (my_maxID<=my_Int64) {
        my_maxID=my_Int64+1;
                }
        }
this.textBox5.Text=my_maxID.ToString();
my_maxID=0; // обнуляем максимальный номер и вводим данные
newRow=dataTable.NewRow();
newRow["Сумма"]=this.textBox1.Text;
newRow["Цель"]=this.textBox2.Text;
newRow["Фирма"]=this.textBox3.Text;
newRow["Дата"]=this.textBox4.Text;
newRow["ID"]=this.textBox5.Text;
// добавляем запись
dataTable.Rows.Add(newRow);
this.tabControl1.SelectedIndex=0; // переключаемя за закладку «База данных»
balance(); // считаем остаток на карточке
```

```
}
```

**Шаг 12. Создаем процедуру удаления записи**

Для кнопки «Удалить» закладка «Редактор» пишем:

```
private void button2_Click(object sender, System.EventArgs e) {
if (this.textBox5.Text = = "0") {// если идентификационный номер строки 0
 MessageBox.Show("Строка уже удалена","Ошибка");
return;}
string remove_ID=textBox5.Text;
foreach (DataRow row in dataTable.Rows) {
my_ID=(row["ID"].ToString());
if (my_ID = = remove_ID) {
        remove_row=row;
                }
        }
dataTable.Rows.Remove(remove_row);
// обнуляем номер строки, чтобы избежать исключения о ненайденной строке
this.textBox5.Text="0";
this.tabControl1.SelectedIndex=0;
balance(); // пересчитываем остаток
}
```

**Шаг 13. Создаем процедуру замены записи при редактировании данных**

Для кнопки «Заменить» закладки «Редактор» пишем:

```
private void button3_Click(object sender, System.EventArgs e) {
// сначала удаляем строку
if (this.textBox5.Text=="0") {
MessageBox.Show("Строка уже удалена","Ошибка");
return;}
string remove_ID=textBox5.Text;
foreach (DataRow row in dataTable.Rows) {
        my_ID=(row["ID"].ToString());
```

```
        if (my_ID==remove_ID) {
                remove_row=row;
                }
        }
dataTable.Rows.Remove(remove_row);
// считываем значения
appe_row();
// вводим данные
newRow=dataTable.NewRow();
newRow["Сумма"]=this.textBox1.Text;
newRow["Цель"]=this.textBox2.Text;
newRow["Фирма"]=this.textBox3.Text;
newRow["Дата"]=this.textBox4.Text;
// переписываем номер
newRow["ID"]=remove_ID;
// добавляем запись
dataTable.Rows.Add(newRow);
// обнуляем номер строки, чтобы избежать исключения о ненайденной строке
this.textBox5.Text="0";
this.tabControl1.SelectedIndex=0;
balance();
}
```

**Шаг 14. Создаем функцию фильтра целевых расходов**

```
private void cell_sum() {
float my_balance=0;
foreach (DataRow row in dataTable.Rows) {
my_summa=(row["Сумма"].ToString());
my_cell=(row[my_chan].ToString());
if (my_filt==my_cell) {
         my_balance=my_balance+float.Parse(my_summa);
```

```
            }
        }
textBox6.Text=my_balance.ToString();
}
```

**Шаг 15. Создаем процедуру фильтрации целевых расходов**

Для кнопки «Фильтр цели» пишем:

```
private void button4_Click(object sender, System.EventArgs e) {
// внимательно: в кавычки берется апостроф
dataTable.DefaultView.RowFilter="Цель='"+this.textBox2.Text+"'";
dataGrid1.DataSource=dataTable;
this.tabControl1.SelectedIndex=0;
my_filt=this.textBox2.Text;
my_chan="Цель";
cell_sum(); // считаем результат
}
```

**Шаг 16. Создаем процедуру фильтрации расходов по фирмам**

Для кнопки «Фильтр фирмы» пишем:

```
private void button6_Click(object sender, System.EventArgs e) {
dataTable.DefaultView.RowFilter="Фирма='"+this.textBox3.Text+"'";
dataGrid1.DataSource=dataTable;
this.tabControl1.SelectedIndex=0;
my_filt=this.textBox3.Text;
my_chan="Фирма";
cell_sum();
}
```

**Шаг 17. Создаем процедуру удаления фильтра**

Для кнопки «Показать все» пишем:

```
private void button5_Click(object sender, System.EventArgs e) {
DataView sortedView=new DataView(dataTable);
// обратная сортировка
```

```
sortedView.Sort="ID DESC";
dataGrid1.DataSource=sortedView;
this.tabControl1.SelectedIndex=0;
balance();
}
```

**Шаг 18. Создаем процедуру запроса с группировкой данных**

Для кнопки «Найти» закладки «База данных» пишем:

```
private void My_Query(){
// выборка целей платежей -  создаем структуру таблицы DataSet1
dataSet1=new DataSet();
dataTable1=dataSet1.Tables.Add("Платежи");
// добавляем столбцы
dataTable1.Columns.Add("Цель",System.Type.GetType("System.String"));
dataTable1.Columns.Add("Сумма",System.Type.GetType("System.String"));
// добавляем первую строку для шаблона
int my_test=0;
string my_sum;
float my_summa=0;
foreach (DataRow row in dataTable.Rows) { // перебираем все записи
my_fuel=(row["Цель"].ToString());
my_test=0;
DataRow[] rowCell=dataTable.Select("Цель='"+my_fuel+"'");
// вложенные циклы выборки записей по условию
        foreach (DataRow rowTable in rowCell) {
        my_summa=my_summa+float.Parse(rowTable["Сумма"].ToString());
        }
        foreach (DataRow row1 in dataTable1.Rows){
        my_service=(row1["Цель"].ToString());
                if (my_fuel==my_service)
                my_test=1;
```

```
        }
if (my_test==0){
        my_sum=my_summa.ToString();
        object[] a1={my_fuel,my_sum};
        dataTable1.Rows.Add(a1);
                }
my_summa=0;
        }
//привязываем
dataGrid1.TableStyles.Clear();
// стиль грид делаем программно
DataGridTableStyle ts= new DataGridTableStyle();
ts.MappingName="Платежи";
DataGridTextBoxColumn cs=new DataGridTextBoxColumn();// первый столбец
cs.Width=120;
cs.MappingName="Цель";
cs.HeaderText="Выберите платеж";
cs.NullText="";
ts.GridColumnStyles.Add(cs);
cs = new DataGridTextBoxColumn();// второй столбец
cs.Width=70;
cs.MappingName="Сумма";
cs.HeaderText="ИТОГО";
ts.GridColumnStyles.Add(cs);
this.dataGrid1.TableStyles.Add(ts);
DataView sortedView1=new DataView(dataTable1);
sortedView1.Sort="Цель";
dataGrid1.DataSource=sortedView1;
my_grid=1;
}
```

**Шаг 19. Создаем процедуру для кнопки «Найти»**

```
private void button7_Click(object sender, System.EventArgs e) {
if (this.dataGrid1.Visible==false) {// выводим сообщение с картинкой
MessageBox.Show("Откройте или создайте файл данных!",
        "Ошибка", MessageBoxButtons.OK,
        MessageBoxIcon.Asterisk,MessageBoxDefaultButton.Button1 );
        return;}
My_Query();
}
```

**Шаг 20. Создаем процедуру выборки данных из базы по условию**

Для клика записи ДатаГрид пишем:

```
private void dataGrid1_Click(object sender, System.EventArgs e) {
// номер «кликнутой» строки
int N_row=dataGrid1.CurrentCell.RowNumber;
// номер сторбца
int N_col=dataGrid1.CurrentCell.ColumnNumber;
if (my_grid==1) {
my_fuel=dataGrid1[N_row,0].ToString();
DataView sortedView=new DataView(dataTable);
sortedView.Sort="ID DESC";
sortedView.RowFilter="Цель='"+my_fuel+"'";
dataGrid1.TableStyles.Clear();
dataGrid1.TableStyles.Add(dataGridTableStyle1);
dataGrid1.DataSource=sortedView;
my_grid=0;
return;
        }
// вывод значения ячейки
my_ID=dataGrid1[N_row,0].ToString()+"\n"+
```

```
        dataGrid1[N_row,1].ToString()+"\n"+
        dataGrid1[N_row,2].ToString()+"\n"+
        dataGrid1[N_row,3].ToString()+"\n"+
        dataGrid1[N_row,4].ToString();
textBox1.Text=dataGrid1[N_row,0].ToString();
textBox2.Text=dataGrid1[N_row,1].ToString();
textBox3.Text=dataGrid1[N_row,2].ToString();
textBox4.Text=dataGrid1[N_row,3].ToString();
textBox5.Text=dataGrid1[N_row,4].ToString();
string mes=N_col.ToString();;
if (N_col==0) {mes="Сумма";}
        if (N_col==1) {mes="Цель";}
                if (N_col==2) {mes="Фирма";}
                        if (N_col==3) {mes="Дата";}
                                if (N_col==4) {mes="ID";}
if (MessageBox.Show(my_ID+"\n Редактировать?",mes,
        MessageBoxButtons.YesNo,
        MessageBoxIcon.Question,MessageBoxDefaultButton.Button2) =
        = DialogResult.Yes)
        this.tabControl1.SelectedIndex=1;
}
```

### 3.3. Проверка вводимых данных

Задача темы: изучить методику и получить навыки разработке средств контроля неквалифицированных действий пользователей

Разработка: Приложение «Валидатор» для контроля вводимых в текстовое поле сиволов – цифра или буква.

Состав выполняемых функций:

- Файловые операции: нет
- Алгоритмы: проверка вводимых символов с помощью функции, класса и библиотеки
- Информационно-справочные: нет

Наращивание функциональности не предусматривается.

Защита данных – от неквалифицированных действий пользователя.

В качестве входных данных используются символы, вводимые в контрольное текстовое поля с клавиатуры. Выходные данные программы – графическое отображение результатов в виде текстовых сообщений для SmartDevice

#### Шаг 1. Создание библиотеки контроля вводимых символов

Для создания библиотек используется специальная функциональность Visual Studio – создаем проект ClassLibrary. В нашем примере создается проект библиотеки с именем DigitValidateDLL. В проекте создается всего один класс public class Class1: System.ComponentModel.Component, в который записываем:

```
{// переменная (string DigitValidate) возвращаемого значения
public string ReturnString(string DigitValidate) {
// вычисляем длину строки
int L_str=DigitValidate.Length;
// цикл считывания символов из строки
```

```
int i;
string V_str="";
string T_str="";
for(i=0;i<L_str;i++) {
        V_str=V_str+i.ToString();
        T_str=DigitValidate.Substring(0,i+1);
// проверяем каждый символ на цифру
        if (!char.IsDigit(DigitValidate[i])) {
        return("Ошибка");}
                }
return ("Правильно"); // возвращаем значение в вызывающую программу
        }
}
```

Компилируем библиотеку. Полученный файл DigitValidateDLL.dll, следует разместить в каталоге bin\Debug нашего приложения «Валидатор» - рис.26

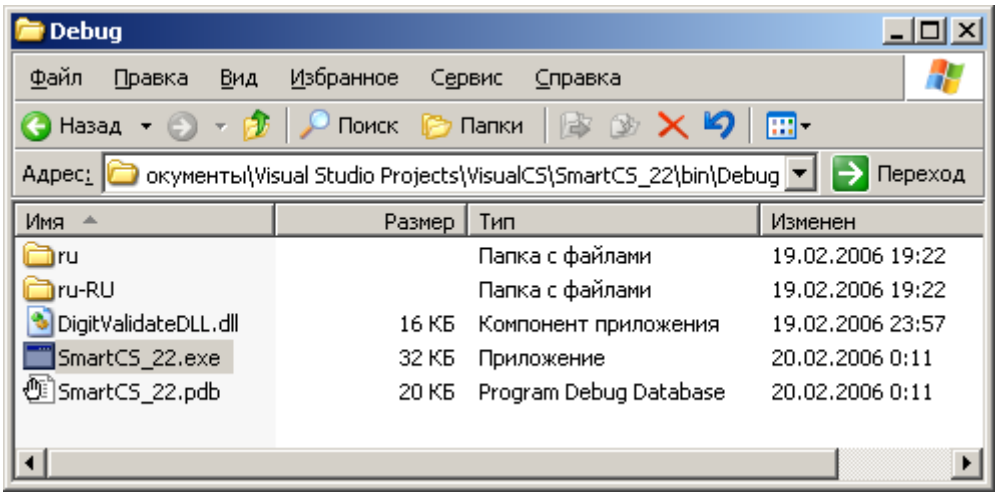


Рис.26 Размещение библиотеки (компонента)
в каталоге запуска приложения

**Шаг 2. Создаем графический интерфейс приложения «Валидатор»**

Приложение включает одну форму с двумя надписями «Введите что-нибудь» и «Проверьте наличие цифр в строке»; тектовое поле ввода и при

кнопки «Функция», «Класс», «Библиотека». Примерный дизай приведен на рис. 27

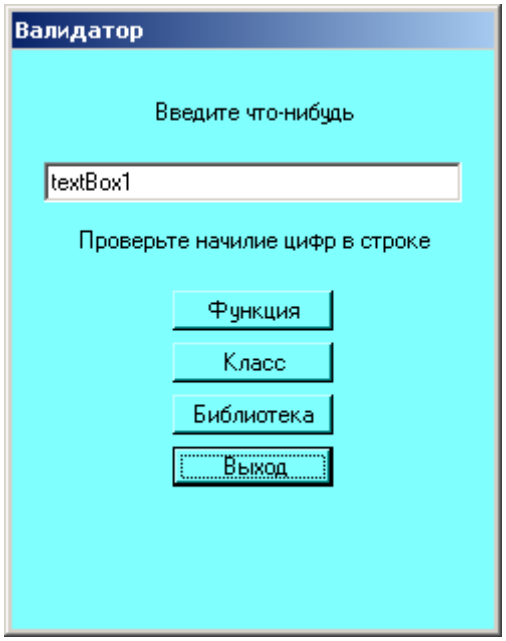


Рис. 27

**Шаг 3. Добавляем собственную библиотеку в проект**

```
using System;
using System.Drawing;
using System.Collections;
using System.Windows.Forms;
using System.Data;
// добавляем собственную библиотеку (Add Reference -> Project)
using DigitValidateDLL;
```

**Шаг 4. Создаем процедуру вызова библиотеки**

Для кнопки «Библиотека» пишем:

```
private void button3_Click(object sender, System.EventArgs e) {
// переменная DigitValidate уже определена в подключенной библиотеке
DigitValidate =this.textBox1.Text;
// создаем ссылочную переменную DVL для точки входа Class1
DigitValidateDLL.Class1 DVL=new DigitValidateDLL.Class1();
// задаем исходное значение для одноименной переменной библиотеки
if (DVL.ReturnString(DigitValidate) = = "Ошибка") {
```

```
MessageBox.Show("Введите цифру","Библиотека проверки цифр в строке");
}
else  {
MessageBox.Show("Все символы в строке - цифры!",
        "Библиотека проверки цифр в строке");}
}
```

**Шаг 5. Создаем процедуру проверки символов**

Для кнопки «Функция» пишем:

```
private void button1_Click(object sender, System.EventArgs e) {
DigitValidate =this.textBox1.Text;
// вычисляем длину строки
int L_str=DigitValidate.Length;
// цикл считывания символов из строки
int i;
string V_str="";
string T_str="";
for(i=0;i<L_str;i++) {
V_str=V_str+i.ToString();
T_str=DigitValidate.Substring(0,i+1);
// проверяем каждый символ на цифру
        if (!char.IsDigit(DigitValidate[i])) {
        MessageBox.Show(T_str+"\n"+
        i.ToString()+"-й символ не цифра!","Ошибка");
        return;}
        }
MessageBox.Show("Длина строки -"+L_str.ToString()+" знаков,\n"+
        "Номера символов в строке "+V_str+"\n"+
        T_str+"\n"+
        "Все символы в строке - цифры!","Функция проверки цифр в строке");
}
```

**Шаг 6. Создаем класс ClassDigitValidate.cs проверки символов в строке**

Хотя код в значительной степени совпадает с кодом процедуры кнопки и библиотеки, код приводим полностью:

```
using System;
// добавляем для вывода MessageBox
using System.Windows.Forms;
namespace SmartCS_22 // пространство имен
{
public class ClassDigitValidate
    { // меняем public ClassDigitValidate() на
    public static void ClassDigit()
        {//Передаем значение из класса Form1 в переменную
        string DigitValidate=Form1.DigitValidate;
        // теперь копируем функцию из класса Form1
        // вычисляем длину строки
        int L_str=DigitValidate.Length;
        // цикл считывания символов из строки
        int i;
        string V_str="";
        string T_str="";
        for(i=0;i<L_str;i++) {
            V_str=V_str+i.ToString();
            T_str=DigitValidate.Substring(0,i+1);
        // проверяем каждый символ на цифру
            if (!char.IsDigit(DigitValidate[i])) {
                MessageBox.Show(T_str+"\n"+
                i.ToString()+"-й символ не цифра!","Ошибка");
                return;}
```

```
                }
        MessageBox.Show("Длина строки -"+L_str.ToString()+" знаков,\n"+
        "Номера символов в строке "+V_str+"\n"+
        T_str+"\n"+
        "Все символы в строке - цифры!","Класс проверки цифр в строке");
                }
        }
}
```

**Шаг 7. Создаем процедуру вызова класса**

Для кнопки «Класс» пишем:

```
private void button2_Click(object sender, System.EventArgs e) {
DigitValidate =this.textBox1.Text;
// при правильном коде класса будет подсказка вызова
ClassDigitValidate.ClassDigit();
}
```

**Шаг 8. Выход из программы**

```
private void button4_Click_1(object sender, System.EventArgs e) {
Application.Exit();
}
```

## 3.4. Запросы

Задача темы: изучить методику и получить навыки в разработке методов выборки данных из таблиц

Разработка: Приложение «Запросы» для выборки данных из таблиц движения средств на банковских пластиковых карточках

Состав выполняемых функций:

- Файловые операции: открыть
- Алгоритмы: выборка данных по целям платежей и фирмам с группировкой
- Информационно-справочные: О программе

Наращивание функциональности не предусматривается.

Защита данных – запрет доступа пользователя к данным.

В качестве входных данных используются файлы с данными о движении средств на банковских карточках. Выходные данные программы – графическое отображение результатов в виде таблиц DataGrid для SmartDevice

### Шаг 1. Разработка графического интерфейса

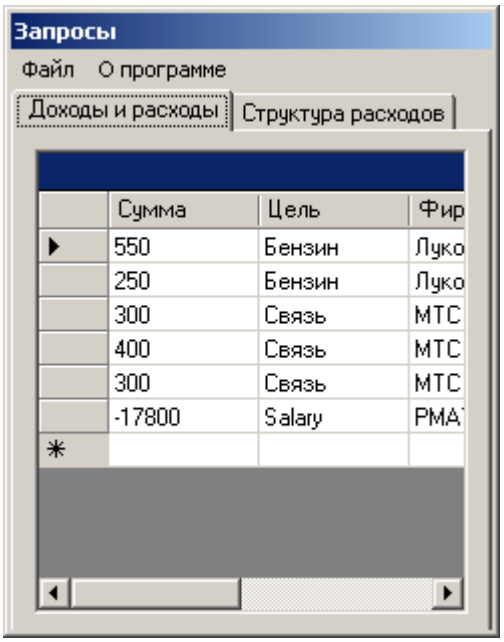


Рис.28

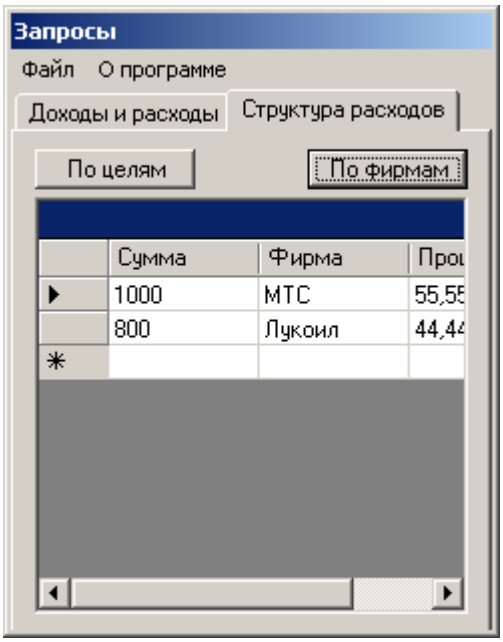


Рис. 29

Предусматривается одна форма с TabControl из двух закладок «Доходы и расходы» с единственным элементом DataGrid и «Структура расходов» с DataGrid и двумя кнопками запуска запросов «По целям» и «По фирмам».

Из невидимых элементов – главное меню с опциями «Открыть», «Выход»; openFileDialog со свойством Filter = Файлы XML (*.xml)|*.xml – примерный дизайн на рис. 28 и 29.

**Шаг 2. Добавляем библиотеки**

```
using System;
using System.Drawing;
using System.Collections;
using System.Windows.Forms;
using System.Data;
//добавляем
using System.IO;
using System.Xml;
```

**Шаг 3. Добавляем переменные в public class Form1 : System.Windows.Forms.Form**

```
string file_name="dataSet.xml";
// глобальный датасет
DataSet dataSet;
DataSet dataSetCell;
//глобальный дататейбл
DataTable dataTable;
DataTable dataCell;
DataTable dataFirma;
DataRow newRow;
```

**Шаг 4. В public Form1() вызываем функцию чтения данных file_read();**

```
{
InitializeComponent();
file_read();
```

```
}
```

**Шаг 5. Создаем функцию чтения данных из XML-файла**

```
private void file_read() {
FileStream fin;
try {
fin=new FileStream(file_name,FileMode.Open);
fin.Close();
        }
catch(IOException exc) {
MessageBox.Show ("Невозможно открыть файл \n"+file_name,"Ошибка");
return;}
// Считываем значения из файла
XmlTextReader xml_in=new XmlTextReader(file_name);
dataSet=new DataSet();
dataTable=dataSet.Tables.Add("CardVISA");
dataSetCell=new DataSet();
dataCell=dataSetCell.Tables.Add("CardCell");
dataFirma=dataSet.Tables.Add("CardFirma");
dataTable.Columns.Add("Сумма",System.Type.GetType("System.String"));
dataTable.Columns.Add("Цель",System.Type.GetType("System.String"));
dataTable.Columns.Add("Фирма",System.Type.GetType("System.String"));
dataTable.Columns.Add("Дата",System.Type.GetType("System.String"));
dataTable.Columns.Add("ID",System.Type.GetType("System.Int64"));
dataTable.Columns["Сумма"].ReadOnly=true;
dataTable.Columns["Цель"].ReadOnly=true;
dataTable.Columns["Фирма"].ReadOnly=true;
dataTable.Columns["Дата"].ReadOnly=true;
dataTable.Columns["ID"].ReadOnly=true;
dataSet.ReadXml(@xml_in);
xml_in.Close();
```

```
DataView sortedView=new DataView(dataTable);
sortedView.Sort="ID DESC";
dataGrid1.DataSource=sortedView;
// добавляем столбцы в dataCell
dataCell.Columns.Add("Процент",System.Type.GetType("System.String"));
dataCell.Columns.Add("Цель",System.Type.GetType("System.String"));
dataCell.Columns.Add("Сумма",System.Type.GetType("System.String"));
// добавляем столбцы в dataFirma
dataFirma.Columns.Add("Сумма",System.Type.GetType("System.String"));
dataFirma.Columns.Add("Фирма",System.Type.GetType("System.String"));
dataFirma.Columns.Add("Процент",System.Type.GetType("System.String"));
}
```

**Шаг 6. Создаем диалог открытия файла**

Для опции главного меню «Открыть» пишем:

```
private void menuItem2_Click(object sender, System.EventArgs e) {
if (openFileDialog1.ShowDialog()==DialogResult.OK) {
file_name=openFileDialog1.FileName;
file_read();
        }
}
```

**Шаг 7. Создаем запрос- выборку данных «По фирмам»**

Для кнопки «По фирмам» пишем:

```
// обновляем запрос по фирмам
private void button2_Click_1(object sender, System.EventArgs e) {
// считывание строк из dataTable и добавление в dataCell
string my_cell;
foreach (DataRow row in dataTable.Rows) {
newRow=dataFirma.NewRow();
my_cell=(row["Фирма"].ToString());
newRow["Фирма"]=my_cell;
```

```
// ищем строку - создаем массив строк
DataRow[] rowCell=dataFirma.Select("Фирма='"+my_cell+"'");
//Если длина (не size, а Length!) массива 0
if (rowCell.Length==0) {
this.dataFirma.Rows.Add(newRow);
                }
        }
// А теперь считаем суммы
float  my_summa=0;
foreach (DataRow row in dataFirma.Rows) {
my_cell=(row["Фирма"].ToString());
DataRow[] rowCell=dataTable.Select("Фирма='"+my_cell+"'");
foreach (DataRow rowTable in rowCell) {
my_summa=my_summa+float.Parse(rowTable["Сумма"].ToString());
                }
if (my_summa>=0) {
row["Сумма"]=my_summa.ToString();
                }
else {row["Сумма"]="0";
                }
my_summa=0;
        }
// А теперь считаем итого
foreach (DataRow row in dataFirma.Rows) {
my_summa=my_summa+float.Parse(row["Сумма"].ToString());
        }
float my_total=my_summa;
// заполняем структуру
double my_procent;
string my_format;
```

```
foreach (DataRow row in dataFirma.Rows) {
my_procent=double.Parse(row["Сумма"].ToString())*100/my_total;
// добавляем 0, если меньше 10
if (my_procent>=10) {
my_format=my_procent.ToString();
                }
else {my_format="0"+my_procent.ToString();
                }
// обрезаем строку до 5 символов
if (my_format.Length>=5) {
row["Процент"]=(my_format.Substring(0,5));
                }
else {row["Процент"]=my_format;
                }
        }
// А теперь считаем 100 %
my_summa=0;
foreach (DataRow row in dataFirma.Rows) {
my_summa=my_summa+float.Parse(row["процент"].ToString());
                }
// сортируем по цели
DataView sortedCell=new DataView(dataFirma);
sortedCell.Sort="Процент DESC";
// фильтруем пустые строки
sortedCell.RowFilter="Сумма<>'0'";
dataGrid2.DataSource=sortedCell;
}
```

**Шаг 8. Создаем запрос-выборку «По целям»**

Для кнопки «По целям пишем»:

```
private void button1_Click(object sender, System.EventArgs e) {
```

```
// считывание строк из dataTable и добавление в dataCell
string my_cell;
foreach (DataRow row in dataTable.Rows) {
newRow=dataCell.NewRow();
my_cell=(row["Цель"].ToString());
newRow["Цель"]=my_cell;
// ищем строку - создаем массив строк
DataRow[] rowCell=dataCell.Select("Цель='"+my_cell+"'");
//Если длина (не size, а Length!) массива 0
if (rowCell.Length==0) {
this.dataCell.Rows.Add(newRow);
                }
        }
// А теперь считаем суммы
float  my_summa=0;
foreach (DataRow row in dataCell.Rows) {
my_cell=(row["Цель"].ToString());
DataRow[] rowCell=dataTable.Select("Цель='"+my_cell+"'");
foreach (DataRow rowTable in rowCell) {
my_summa=my_summa+float.Parse(rowTable["Сумма"].ToString());
                }
if (my_summa>=0) {
row["Сумма"]=my_summa.ToString();
                }
else {row["Сумма"]="0";
                }
my_summa=0;
        }
// А теперь считаем итого
foreach (DataRow row in dataCell.Rows) {
```

```
my_summa=my_summa+float.Parse(row["Сумма"].ToString());
        }
float my_total=my_summa;
// заполняем структуру
double my_procent;
string my_format;
foreach (DataRow row in dataCell.Rows) {
my_procent=double.Parse(row["Сумма"].ToString())*100/my_total;
// добавляем 0, если меньше 10
if (my_procent>=10) {
my_format=my_procent.ToString();
                }
else {my_format="0"+my_procent.ToString();
                }
// обрезаем строку до 5 символов
if (my_format.Length>=5) {
row["Процент"]=(my_format.Substring(0,5));
                }
else {row["Процент"]=my_format;
                }
        }
// А теперь считаем 100 %
my_summa=0;
foreach (DataRow row in dataCell.Rows) {
my_summa=my_summa+float.Parse(row["процент"].ToString());
                }
// сортируем по цели
DataView sortedCell=new DataView(dataCell);
sortedCell.Sort="Процент DESC";
// фильтруем пустые строки
```

```
sortedCell.RowFilter="Сумма<>'0'";
dataGrid2.DataSource=sortedCell;
}
```

**Шаг 9. Создаем процедуру «О программе»**

Для опции главного меню «О программе» пишем:

```
private void menuItem5_Click(object sender, System.EventArgs e) {
MessageBox.Show("Программа запросов с группировкой данных\n"+
    "по целям платежей и фирмам-поставщикам товаров и услуг\n"+
    "(C)Родигин Л.А., март 2006,\n"+
    "Платформа C#.NET MS VisualStudio 2003","О программе");
}
```

**Шаг 10. Выход из приложения**

Для опции главного меню «Выход» пишем:

```
private void menuItem4_Click(object sender, System.EventArgs e) {
Application.Exit();
}
```

## 3.5. Оформление интерфейса

А теперь сделаем небольшую передышку и поговорим об оформительских средствах приложений.

Задача темы: изучить методику и получить навыки использования оформительских средств в разработках

Разработка: Приложение «Как я живу?» для оценки распределения своего личного времени

Состав выполняемых функций:

- Файловые операции: нет
- Алгоритмы: вычисление структуры распределения личного времени по 10 позициям
- Информационно-справочные: О программе

Наращивание функциональности не предусматривается.

Защита данных – только от неквалифицированных действий пользователя.

В качестве входных данных используются программные данные с начальной структурой затрат личного времени. Выходные данные программы – графическое отображение результатов в виде гистограммы структуры данных для SmartDevice

### Шаг 1. Разработка графического интерфейса

Используются две формы – одна для ввода данных, другая – для вывода. Вторая форма (добавьте в проект) предназначена для вывода диаграммы со структурой распределения личного времени и не содержит никаких элементов, так как оформляется программно. Единственное предусмотренное свойство формы Text = Вот так и живу!

Первая форма содержит 10 надписей: Работа, Учеба, Культурный отдых, Спорт, Шопинг, Хобби, Безделье, Любовь, Еда, Сон; кнопку «Очистить» и 10 полей, со следующими свойствами – таблица 1.

Примерный дизайн формы показан на рис.30

Из невидимых элементов – главное меню с опциями «Файл» и «Диаграмма». В списке опций «Файл» можно предусмотреть «Открыть», «Сохранить как» и «Выход» - процедуры уже рассматривались выше и могут быть созданы самостоятельно. Только не забудьте добавить элементы диалогов открытия и записи файлов.

Таблица 1.

Свойства полей формы (Form1) «Как я живу?»

| Свойство BackColor | Свойство Text |
|---|---|
| Lime | 18 |
| 255; 192; 128 | 16 |
| 192; 0; 0 | 14 |
| 192; 0; 0 | 12 |
| Blue | 10 |
| 128; 255; 255 | 8 |
| 128; 255; 128 | 6 |
| Yellow | 4 |
| 255; 128; 0 | 2 |
| Red | 10 |

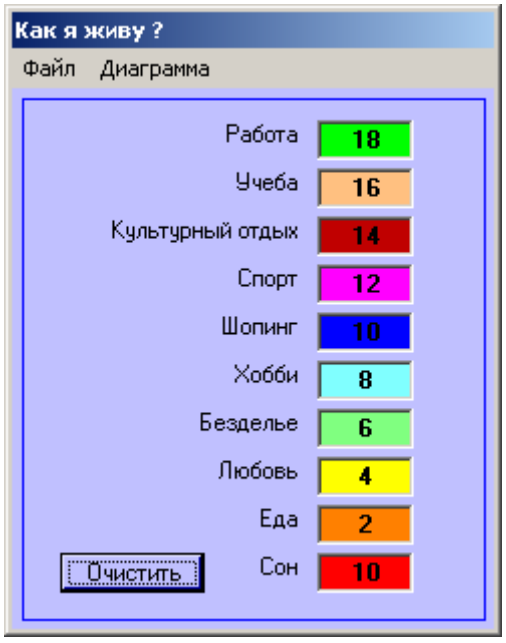

Рис.30

**Шаг 2. Добавляем библиотеки**

Для Form1:

```
using System;
using System.Drawing;
using System.Collections;
using System.Windows.Forms;
using System.Data;
// добавляем собственную библиотеку (Add Reference -> Project)
using DigitValidateDLL;
using System.ComponentModel; // для графики
/*добавляем, если есть намерение самостоятельно разработать процедуры открытия, загрузки и записи исходных данных файлов диаграмм
*/
using System.IO;
using System.Xml;
```

Для Form2:

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
```

**Шаг 3. Добавляем переменные public class Form1 : System.Windows.Forms.Form**

```
// объявляем глобальную переменную для использования в обоих классах
public static string DigitValidate="0";
// добавляем массив переменных для данных диаграммы
public static int[] My_Life=new int[10];
```

**Шаг 4. Создаем функцию проверки цифровых значений всех техстбоксов**

```
void ValidaTextBox() {
foreach (Control NT in this.Controls)
if (NT is TextBox)
        if (NT.Text=="") {// проверка пустых значений
        NT.Text="0";
        }
foreach (Control NT in this.Controls)
if (NT is TextBox) {// проверка цифр текстбокса
DigitValidate =NT.Text;
// подключаем библиотеку
DigitValidateDLL.Class1 DVL=new DigitValidateDLL.Class1();
if (DVL.ReturnString(DigitValidate)=="Ошибка") {
        NT.Text="0";
                }
        }
this.textBox10.Text="0"; // обнуляем сон
// ограничение 50 % для каждого текстбокса
foreach (Control NT in this.Controls)
if (NT is TextBox) {
if (int.Parse(NT.Text)>=50) {
NT.Text="50";
        }
}
// ограничение 100 % для суммы текстбоксов
int maxNT=0;
foreach (Control NT in this.Controls)
if (NT is TextBox) {
if (maxNT+int.Parse(NT.Text)>=100) {
 NT.Text=((100-maxNT).ToString());
 maxNT=100;
```

```
        }
else {
 maxNT=maxNT+int.Parse(NT.Text);
        }
}
// Вывчисляем сон, если сумма меньше 100
maxNT=0;
foreach (Control NT in this.Controls)
if (NT is TextBox) {
maxNT=maxNT+int.Parse(NT.Text);
        }
if (maxNT<=100) {
this.textBox10.Text=(100-maxNT).ToString();
        }
}
```

**Шаг 5. Создаем процедуры вызова функции проверки при кликании каждого из техтбоксов**

```
private void textBox1_TextChanged_1(object sender, System.EventArgs e) {
ValidaTextBox();
}
private void textBox2_TextChanged_2(object sender, System.EventArgs e) {
ValidaTextBox();
}
private void textBox3_TextChanged(object sender, System.EventArgs e) {
ValidaTextBox();
}
private void textBox4_TextChanged(object sender, System.EventArgs e) {
ValidaTextBox();
}
private void textBox5_TextChanged(object sender, System.EventArgs e) {
```

```
ValidaTextBox();
}
private void textBox6_TextChanged(object sender, System.EventArgs e) {
ValidaTextBox();
}
private void textBox7_TextChanged(object sender, System.EventArgs e) {
ValidaTextBox();
}
private void textBox8_TextChanged(object sender, System.EventArgs e) {
ValidaTextBox();
}
private void textBox9_TextChanged(object sender, System.EventArgs e) {
ValidaTextBox();
}
private void textBox10_TextChanged(object sender, System.EventArgs e) {
// кроме десятого, так как оно результирующее
}
```

**Шаг 6. Офрмляем дизайн Form1**

Для того, чтобы программно раскрасить форму нужно в свойствах событий Form1 Paint написать имя вызываемого метода – OnPaint, для которого пишем процедуру:

```
private void OnPaint (object sender, System.Windows.Forms.PaintEventArgs e) {
this.CreateGraphics();// создаем графический объект
Graphics g = e.Graphics;
SolidBrush brush = new SolidBrush(Color.Green);
Pen pen = new Pen(Color.Blue); // красим в голубой цвет
g.DrawRectangle(pen,5,5,ClientRectangle.Width - 10,
    ClientRectangle.Height - 10); // рисуем рамочку формы
}
```

**Шаг 7. Создаем диаграмму в Form2**

Также как и в Form1, загрузке графики предшествует определение имени вызываемого события, кстати, тоже OnPaint:

```
private void OnPaint(object sender, System.Windows.Forms.PaintEventArgs e) {
// считываем значения
this.CreateGraphics();
Graphics g = e.Graphics;
SolidBrush brush = new SolidBrush(Color.Red);
SolidBrush brush1 = new SolidBrush(Color.Orange);
SolidBrush brush2 = new SolidBrush(Color.Yellow);
SolidBrush brush3 = new SolidBrush(Color.Green);
SolidBrush brush4 = new SolidBrush(Color.Blue);
SolidBrush brush5 = new SolidBrush(Color.DarkBlue);
SolidBrush brush6 = new SolidBrush(Color.DarkViolet);
SolidBrush brush7 = new SolidBrush(Color.DarkRed);
SolidBrush brush8 = new SolidBrush(Color.DarkOrange);
SolidBrush brush9 = new SolidBrush(Color.Olive);
// вычисляем высоту графика и 10 на нижнюю границу
int My_High=ClientRectangle.Height-10;
Pen pen = new Pen(Color.Blue);
Font font = new Font("Arial",10,0);
g.DrawString(Form1.My_Life[0].ToString()+" - сон",font,brush,130,190);
g.DrawString(Form1.My_Life[1].ToString()+" - еда",font,brush1,130,170);
g.DrawString(Form1.My_Life[2].ToString()+" - любовь",font,brush2,130,150);
g.DrawString(Form1.My_Life[3].ToString()+" - безделье",font,brush3,130,130);
g.DrawString(Form1.My_Life[4].ToString()+" - хобби",font,brush4,130,110);
g.DrawString(Form1.My_Life[5].ToString()+" - шопинг",font,brush5,130,90);
g.DrawString(Form1.My_Life[6].ToString()+" - спорт",font,brush6,130,70);
g.DrawString(Form1.My_Life[7].ToString()+" - культпросвет", font, brush7,
        130,50);
g.DrawString(Form1.My_Life[8].ToString()+" - учеба",font,brush8,130,30);
```

```
g.DrawString(Form1.My_Life[9].ToString()+" - работа",font,brush9,130,10);
Font My_font = new Font("Arial",15,0);
g.DrawString("100 % !",My_font,brush7,140,220);
// Рамка
g.DrawRectangle(pen,5,5,ClientRectangle.Width/2, ClientRectangle.Height-10);
g.FillRectangle(brush,5,5,ClientRectangle.Width/2, ClientRectangle.Height-10);
g.DrawRectangle(pen,5,5,ClientRectangle.Width/2, ClientRectangle.Height - 10-
        Form1.My_Life[0]*My_High/100);
g.FillRectangle(brush1,5,5,ClientRectangle.Width/2, ClientRectangle.Height - 10-
        Form1.My_Life[0]*My_High/100);
// 2 Нижняя полоса
g.DrawRectangle(pen,5,5,ClientRectangle.Width/2, ClientRectangle.Height - 10-
        (Form1.My_Life[0]+Form1.My_Life[1])*My_High/100);
g.FillRectangle(brush2,5,5,ClientRectangle.Width/2, ClientRectangle.Height -10-
        (Form1.My_Life[0]+Form1.My_Life[1])*My_High/100);
// 3 Нижняя полоса
g.DrawRectangle(pen,5,5,ClientRectangle.Width/2, ClientRectangle.Height -10-
(Form1.My_Life[0]+Form1.My_Life[1]+Form1.My_Life[2])*My_High/100);
g.FillRectangle(brush3,5,5,ClientRectangle.Width/2, ClientRectangle.Height -10-
(Form1.My_Life[0]+Form1.My_Life[1]+Form1.My_Life[2])*My_High/100);
// 4 Нижняя полоса
g.DrawRectangle(pen,5,5,ClientRectangle.Width/2, ClientRectangle.Height - 10-
        (Form1.My_Life[0]+Form1.My_Life[1]+Form1.My_Life[2]+
        Form1.My_Life[3])*My_High/100);
g.FillRectangle(brush4,5,5,ClientRectangle.Width/2, ClientRectangle.Height -10-
        (Form1.My_Life[0]+Form1.My_Life[1]+Form1.My_Life[2]+
        Form1.My_Life[3])*My_High/100);
// 5 Нижняя полоса
g.DrawRectangle(pen,5,5,ClientRectangle.Width/2, ClientRectangle.Height - 10-
        (Form1.My_Life[0]+Form1.My_Life[1]+Form1.My_Life[2]+
```

```
        Form1.My_Life[3]+Form1.My_Life[4])*My_High/100);
g.FillRectangle(brush5,5,5,ClientRectangle.Width/2, ClientRectangle.Height -10-
        (Form1.My_Life[0]+Form1.My_Life[1]+Form1.My_Life[2]+
        Form1.My_Life[3]+Form1.My_Life[4])*My_High/100);
// 6 Нижняя полоса
g.DrawRectangle(pen,5,5,ClientRectangle.Width/2, ClientRectangle.Height - 10-
        (Form1.My_Life[0]+Form1.My_Life[1]+Form1.My_Life[2]+
        Form1.My_Life[3]+Form1.My_Life[4]+Form1.My_Life[5])*
        My_High/100);
g.FillRectangle(brush6,5,5,ClientRectangle.Width/2, ClientRectangle.Height -10-
        (Form1.My_Life[0]+Form1.My_Life[1]+Form1.My_Life[2]+
        Form1.My_Life[3]+Form1.My_Life[4]+Form1.My_Life[5])*
        My_High/100);
// 7 Нижняя полоса
g.DrawRectangle(pen,5,5,ClientRectangle.Width/2, ClientRectangle.Height - 10-
        (Form1.My_Life[0]+Form1.My_Life[1]+Form1.My_Life[2]+
        Form1.My_Life[3]+Form1.My_Life[4]+Form1.My_Life[5]+
        Form1.My_Life[6])*My_High/100);
g.FillRectangle(brush7,5,5,ClientRectangle.Width/2, ClientRectangle.Height -10-
        (Form1.My_Life[0]+Form1.My_Life[1]+Form1.My_Life[2]+
        Form1.My_Life[3]+Form1.My_Life[4]+Form1.My_Life[5]+
        Form1.My_Life[6])*My_High/100);
// 8 Нижняя полоса
g.DrawRectangle(pen,5,5,ClientRectangle.Width/2, ClientRectangle.Height - 10-
        (Form1.My_Life[0]+Form1.My_Life[1]+Form1.My_Life[2]+
        Form1.My_Life[3]+Form1.My_Life[4]+Form1.My_Life[5]+
        Form1.My_Life[6]+Form1.My_Life[7])*My_High/100);
g.FillRectangle(brush8,5,5,ClientRectangle.Width/2, ClientRectangle.Height -10-
        (Form1.My_Life[0]+Form1.My_Life[1]+Form1.My_Life[2]+
        Form1.My_Life[3]+Form1.My_Life[4]+Form1.My_Life[5]+
```

```
        Form1.My_Life[6]+Form1.My_Life[7])*My_High/100);
// 9 Нижняя полоса
g.DrawRectangle(pen,5,5,ClientRectangle.Width/2, ClientRectangle.Height - 10-
        (Form1.My_Life[0]+Form1.My_Life[1]+Form1.My_Life[2]+
        Form1.My_Life[3]+Form1.My_Life[4]+Form1.My_Life[5]+
        Form1.My_Life[6]+Form1.My_Life[7]+Form1.My_Life[8])*
        My_High/100);
g.FillRectangle(brush9,5,5,ClientRectangle.Width/2, ClientRectangle.Height -10-
        (Form1.My_Life[0]+Form1.My_Life[1]+Form1.My_Life[2]+
        Form1.My_Life[3]+Form1.My_Life[4]+Form1.My_Life[5]+
        Form1.My_Life[6]+Form1.My_Life[7]+Form1.My_Life[8])*
        My_High/100);
// 10 Нижняя полоса
}
```

**Шаг 8. Создаем процедуру очистки текстбоксов в Form1**

Для кнопки «Очистить» пишем:

```
private void button1_Click_1(object sender, System.EventArgs e) {
foreach (Control NT in this.Controls)
if (NT is TextBox) {
NT.Text="0"; }
}
```

**Шаг 9. Создаем процедуру вызова диаграммы для просмотра**

Для опции главного меню «Диаграмма» пишем:

```
private void menuItem6_Click(object sender, System.EventArgs e) {
int i=0;
foreach (Control NT in this.Controls)
if (NT is TextBox) {// считываем значения
My_Life[i]=int.Parse(NT.Text); // пишем в массив
i=i+1;
        }
```

```
Form AnForm=new Form2();
AnForm.Show();
}
```

## 3.6. Связанные таблицы

Задача темы: изучить методику и получить навыки использования связей между таблицами в реляционных хранилищах данных

Разработка: Приложение «DataRelation» для оценки скорости выполнения запросов в связанных таблицах

Состав выполняемых функций:

- Файловые операции: автоматическое открытие и загрузка данных из двух файлов
- Алгоритмы: выборка данных из одного файла и добавление их в другой; оценка скорости выполнения запроса
- Информационно-справочные: О программе

Наращивание функциональности не предусматривается.

Защита данных – нет.

В качестве входных данных используется XML – файл Mersedes.xml с записями о периодах проведения регламентных работ и структурой, состоящей из полей: <Деталь>; <Цена>; <Фирма>; <Дата>; <Группа>; <О_ фирме>; <Услуга>; <Об _услуге>; <Пробег>; <ID> (фрагмент файла показан на рис.31).

Выходные данные программы:

- XML – файл test_norm.xml с выборкой записей из файла Mersedes.xml о фактическом пробеге в момент проведения регламентных работ и структурой, состоящей из полей: <Деталь>; <Норма пробега>; <Фактический пробег>; <ID>. Выходной файл содержит несколько записей с заполненными полями, значение фактического пробега в поле <Фактический пробег> равно 0 – это поле будет заполнено в ре-

зультате выполнения запроса. Полем связи таблиц является поле <Деталь>;

- графическое отображение результатов результата выполнения запроса в DataGrid файла test_norm.xml и скорости выполнения запроса в текстовых полях формы для SmartDevice.

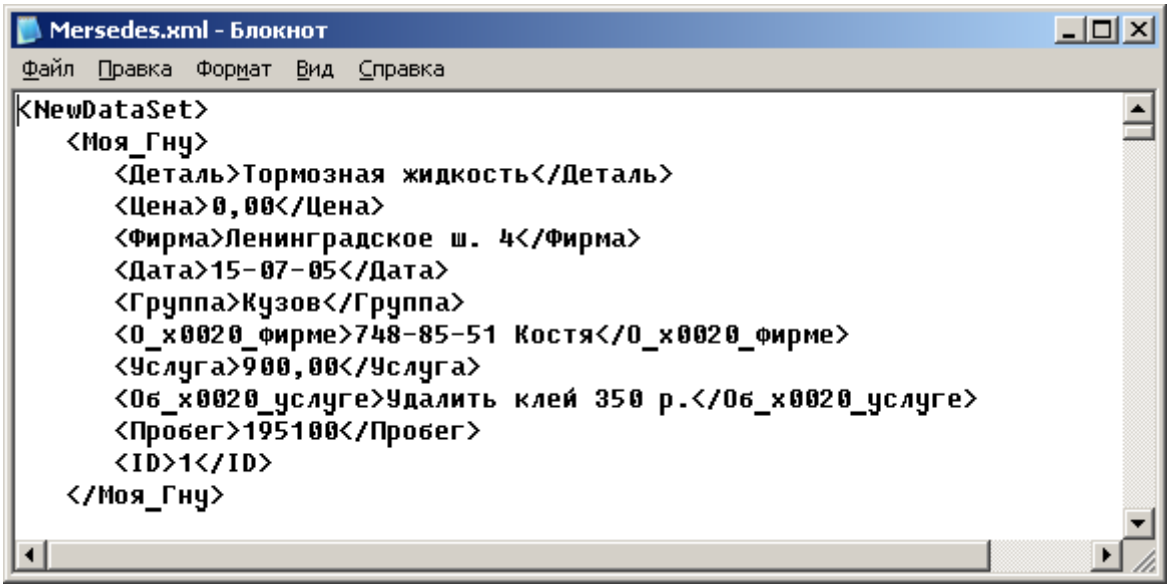


Рис. 31

**Шаг 1. Организапция дискового пространства**

В каталоге \bin\debug размещаем оба файла – Mersedes.xml и test_norm.xml

**Шаг 2. Организация графического интерфейса**

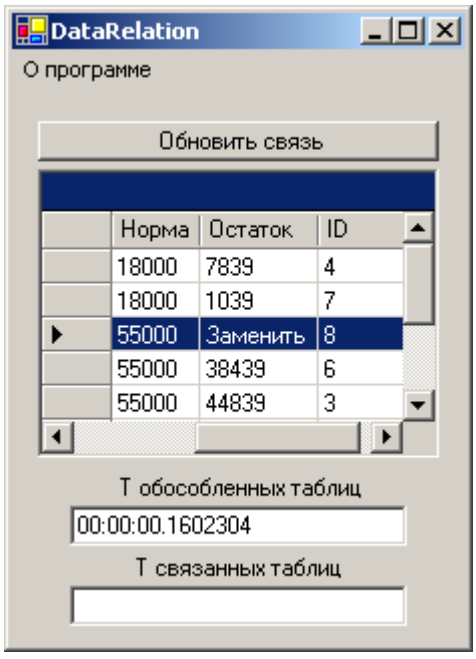

Рис.32

Одна форма с кнопкой «Обновить связь», элементом DataGrid для вывода данных файла test_norm.xml, двумя надписями «Т обособленных таблиц» и «Т связанных таблиц», двумя полями для вывода результатов запроса к связанным таблицам и для вывода выборки из данных Mersedes.xml по ключу таблицы test_norm.xml. Примерный дизайн показан на рис.32

Из не видимых элементов – главное меню с опцией «О программе».

**Шаг 3. Добавляем библиотеки**

```
using System;
using System.Drawing;
using System.Collections;
using System.Windows.Forms;
using System.Data;
//добавляем
using System.IO;
using System.Xml;
using System.Runtime;
```

**Шаг 4. Определяем переменные в public class Form1 : System.Windows.Forms.Form**

```
// организуем доступ (public static) из других классов (необязательно)
public static string file_name=@"\Mersedes.xml"; // пробег деталей
public static string file_norm=@"\test_norm.xml"; // нормы пробега
// глобальный датасет - объекты, содержащие ссылки на таблицы
public static DataSet dataSetApp, dataSet, dataSet1;
//глобальный дататейбл - таблицы
public static DataTable dataTableApp, dataTable, dataTable1 ;
```

**Шаг 5. Дополняем процедуру загрузки формы public Form1()**

```
{ // Required for Windows Form Designer support
InitializeComponent();
//создаем один объект для 2-х таблиц, иначе связь невозможна
```

```
dataSetApp=new DataSet();
dataTableApp=dataSetApp.Tables.Add("Нормы");//1-я таблица в dataSetApp
// добавляем столбцы
dataTableApp.Columns.Add("Деталь",System.Type.GetType("System.String"));
dataTableApp.Columns.Add("Норма",System.Type.GetType("System.String"));
dataTableApp.Columns.Add("Факт",System.Type.GetType("System.String"));
dataTableApp.Columns.Add("ID",System.Type.GetType("System.Int64"));
// вторая таблица в dataSetApp
dataTable1=dataSetApp.Tables.Add("Пробег замененных частей");
// добавляем столбцы
dataTable1.Columns.Add("Деталь",
        System.Type.GetType("System.String")).MaxLength=100;
dataTable1.Columns.Add("Пробег",
        System.Type.GetType("System.String")).MaxLength=10;
dataTable1.Columns.Add("Дата",
        System.Type.GetType("System.String")).MaxLength=10;
// пытаемся прочитать файл
FileStream fin;
try {
fin=new FileStream(file_norm,FileMode.Open);
fin.Close();
// Считываем значения из файла
XmlTextReader xml_in=new XmlTextReader(file_norm);
dataSetApp.ReadXml(@xml_in);
xml_in.Close();
        }
catch(IOException exc) {
MessageBox.Show("Нет файла с нормами пробега \n"+file_norm,
        "Ошибка открытия файла");
return;}
```

```
// программируем стиль грида, сортируем и привязываем данные файла норм
this.dataGrid1.TableStyles.Clear();
DataGridTableStyle tsApp= new DataGridTableStyle();
tsApp.MappingName="Нормы";
DataGridTextBoxColumn csApp=new DataGridTextBoxColumn();
csApp.Width=104;
csApp.MappingName="Деталь";
csApp.HeaderText="Деталь";
tsApp.GridColumnStyles.Add(csApp);

csApp = new DataGridTextBoxColumn();
csApp.Width=44;
csApp.MappingName="Норма";
csApp.HeaderText="Норма";
tsApp.GridColumnStyles.Add(csApp);

csApp = new DataGridTextBoxColumn();
csApp.Width=54;
csApp.MappingName="Факт";
csApp.HeaderText="Остаток";
tsApp.GridColumnStyles.Add(csApp);

csApp = new DataGridTextBoxColumn();
csApp.Width=50;
csApp.MappingName="ID";
csApp.HeaderText="ID";
tsApp.GridColumnStyles.Add(csApp);

this.dataGrid1.TableStyles.Add(tsApp);
DataView sortedView=new DataView(dataTableApp);
```

```
sortedView.Sort="Норма, Деталь";//  сортировка
dataGrid1.DataSource=sortedView;// привязываем
}
```

### Шаг 6. Создаем процедуру инициации запроса

Для кнопки «Обновить данные» пишем:

```
private void button1_Click(object sender, System.EventArgs e) {
FileStream fin;
try {
fin=new FileStream(file_name,FileMode.Open);
fin.Close();
        }
catch(IOException exc) {
MessageBox.Show ("Невозможно открыть файл \n"+file_name,"Ошибка");
return;}
// Считываем значения из файла
XmlTextReader xml_in=new XmlTextReader(file_name);
dataSet=new DataSet();
//Грузим данные из таблицы DataTable
dataTable=dataSet.Tables.Add("Моя_Гну");
dataTable.Columns.Add("Деталь",System.Type.GetType("System.String"));
dataTable.Columns.Add("Цена",System.Type.GetType("System.String"));
dataTable.Columns.Add("Фирма",System.Type.GetType("System.String"));
dataTable.Columns.Add("Дата",System.Type.GetType("System.String"));
dataTable.Columns.Add("Группа",System.Type.GetType("System.String"));
dataTable.Columns.Add("О фирме",System.Type.GetType("System.String"));
dataTable.Columns.Add("Услуга",System.Type.GetType("System.String"));
dataTable.Columns.Add("Об услуге",System.Type.GetType("System.String"));
dataTable.Columns.Add("Пробег",System.Type.GetType("System.String"));
dataTable.Columns.Add("ID",System.Type.GetType("System.Int64"));
```

```
dataSet.ReadXml(@xml_in);
xml_in.Close();
// добавляем первую строку для шаблона
int my_test=0;
int my_maxpath=0;
int my_maxpath_up=0;
int my_maxpath_down=0;
string my_date="err";
string my_path="err";
string my_fuel, my_service;
foreach (DataRow row0 in dataTable.Rows) {
//вычисляем максимальный пробег
my_maxpath_up=int.Parse(row0["Пробег"].ToString());
if (my_maxpath<=my_maxpath_up)
my_maxpath=my_maxpath_up;
        }
my_maxpath_up=0;// обнуляем для выборки в группе
foreach (DataRow row in dataTable.Rows) {//cicle
my_fuel=(row["Деталь"].ToString());
// фильтруем с сортировкой
DataRow[] my_rows=dataTable.Select("Деталь='"+my_fuel+"'");
foreach (DataRow row2 in my_rows ){
// вычисляем максимальный пробег детали
my_maxpath_up=int.Parse(row2["Пробег"].ToString());
if (my_maxpath_down<=my_maxpath_up) {
my_maxpath_down=my_maxpath_up;
my_path=(my_maxpath-my_maxpath_down).ToString();
my_date=(row2["Дата"].ToString());
                        }
                }
```

```
my_maxpath_up=0;// обнуляем для выборки в группе
my_maxpath_down=0;// обнуляем для выборки в группе
my_test=0;
// проверяем группу в новой таблице по самому большому ID
foreach (DataRow row1 in dataTable1.Rows){
my_service=(row1["Деталь"].ToString());
if (my_fuel==my_service)
my_test=1;
                }
if (my_test==0){
object[] a1={my_fuel,my_path,my_date};
dataTable1.Rows.Add(a1);
                }
        }
/* устанавливаем связь между таблицами dataTableApp и dataTable1:
Объект главной таблицы - имя связи - поле связи в главной - поле связи в связанной таблице. Главная таблица та - в которой больше записей, а не наоборот, в противном случае генерируется исключение, что не хватает записей
*/
DataColumn[]{dataTableApp.Columns["Деталь"]};// в таблице "Нормы"
DataColumn[]{dataTable1.Columns["Деталь"]};//"Пробег замененных частей"
// тестируем варианты
DateTime time_start;
DateTime time_finish;
if (MessageBox.Show("Выбираем данные из обособленных (Да)
        или связанных (Нет)таблиц?", "Тестируем
        варианты", MessageBoxButtons.YesNo,
        MessageBoxIcon.Question,MessageBoxDefaultButton.Button2) =
        =  DialogResult.Yes) {// выборка по ключу
// считываем значения из таблицы норм
```

```
time_start=DateTime.Now;
foreach (DataRow row0 in dataTableApp.Rows) {
my_fuel=(row0["Деталь"].ToString());
my_maxpath=int.Parse(row0["Норма"].ToString());
foreach (DataRow row1 in dataTable1.Rows) {
my_service=(row1["Деталь"].ToString());
//вычисляем максимальный пробег
my_maxpath_up=int.Parse(row1["Пробег"].ToString());
if (my_fuel==my_service)
my_maxpath=my_maxpath-my_maxpath_up;
                        }
if (my_maxpath>=int.Parse(row0["Норма"].ToString()))
row0[2]="Заменить";
else
row0[2]=my_maxpath.ToString();
                }
time_finish=DateTime.Now;
this.textBox1.Text=(time_finish-time_start).ToString();
        }
else{ // вариант запроса к  связанным таблицам
time_start=DateTime.Now;
dataSetApp.Relations.Add("Set_My_Relation",
dataTable1.Columns["Деталь"], dataTableApp.Columns["Деталь"], true);
for (int i=0; i<dataTable1.Rows.Count; i++) {
//вычисляем пробег
my_maxpath_up=int.Parse(dataTable1.Rows[i]["Пробег"].ToString());
// считываем значения из таблицы норм
foreach (DataRow row0
        in dataTable1.Rows[i].GetChildRows("Set_My_Relation")) {
my_fuel=(row0["Деталь"].ToString());
```

```
my_maxpath=int.Parse(row0["Норма"].ToString());
my_maxpath=my_maxpath-my_maxpath_up;
if (my_maxpath>=int.Parse(row0["Норма"].ToString()))
row0[2]="Заменить";
else
row0[2]=my_maxpath.ToString();
                    }
                }
dataSetApp.Relations.Remove("Set_My_Relation");// разрываем связь
time_finish=DateTime.Now;
this.textBox2.Text=(time_finish-time_start).ToString();
        }
time_finish=DateTime.Now;
MessageBox.Show("Загружены данные из файла \n"+file_name, "Готово");
}
```

**Шаг 7. Создаем процедуру просмотра строк DataGrid**

```
private void dataGrid1_Click(object sender, System.EventArgs e) {
int N_row=dataGrid1.CurrentCell.RowNumber;
MessageBox.Show(// вывод значения ячейки
        dataGrid1[N_row,0].ToString()+"\n"+
        dataGrid1[N_row,1].ToString()+"\n"+
        dataGrid1[N_row,2].ToString(),dataGrid1[N_row,3].ToString());
}
```

**Шаг 8. Создаем процедурку «О программе»**

Для опции главного меню «О программе» пишем:

```
private void menuItem1_Click(object sender, System.EventArgs e) {
MessageBox.Show("Проблема:\n"+"Есть 2 XML-файла,из одного нужно
        выбрать данные и ввести их в другой.\n"+ "Решение:\n"+"Выборка
        перебором записей по условию в таблицу DataTable1,\n"+"связанной
        с таблицей DataTableApp в которой заполняются вычисляемые
```

```
поля.\n"+"(C)Родигин Л.А., ноябрь 2006, платформа C# Visual
Studio 2003", "О программе");
}
```

### 3.7. Динамическое распределение памяти

Одной из сильных сторон операционной системы для Pocket PC - Windows CE - является то, что в основу ее разработки были положены преимущества версии Windows для настольных ПК. Вы можете заметить сходства между двумя системами, но важно обратить внимание и на различия между реализацией систем Windows CE и Windows NT. Одним из различий является то, что система Windows CE использует значительно облегченные процессы по сравнению с Windows NT, что позволило снизить затраты ресурсов на производительность. В отличие от Windows NT, система Windows CE задумывалась как настраиваемая система. Использование процессов для развертывания API системы позволило регулировать запросы к ПЗУ и ОЗУ в зависимости от того, какая часть ресурсов операционной системы требовалась для определенного устройства. Например, устройство можно настроить на работу только с одним процессом API — системным ядром (nk.exe). Другие устройства, которым требуется поддержка интерфейса пользователя, должны будут запустить процесс интерфейса пользователя (gwes.exe) [8].

Тем не менее, для рассматриваемых в книге версий Windows CE (5.0) существует ограничение на число одновременно выполняемых процессов (32), что может не проявляться в условиях эмуляции среды на настольной системе, но приводить к генерации исключения непосредственно на девайсе.

Задача темы: изучить методику и получить навыки программирования приложений, использующих динамическое распределение оперативной памяти.

Разработка: Приложение «Фотогалерея», позволяющее формировать списки и загружать рисунки, расположенные в различных местах диска.

Состав выполняемых функций:

- Файловые операции: открыть, сохранить, закрыть
- Алгоритмы: изменение размеров просмотра рисунка; прокрутка записей вперед и назад списка рисунков; добавление, замена и удаление записей из списка рисунков галереи.
- Информационно-справочные: О программе

Наращивание функциональности не предусматривается.

Защита данных – нет.

В качестве входных данных используются файлы галереи рисунков, файлы рисунков. Выходные данные программы – графическое отображение результатов в виде просматриваемого изображения рисунка, таблица просмотра записей галереи данных для SmartDevice

**Шаг 1. Разработка графического интерфейса**

Предусматривается одна форма с TabControl, включающим две закладки – Просмотр и Галерея.

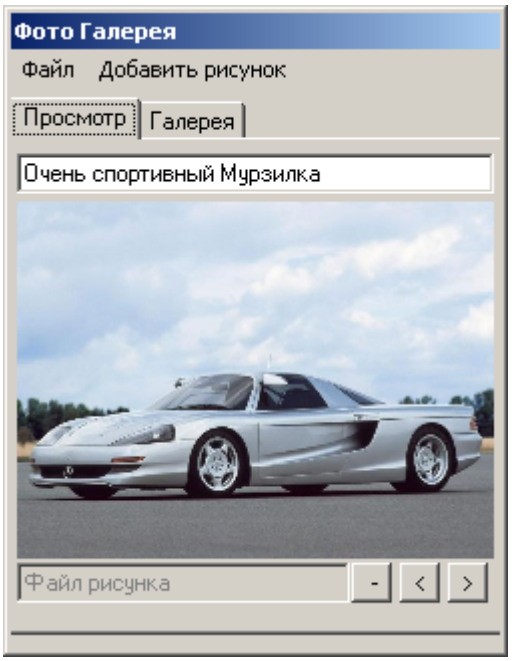


Рис.33

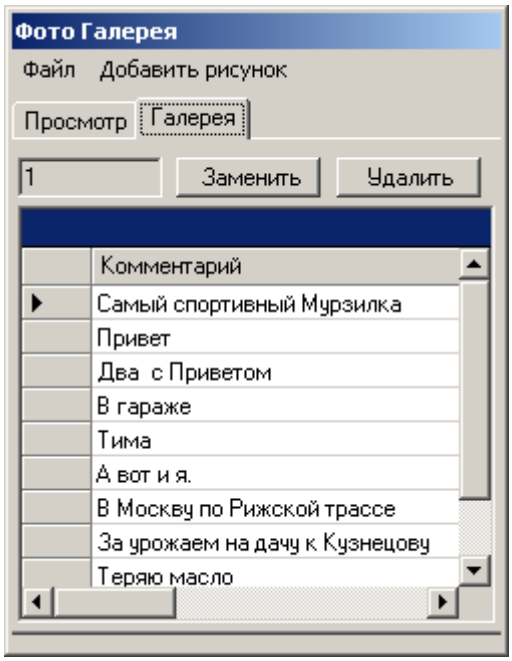


Рис.34

Закладка «Просмотр» содержит контейнер для просмотра рисунков PictureBox, текстовое поле для комментария к рисунку, текстовое поля для отображения имени файла рисунка (свойство ReadOnly = True), кнопку «-» для изменения размеров рисунка, кнопку «<» прокрутки просмотра фотога-

лереи назад, кнопку «>» для просмотра фотогалереи вперед и кнопку «Добавить рисунок» (изначально свойство Visible=False) для занесения в базу данных пути и имени файла рисунка.

Закладка «Галерея» содержит DataGrid для просмотра XML-файла списка рисунков, поле порядкого номера рисунка в галереи (свойство ReadOnly = True), кнопку «Заменить» для замены записи в списке рисунков галереи и кнопку «Удалить» для удаления записи из списка.

Из невидимых элементов интерфейса – главное меню с опциями «Файл» и «Добавить рисунок».

Меню «Файл» в свою очередь, содержит опции «Открыть галерею», «Сохранить галерею», «О программе» и «Выход».

Другими невидимыми элементами являются два OpenFileDialog (для девайса рекомендуется свойство InitialDirectory = FlashDisk\image\ ) для открытия файла галерей (свойство Filter = Файлы XML |*.xml) и для открытия файла рисунка (свойство Filter = Рисунки |*.jpg ), и один SaveFileDialog для записи файла галереи.

Примерный дизайн формы приведен на рис. 33 и 34.

**Шаг 2. Добавляем библиотеки**

```
using System;
using System.Drawing;
using System.Collections;
using System.Windows.Forms;
using System.Data;
//добавляем
using System.IO;
using System.Xml;
using System.Runtime.InteropServices; // beep
```

**Шаг 3. Добавляем переменные в public class Form1 : System.Windows.Forms.Form**

```
int my_picture; //номер файла
```

```
int my_width=238;//геометрия рисунка
int my_loc=1;// позиционирование рисунка
public string file_test=@"FlashDisk\image\auto_foto.xml";//
public static DataSet dataSetApp;// глобальный датасет
public static DataTable dataTableApp;//глобальный дататейбл
System.Int64 my_Int64;   // переменная номера строки
//переменная для удаления строки, чтобы не нарушать коллекцию foreach
DataRow remove_row;
DataRow newRow;
Int64 my_maxID=0;
string file_name;
Bitmap Circle; // создаем ссылочную переменную рисунка
// Runtime beep (на девайсе может не работать)
[DllImport("user32.dll")]
public static extern int MessageBeep(uint n);
```

**Шаг 4. Добавляем инициализацию public Form1()**

```
{
InitializeComponent();
// по умолчанию грузим шаблон галереи
dataSetApp=new DataSet();
dataTableApp=dataSetApp.Tables.Add("FotoDrive");
// добавляем столбцы
dataTableApp.Columns.Add("Комментарий",
        System.Type.GetType("System.String"));
dataTableApp.Columns.Add("t",System.Type.GetType("System.String"));
dataTableApp.Columns.Add("ID",System.Type.GetType("System.Int64"));
dataTableApp.Columns.Add("File",System.Type.GetType("System.String"));
// пытаемся прочитать файл
FileStream fin;
try {
```

```
fin=new FileStream(file_test,FileMode.Open);
fin.Close();
// Считываем значения из файла
XmlTextReader xml_in=new XmlTextReader(file_test);
dataSetApp.ReadXml(@xml_in);
xml_in.Close();
        }
catch(IOException exc) {// если файла нет, то создаем
// добавляем первую строку для шаблона
object[] a1={"Самый спортивный Мурзилка","1","1",
        @"FlashDisk\image\1.jpg"};    // массив строки
dataTableApp.Rows.Add(a1);
this.textBox1.Text="Самый спортивный Мурзилка";
FileStream fin_out;
fin_out=new FileStream(file_test,FileMode.OpenOrCreate);
fin_out.Close();
StreamWriter xml_out2;
xml_out2=new StreamWriter(file_test);//}
// форматируем xml
XmlTextWriter xmlWriter=new XmlTextWriter(xml_out2);
// способ форматирования
xmlWriter.Formatting=Formatting.Indented;
xmlWriter.Indentation=3;
// принимаем изменения в dataSet !!!
dataTableApp.AcceptChanges();
// выводим в xml
dataSetApp.WriteXml(xmlWriter,XmlWriteMode.IgnoreSchema);
xmlWriter.Close();
MessageBox.Show("Файл записан под именем \n"+
        @"FlashDisk\image\auto_foto.xml","Сохранение");
```

```
}
this.dataGrid1.Visible=true;
this.dataGrid1.TableStyles.Clear();
DataGridTableStyle tsApp= new DataGridTableStyle();
tsApp.MappingName="FotoDrive";
DataGridTextBoxColumn csApp=new DataGridTextBoxColumn();
csApp.Width=200;
csApp.MappingName="Комментарий";
csApp.HeaderText="Комментарий";
tsApp.GridColumnStyles.Add(csApp);
csApp = new DataGridTextBoxColumn();
csApp.Width=10;
csApp.MappingName="t";
csApp.HeaderText="t";
tsApp.GridColumnStyles.Add(csApp);
csApp = new DataGridTextBoxColumn();
csApp.Width=50;
csApp.MappingName="ID";
csApp.HeaderText="ID";
tsApp.GridColumnStyles.Add(csApp);
csApp = new DataGridTextBoxColumn();
csApp.Width=300;
csApp.MappingName="File";
csApp.HeaderText="File";
tsApp.GridColumnStyles.Add(csApp);
this.dataGrid1.TableStyles.Add(tsApp);
DataView sortedView=new DataView(dataTableApp);
sortedView.Sort="ID";//  сортировка
dataGrid1.DataSource=sortedView;// привязываем
}
```

### Шаг 5. Создаем процедуру просмотра нового рисунка для добавления в галерею

Для опции главного меню «Добавить рисунок» пишем:

```
private void menuItem2_Click_1(object sender, System.EventArgs e) {
if (openFileDialog1.ShowDialog()==DialogResult.OK) {
file_name=openFileDialog1.FileName;
this.tabControl1.SelectedIndex=0;
this.textBox1.Text=file_name;
Circle =new Bitmap(file_name);
pictureBox1.Image = Circle;
this.button5.Visible=true; // показываем кнопку добавления рисунка в список
        }
}
```

### Шаг 6. Создаем процедуру добавления рисунка в галерею

Для кнопки «Добавить в галерею» пишем:

```
private void button5_Click(object sender, System.EventArgs e) {
// вычисляем максимальный номер записи:
foreach (DataRow my_Row in dataTableApp.Rows) {
string my_IDD=(my_Row["ID"].ToString());
my_Int64=Int64.Parse(my_IDD);
if (my_maxID<=my_Int64) {
my_maxID=my_Int64+1;
                }
        }
this.textBox3.Text=my_maxID.ToString();
// вводим данные
newRow=dataTableApp.NewRow();
newRow["Комментарий"]=this.textBox2.Text;
newRow["t"]="1";
newRow["File"]=this.textBox1.Text;
```

```
// переписываем номер
newRow["ID"]=this.textBox3.Text;
// обнуляем  my_maxID
my_maxID=0;
// добавляем запись
dataTableApp.Rows.Add(newRow);
this.tabControl1.SelectedIndex=1;
MessageBox.Show("Ok");
this.button5.Visible=false; // прячем кнопку добавления рисунка
}
```

**Шаг 7. Создаем процедуру выборочного просмотра рисунков из галереи**

Для клика строки DataGrid пишем:

```
private void dataGrid1_Click(object sender, System.EventArgs e) {
int N_row=dataGrid1.CurrentCell.RowNumber;
this.tabControl1.SelectedIndex=0;
file_name=dataGrid1[N_row,3].ToString();
this.textBox1.Text=file_name;
this.textBox2.Text=dataGrid1[N_row,0].ToString();
this.textBox3.Text=dataGrid1[N_row,2].ToString();
// значение переменной в функции Pic_Show();
my_picture=int.Parse(this.textBox3.Text);
Pic_Show(); // вызываем функцию просмотра рисунка
return;
}
```

**Шаг 8. Создаем функцию просмотра рисунка**

```
void Pic_Show(){ // проверка существования
FileStream fin_picShow;
try {
fin_picShow=new FileStream(file_name,FileMode.Open);
fin_picShow.Close(); // закрываем поток
```

```
Circle =new Bitmap(file_name);// динамическое выделение памяти - здесь
        }
catch(FileNotFoundException exc) {// если файл не найден
MessageBox.Show("Запись "+my_picture+". Нет файла рисунка\n" + file_name,
        "Ошибка");
return;}
catch(IOException exc) {// файл недоступен
MessageBox.Show("Запись "+my_picture+". Используется другим
        процессом\n" + file_name,"Ошибка");
Circle.Dispose();// чистим динамическую память
Circle =new Bitmap(file_name);
        }
catch {/* перехват всех исключений –
необходимость только для PC из-за конфликта версий
*/
MessageBox.Show(this.textBox3.Text+"\n"+
        "Запись "+my_picture+". Фиг знает, где висим\n"+
        this.textBox1.Text+"\n"+file_name+"\n"+
        this.textBox2.Text,"Ошибка");
//конфликт версий, на PC стоит более поздняя: основная 100, доп. 0042
return; // чтобы программы не завершелась аварийно
        }
this.tabControl1.SelectedIndex=0;
pictureBox1.Image = Circle;
return; // очистку динамической памяти предоставляем сборщику мусора
}
```

**Шаг 9. Прокручиваем записи вперед**

Для кнопки «Вперед» пишем:

```
private void button1_Click_1(object sender, System.EventArgs e) {
my_picture=int.Parse(this.textBox3.Text);
```

```
//вычисляем максимальную запись
foreach (DataRow my_Row in dataTableApp.Rows) {
string my_IDD=(my_Row["ID"].ToString());
my_Int64=Int64.Parse(my_IDD);
if (my_maxID<=my_Int64) {
my_maxID=my_Int64+1;
        }
}
if (my_picture==my_maxID-1) {
my_picture=1;
        }
else
my_picture=my_picture+1;
this.textBox3.Text=my_picture.ToString();
int N_row=my_picture-1; //счет идет с нулевой строки
file_name=dataGrid1[N_row,3].ToString();
this.textBox1.Text=file_name;
this.textBox2.Text=dataGrid1[N_row,0].ToString();
Pic_Show();
return;
}
```

**Шаг 10. Прокручиваем записи назад**

Для кнопки «Назад» пишем:

```
private void button3_Click(object sender, System.EventArgs e) {
my_picture=int.Parse(this.textBox3.Text);
//вычисляем максимальную запись
foreach (DataRow my_Row in dataTableApp.Rows) {
string my_IDD=(my_Row["ID"].ToString());
my_Int64=Int64.Parse(my_IDD);
if (my_maxID<=my_Int64) {
```

```
my_maxID=my_Int64+1;
                }
        }
if (my_picture==1) {
this.textBox3.Text=my_maxID.ToString();
my_picture=int.Parse(this.textBox3.Text)-1;
        }
else
my_picture=my_picture-1;
this.textBox3.Text=my_picture.ToString();
int N_row=my_picture-1;
file_name=dataGrid1[N_row,3].ToString();
this.textBox1.Text=file_name;
this.textBox2.Text=dataGrid1[N_row,0].ToString();
Pic_Show();
return;
}
```

**Шаг 11. Изменяем геометрию просматриваемого рисунка**

```
private void button4_Click(object sender, System.EventArgs e) {
if (my_width==238) {
my_width=134;
my_loc=53;
this.button4.Text="+"; // меняем свойство кнопки
        }
else {
my_width=238;
my_loc=1;
this.button4.Text="-";
        }
this.pictureBox1.Location = new System.Drawing.Point(my_loc, 30);
```

```
this.pictureBox1.Size = new System.Drawing.Size(my_width, 178);
}
```

**Оставшуюся функциональность проекта предлагается разработать самостоятельно.**

## 3.8. Схема данных XML - документа

Перед заполнением объектов типа DataSet необходимо иметь структуру данных XML-файла. Во всех рассмотренных выше примерах эта структура создавалась программным путем с явным добавлением в DataSet таблиц, колонок и ограничений. Существует другой метод генерации DataSet - на основе схемы данных XML-файла. Схема данных – это файл в котором описана структура данных: связи между полями; ограничения на поля; спецификация типов полей.

Задача темы: изучить методику и получить навыки создания и использования схем данных XML-файлов

Разработка: Приложение «XML Structure Definition» для создания и просмотра схем данных XML-файлов

Состав выполняемых функций:

- Файловые операции: открыть, закрыть, сохранить
- Алгоритмы: просмотр данных и XSD-схем XML-файлов
- Информационно-справочные: нет

Наращивание функциональности не предусматривается.

Защита данных – нет.

В качестве входных данных используются XML–файлы с данными. Выходные данные программы – файлы-схемы и графическое отображение данных и содержания схемы XML-файла для SmartDevice

### Шаг 1. Разработка графического интерфейса

Предусматривается одна форма с текстовым полем вывода имени загруженного XML-файла, элементом DataGrid для просмотра данных XML-

файла и листбокса для просмотра содержимого схемы, а также кнопка построчного просмотра листбокса.

Из невидимых элементов – главное меню с опцией «Файл XML».Меню «Файл», в свою очередь, содержит опции «Открыть» и «Создать схему».

Элемент OpenFileDialog имеет свойство Filter = Файлы XML|*.XML, а элемент SaveFileDialog имеет свойство Filter = Схемы файлов XML|*.XSD (файлы схем имеют расширение XSD).

Примерный дизайн показан на рис. 35.

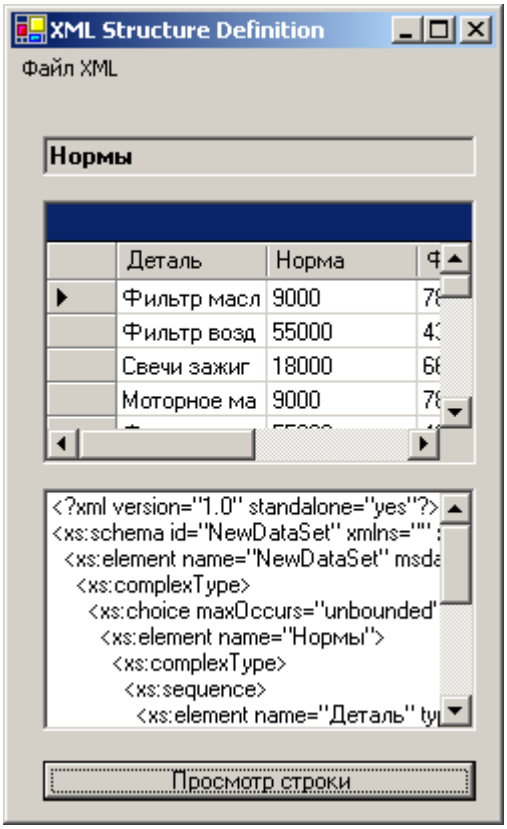


Рис. 35

**Шаг 2. Добавляем библиотеки**

```
using System;
using System.Drawing;
using System.Collections;
using System.Windows.Forms;
using System.Data;
//добавляем
using System.IO;
```

```
using System.Xml;
using System.Xml.Schema;
```

**Шаг 3. Добавляем переменные в public class Form1 : System.Windows.Forms.Form**

```
string file_name=@"c:\dataSet.xml"; // переменная для имени файла
XmlTextReader xml_read;
FileStream fin;
DataSet DataXML;
```

**Шаг 4. Создаем процедуру открытия XML-файла произвольной структуры**

Для опции «Открыть» меню «Файл XML» пишем:

```
private void menuItem2_Click(object sender, System.EventArgs e) {
if (openFileDialog1.ShowDialog()==DialogResult.OK) {
file_name=openFileDialog1.FileName;
try {
fin=new FileStream(file_name,FileMode.Open);
xml_read=new XmlTextReader(fin);
MessageBox.Show(file_name,"Открываем файл");
        }
catch(IOException exc) {
MessageBox.Show ("Невозможно открыть файл \n"+file_name,"Ошибка");
return;}
DataXML=new DataSet();
DataXML.ReadXml(xml_read, XmlReadMode.InferSchema);
string str="";
this.listBox1.Items.Clear();
int i=0;
for (i=0;i<DataXML.Tables.Count;i++){//считаем число таблиц в dataSet
this.textBox1.Text=(DataXML.Tables[i].TableName.ToString());// имя таблицы
for (int j=0; j<DataXML.Tables[i].Rows.Count; j++){
```

```
// сосчитали строки в i-ой таблице
for (int k=0; k<DataXML.Tables[i].Columns.Count; k++){//добавляем
str=str+" "+DataXML.Tables[i].Rows[j][k].ToString();
                }
this.listBox1.Items.Add(str); // пишем в листбокс
dataGrid1.DataSource=DataXML.Tables[i]; // очень просто привязали
str="";//обнуляем – изщряемся для листбокса
                }
        }
fin.Close();
        }
}// данные выведены и в листбокс, и в датагрид
```

**Шаг 5. Сохраняем схему полученных данных**

Для опции «Сохранить схему» меню «Файл XML» пишем:

```
private void menuItem3_Click(object sender, System.EventArgs e) {
if (this.dataGrid1.DataSource==null){
MessageBox.Show("Откройте файл *.XML","Ошибка");
return;}
int L_name=this.file_name.Length-4;
file_name=file_name.Substring(0,L_name)+".xsd";
saveFileDialog1.FileName=file_name;
if (this.saveFileDialog1.ShowDialog()==DialogResult.OK) {
file_name=saveFileDialog1.FileName;
DataXML.WriteXmlSchema(file_name);
//..выводим листбокс
try {//открываем файл
fin=new FileStream(file_name,FileMode.Open);
        }
catch(IOException exc) {//если файл недоступен
 MessageBox.Show ("Невозможно открыть файл \n"+file_name,"Ошибка");
```

```
return;}
long L=fin.Length; // вычисляем длину файла (мусор?)
// Считываем значения из файла
StreamReader fstr_in=new StreamReader(fin);
//в переменную считываются все символы, пока не кончится строка
string sc="0"; // загружаемая из файла строка
int s=0; // число символов в строке
listBox1.Items.Clear();
while((sc=fstr_in.ReadLine())!=null) {
s=s+sc.Length;
listBox1.Items.Add(sc); //добавляем строку в листбокс
        }
fstr_in.Close(); // закрываем поток и выводим сообщение с результатами
        }
}
```

**Шаг 6. Создаем процедуру просмотра строки листбокса со схемой**

Для кнопки «Просмотр строки» пишем:

```
private void button1_Click(object sender, System.EventArgs e) {
if (this.dataGrid1.DataSource = = null){//если таблица пустая
MessageBox.Show("Откройте файл *.XML","Ошибка");
return;}
MessageBox.Show(this.listBox1.SelectedItem.ToString(),"Строка " +
        this.listBox1.SelectedIndex.ToString());
}
```

**Шаг 7. Создаем функцию объединения файла данных и схемы**

```
void Convert(){
string xmlfile=@"c:\dataSet.xml";
string xmlfileNew=@"c:\dataSetNew.xml";
DataSet ConvertData=new DataSet();
ConvertData.ReadXml(xmlfile, XmlReadMode.InferSchema);
```

```
ConvertData.WriteXml(xmlfileNew, XmlWriteMode.WriteSchema);
}
```

Вы можете самостоятельно подключить функцию к какому-либо новому элементу приложения.

## ЗАКЛЮЧЕНИЕ

ADO.NET предоставляет новые перспективы и новый подход для доступа к данным в рамках технологии .NET Framework. В ADO.NET аккумулированы лучшие способы доступа к данным для создания универсального API, позволяющего осуществлять доступ как к реляционным, так и к неряляционным данным. Для реализации концепции отсоединенного доступа к данным в ADO.NET используются объекты DataSet. Эти объекты не зависят от источников данных. Управляемые провайдеры данных применяются для подключения объектов Dataset к источникам данных и выполнения команд. ADO.NET поддерживает совместимость с существующими программами, созданными с помощью ADO, и использует XML для поддержки иерархического представления данных.

В ADO.NET XML является базовым форматом для передачи данных через брандмауеры и этот формат рекомендуется для использования в технологии Web – форм. Преобразование как реляционных, так и неряляционных данных в формат XML позволяет использовать иерархическую модель для непоследовательного поиска записей. То есть появляется возможность использовать установленные между таблицами отношения типа «родитель-потомок» при поиске запрошенных записей.

ADO.NET использует XML как способ передачи данных к объекту Dataset и обратно, причем возможно строить Dataset только по данным, что, на наш взгляд, является хорошим решением для SmartDevice. Однако, такие построения чреваты получением неверной структуры, из-за неизвестных типов и размеров полей. При этом полученные данные никак не описывают связи между таблицами.

Для решения этой проблемы Microsoft предлагает использовать файлы XSD, содержащие описание того, как отображаются данные из Dataset в XML – формате. Таким образом, когда данные пересылаются по сети, используется два файла – XML, содержащий собственно данные, и XSD, содержащий метаданные: структуру данных и связи между ними со всеми установленными ограничениями.

ADO.NET – это библиотека NET–классов, которые позволяют подсоединиться к данным и манипулировать ими. Несмотря на то, что все примеры в данной книге написаны на языке C#, общеязыковая исполняющая среда CLR.NET позволяет писать практически аналогичный код на Visual Basic.NET, C++ с управляемыми расширениями или другом управляемом языке.

C#, по определению Microsoft, является «простым, современным, объектно-ориентированным языком программирования, обеспечивающим безопасность типов и представляющим собой логическое развитие языков C и C++» [4]. Язык C# разрабатывался специально для платформы NET – технологии «программное обеспечение в виде службы» для создания сложных Интернет-приложений с использованием всего разнообразия возможностей Windows API. Язык C# позволяет преобразовать любой компонент в службу, доступную по Интернет. С другой стороны, существующие Web-службы могут быть интерпретированы как родные C#-объекты. Таким образом, разработчики смогут объединить собственные навыки объектно-ориентированного программирования, включая полученные и в данной книге, с возможности Web–служб.

## РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА


1. Байдачный С.С. NET Framework. Секреты создания Windows-приложений. – М.: СОЛОН-Пресс, 2004. – 496 с.
2. Вильдермьюс Ш. Практическое использование ADO.NET. Доступ к данным в Интернет. – Издательский дом «Вильямс», 2003. - 288 с.
3. Гамильтон Б. ADO.NET для профессионалов. СПб.: Питер, 2005.- 576 с.
4. Джаяраман Р., Сетхупатхи М. Использование C#. Специальное издание. – М.: Издательский дом «Вильямс», 2002. – 528 с.
5. Павловская Т.А. C#. Программирование на языке высокого уровня. Учебник для ВУЗов. – СПб.: Питер, 2007. – 432 с.
6. Петцольд Ч. Программирование с использованием Microsoft Windows Forms. Мастер-класс. – М.: Русская редакция; Спб.: Питер, 2006. – 434 с.
7. Шилдт Г. C#: Учебный курс. – СПб.: Питер: К.: Издательская группа BHV, 2002.- 512 с.
8. Яо П. Мобильные системы: Введение в новые функции операционной системы Windows Embedded CE 6.0. – MSDN Magazine, Desember, 2006. http://msdn.microsoft.com/msdnmag/issues/06/12/WindowsCE/Default.aspx