 Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Уральский федеральный университет имени первого Президента России Б.Н. Ельцина» (УрФУ)

Институт радиоэлектроники и информационных технологий – РтФ

Школа профессионального и академического образования

Проект «Проводник»  
по дисциплине

«Распределенные информационно-аналитические системы»

Студент: Адилов Марсель Хайитбойович \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(ФИО) (Подпись)

Группа: РИ-391223

Дата: 14.06.2021

Екатеринбург

2022

Оглавление

[1. Описание программы 3](#_Toc106015751)

[2. Интерфейс 5](#_Toc106015752)

[3. Visual Studio 8](#_Toc106015753)

[4. Инструменты и методы при разработке 10](#_Toc106015754)

[5. Листинг 13](#_Toc106015755)

1. Описание программы

Программа под названием «Проводник» представляет собой файловый обозреватель, позволяющий перемещаться по каталогам файловой системы компьютера под управлением операционной системы Windows. Интерфейс представляет из себя окно для вывода информации, поле для ввода команд и строка меню с функциональными кнопками.

Возможности программы:

* перемещение по файловой системе;
* открытие текстовых файлов для чтения в заданной кодировке;
* открытие текстовых файлов для редактирования в заданной кодировке;
* создание текстовых файлов в заданной кодировке;
* сохранение текстовых файлов в заданной кодировке.

Поддерживаемые форматы файлов для чтения: .txt, .log, .md, .cs, .c, .cpp, .html, .xml, .fb2.

Поддерживаемые форматы файлов для редактирования: .txt, .log, .md, .cs, .c, .cpp.

Реализованные команды:

|  |  |  |
| --- | --- | --- |
| Команда | Код | Описание |
| Открыть файл из списка | *<номер файла>* | Открывает файл/папку под введенным номером из списка *<номер файла>*, представленном в окне для вывода информации. Кодировка файла выставлена в соответствии с выбранной кодировкой в поле выбора кодировки. |
| Открыть путь | *-open [path]* | Открывает файл/папку по заданному пути *[path]* в соответствии с выбранной кодировкой в поле выбора кодировки |
| Открыть файл с кодировкой | *-open [path / number] [page\_code]* | Открывает файл по заданному пути или номеру из списка *[path / number]* с указанной кодировкой *[page\_code]* |
| Редактировать файл | *-edit [path / number]* | Открывает файл по заданному пути или номеру из списка *[path / number]* в соответствии с выбранной кодировкой в поле выбора кодировкидля дальнейшего редактирования содержимого |
| Редактировать файл с кодировкой | *-edit [path / number] [page\_code]* | Открывает файл по заданному пути или номеру из списка *[path / number]* с указанной кодировкой *[page\_code]* для дальнейшего редактирования содержимого |
| Перейти к текущему каталогу | *-cur* | Открывает список файлов/папок в папке, с которой была запущена программа |
| Перейти к корневой папке | *-top* | Открывает список файлов/папок в корневой папке С:\ |
| Справка | *-info* | Вызывает справку, в которой указаны доступные команды |

1. Интерфейс

Окно программы состоит из трех секций: верхней, средней и нижней.

Верхняя часть включает в себя строку меню («Справка» и «Выход»), кнопки «Создать», «Открыть и редактировать», «Сохранить»**, «**Список файлов**»**, «Справка» и поле выбора кодировки (UTF-8, Windows-1251 или CP866).

Средняя часть выводит информацию, такую как список файлов/папок, содержимое файлов или справку. Данная секция имеет функцию прокрутки текста, бегунок расположен справа от окна. Также в этой части интерфейса происходит взаимодействие с содержимым текстового файла, если была нажата кнопка «Открыть и редактировать» или прописана команда *-edit*.

Нижняя часть представляет из себя поле для ввода команд, а также кнопки «Выполнить», «Назад», «Корневой каталог» и «Текущий каталог».

При запуске программы пользователя встречает справка с доступными командами и список файлов/папок корневой папки.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 2.1 – Справка

Кнопка «Создать» отвечает за создание нового текстового файла. При ее нажатии средняя секция окна программы очищается и становится активной, пользователь может вводить текст будущего файла.

Кнопка «Открыть и редактировать» вызывает системный Проводник, в котором пользователь может выбрать текстовый файл, который он хочет открыть для дальнейшего редактирования. После того, как пользователь сделал выбор, средняя секция окна программы выводит содержимое файла и становится активной, пользователь может редактировать текст файла. Кодировка файла при открытии устанавливается в соответствии с выбранной в поле выбора кодировок.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2.2 – Редактирование текстового файла

Кнопка «Сохранить» вызывает системный Проводник, в котором пользователь указывает наименование и формат файла для сохранения в выбранном каталоге. Кодировка файла при сохранении устанавливается в соответствии с выбранной в поле выбора кодировок.

Кнопка **«**Список файлов**»** вызывает список файлов и папок в последней посещаемой пользователем директории.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2.3 – Список файлов и папок

Кнопка и пункт меню **«**Справка**»** вызывают справку, в которой указаны доступные для использования команды. Такой же функционал предоставляет команда *-info*, введенная в поле для ввода команд.

При нажатии кнопки «Выполнить» программа считывает команду с поля ввода команд и запускает процесс ее выполнения. Также введенную команду можно выполнить нажатием клавиши *Enter (Ввод)*.

При нажатии кнопки «Назад» программа поднимается на папку выше (в родительскую) и показывает содержимое этой папки. Такой же функционал предоставляет команда *0*, введенная в поле для ввода команд.

Кнопка «Корневой каталог» открывает список файлов и папок в корневой папке С:\. Такой же функционал предоставляет команда *-top*, введенная в поле для ввода команд.

Кнопка «Текущий каталог» открывает список файлов и папок в папке, с которой была запущена программа. Такой же функционал предоставляет команда *-cur*, введенная в поле для ввода команд.

1. Visual Studio

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3.1 – код Program.cs

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3.2 – код Form1.cs

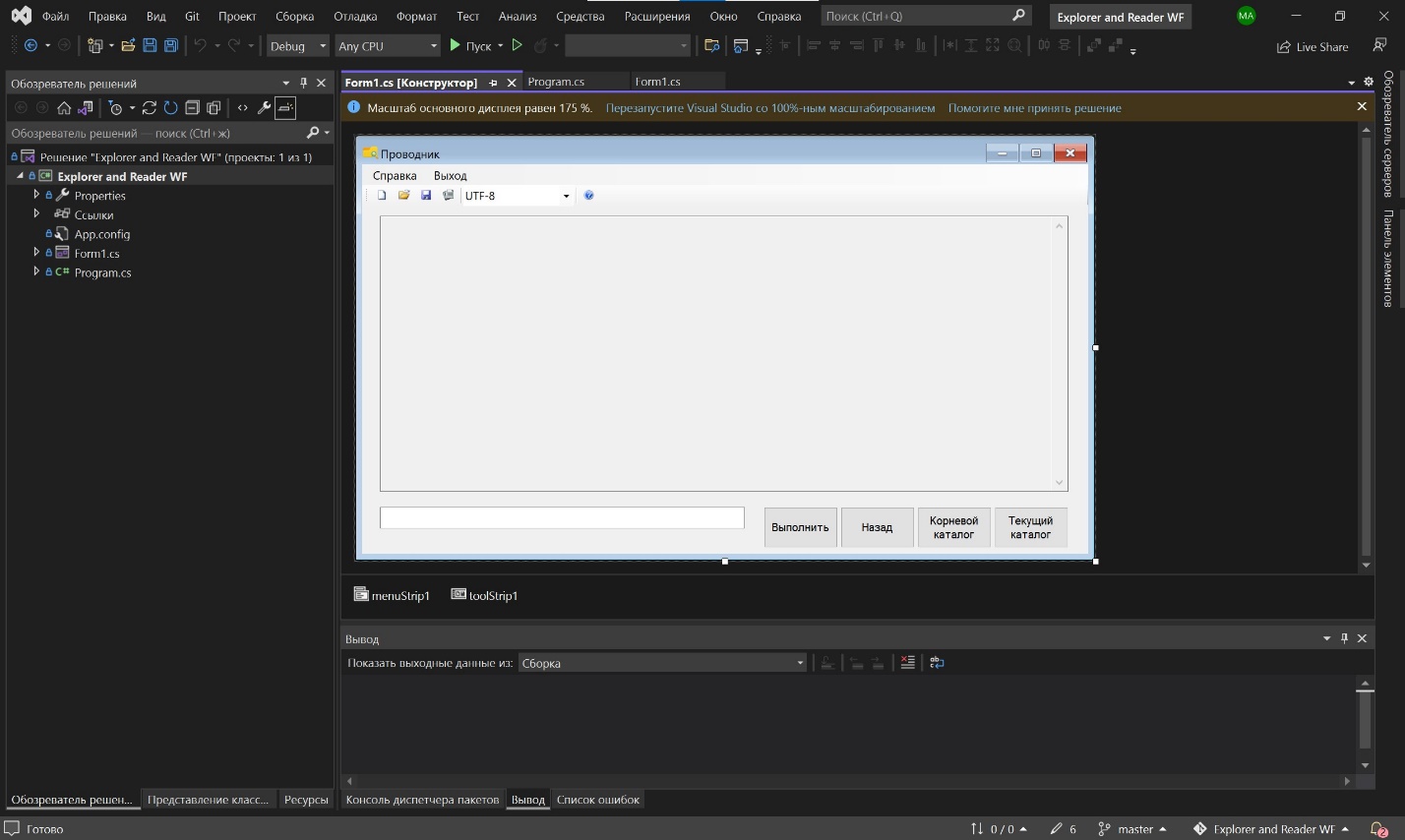


Рисунок 3.3 – конструктор Form1.cs

1. Инструменты и методы при разработке

В конструкторе Form1.cs использовались *menuStrip* для отображения пунктов меню («Справка» и «Выход»), *toolStrip* для отображения кнопок меню («Создать», «Открыть и редактировать», «Сохранить»**, «**Список файлов**»**, «Справка» и поле выбора кодировки), два *textBox* для ввода и вывода текста, четыре *button* для выполнения команд «Выполнить», «Назад», «Корневой каталог» и «Текущий каталог».

Переменные и методы в Program.cs:

* **public static string[] path** – хранит путь поэлементно.
* **private static string[] pathBackup** – копия path на случай ошибки, дающая возможность восстановления.
* **public static int pathIndex** – хранит количество объектов в пути.
* **private static int pathIndexBackup** – копия pathIndex на случай ошибки, дающая возможность восстановления.
* **private static string[] dirList** – хранит список файлов и папок в открытом каталоге.
* **private static int dirIndex**– хранит количество файлов и папок в открытом каталоге.
* **public static void Intro(TextBox textBoxOut)** – выводит в *textBoxOut* справку с доступными командами и вызывает *FileList(PathLinker(pathIndex), textBoxOut)*, выводящий список файлов и папок в открытом каталоге.
* **public static void FileList(string path, TextBox textBoxOut)** – принимает в качестве параметра сформированный путь директории *path* и выводит в *textBoxOut* соответствующий список файлов и папок.
* **public static string PathLinker(int i)** – принимает в качестве параметра количество объектов в требуемом для сформирования пути (*i*) и возвращает путь директории в качестве строки, «склеенной» из *i* элементов массива *path[]*.
* **public static void Request(string command, TextBox textBoxOut)** – принимает в качестве параметра строку *command*, введенную в поле для ввода команд, разделяет на составляющие при необходимости (команда, путь, кодировка) и вызывает другие методы в соответствии с командой (*-open, -top, -cur, <номер файла>: Open(…); -edit: EditFile(…); -info: Intro(…), 0: FileList(…)*). Метод распознает числа для открытия файлов/папок из списка и команды *-open, -top, -cur, -edit, -info*, в противном случае выводит на экран ошибку.
* **private static void Open(TextBox textBoxOut, int num = -1, string p = @"C:", string** **codePage = "65001")** – принимает в качестве параметров число *num* и строку *p*, которые сообщают о необходимости открыть файл/папку по заданному числу или пути соответственно. Параметр *codePage* отвечает за кодировку при открытии текстового файла. Метод открывает папки при помощи вызова *FileList(…),* а в случае файлов проверяет на соответствие поддерживаемым для открытия файловых форматов (.txt, .log, .md, .cs, .c, .cpp, .html, .xml, .fb2) и передает управление *OpenFile(…).*
* **private static void OpenFile(TextBox textBoxOut, string codePage = "65001")** – выводит содержимое текстовых файлов в *textBoxOut* в кодировке *codePage*. Для форматов .xml, .fb2 и .html осуществляется поэлементный вывод. При открытии .xml файла для каждого элемента вызывается метод *OpenXML(…).*
* **private static void OpenXML(TextBox textBoxOut, XNode node)** – выводит в *textBoxOut* элемент *node* и его атрибуты из .xml файла. Проверяется на наличие дочерних элементов, в случае обнаружения вызывается рекурсия.
* **private static void EditFile(TextBox textBoxOut, int num = -1, string p = @"C:", string codePage = "65001")** – выводит содержимое поддерживаемого текстового файла (.txt, .log, .md, .cs, .c, .cpp ) по выбранному номеру из списка *num* или пути *p* с кодировкой *codePage* в *textBoxOut*, который становится активным для пользователя и доступным для редактирования текста. В случае ошибки открытия выводит список файлов/папок из последней директории.

Переменные и методы в Form1.cs:

* **private static string command** – хранит введенную пользователем команду.
* **private void buttonRun\_Click(object sender, EventArgs e)** – вызывается при нажатии на кнопку «Выполнить». Считывает введенную пользователем команду и передает ее в Program.Request(…). Очищает поле для ввода команд и окно для вывода информации.
* **private void textBoxIn\_KeyPress(object sender, KeyPressEventArgs e)** – обработка нажатия клавиши Enter в поле для ввода команд для исполнения прописанной команды путем передачи управления в *buttonRun\_Click(…).*
* **private void buttonBack\_Click(object sender, EventArgs e)** – вызывается при нажатии на кнопку «Назад». Очищает поле для ввода команд и окно для вывода информации и передает управление в Program.Request(…) с командой *0*.
* **private void buttonTop\_Click (object sender, EventArgs e)** – вызывается при нажатии на кнопку «Корневой каталог». Очищает поле для ввода команд и окно для вывода информации и передает управление в Program.Request(…) с командой *-top*.
* **private void buttonCur\_Click(object sender, EventArgs e)** – вызывается при нажатии на кнопку «Текущий каталог». Очищает поле для ввода команд и окно для вывода информации и передает управление в Program.Request(…) с командой *-cur*.
* **private void справкаToolStripMenuItem\_Click(object sender, EventArgs e)** – вызывается при нажатии на кнопку меню «Справка». Очищает поле для ввода команд и окно для вывода информации и передает управление в Program.Request(…) с командой *-info*.
* **private void newFile(object sender, EventArgs e)** – вызывается при нажатии на кнопку меню «Создать». Очищает окно для вывода информации и делает его активным для ввода текста.
* **private void openFile(object sender, EventArgs e)** – вызывается при нажатии на кнопку меню «Открыть и редактировать». Открывает системный Проводник, в котором пользователь может выбрать текстовый файл, который он хочет открыть для дальнейшего редактирования. После того, как пользователь сделал выбор, средняя секция окна программы выводит содержимое файла и становится активной, пользователь может редактировать текст файла. Кодировка файла при открытии устанавливается в соответствии с выбранной в поле выбора кодировок.
* **private void saveFile(object sender, EventArgs e)** – вызывается при нажатии на кнопку меню «Сохранить». Открывает системный Проводник, в котором пользователь указывает наименование и формат файла для сохранения в выбранном каталоге. Кодировка файла при сохранении устанавливается в соответствии с выбранной в поле выбора кодировок.
* **private void exit(object sender, EventArgs e)** – вызывается при нажатии на кнопку меню «Выход». Осуществляет закрытие программы.
* **private void openFileList(object sender, EventArgs e)** – вызывается при нажатии на кнопку меню «Список файлов». Очищает окно для вывода информации и передает управление в Program.FileList(…) для вывода списка файлов/папок.

1. Листинг
   1. Program.cs

using System;

using System.Linq;

using System.Windows.Forms;

using System.Text;

using System.IO;

using System.Xml.Linq;

using System.Xml;

namespace WindowsFormsApp2

{

internal class Program

{

public static string[] path = new string[1000]; // хранит путь поэлементно

private static string[] pathBackup = new string[1000]; // копия path на случай ошибки, дающая возможность отката

public static int pathIndex = 0; // хранит кол-во объектов в пути

private static int pathIndexBackup = 0; // копия pathIndex на случай ошибки, дающая возможность отката

private static string[] dirList; // хранит список файлов в текущем каталоге

private static int dirIndex = 0; // кол-во файлов в текущем каталоге

public static void Intro(TextBox textBoxOut) //вступительное сообщение

{

textBoxOut.Text = ("+-----------------------------------------------------------------------------------------------------------------+\r\n");

textBoxOut.Text += ("|\t\t\tПроводник\t\t\t\t|\r\n");

textBoxOut.Text += ("+-------------------------------------------------------+--------------------------------------------------------+\r\n");

textBoxOut.Text += ("|\tКоманда\t\t\t|\t\tКод\t\t|\r\n");

textBoxOut.Text += ("+-------------------------------------------------------+--------------------------------------------------------+\r\n");

textBoxOut.Text += ("| · Открыть файл из списка\t\t| <номер файла>\t\t\t|\r\n");

textBoxOut.Text += ("| · Открыть путь\t\t\t| -open [path]\t\t\t|\r\n");

textBoxOut.Text += ("| · Открыть файл с кодировкой\t| -open [path / number] [page\_code]\t|\r\n");

textBoxOut.Text += ("| · Редактировать файл\t\t| -edit [path / number]\t\t|\r\n");

textBoxOut.Text += ("| · Редактировать файл (кодировка)\t| -edit [path / number] [page\_code]\t|\r\n");

textBoxOut.Text += ("| · Перейти к текущему каталогу\t| -cur\t\t\t\t|\r\n");

textBoxOut.Text += ("| · Перейти к корневой папке\t| -top\t\t\t\t|\r\n");

textBoxOut.Text += ("| · Справка\t\t\t| -info\t\t\t\t|\r\n");

textBoxOut.Text += ("+-------------------------------------------------------+--------------------------------------------------------+\r\n");

FileList(PathLinker(pathIndex), textBoxOut);

}

public static void FileList(string path, TextBox textBoxOut) //выводит список файлов в директории

{

textBoxOut.ReadOnly = true;

dirIndex = 0;

textBoxOut.Text += ($"\r\n>>>\t{path}\t<<<\r\n");

DirectoryInfo dInfo = new DirectoryInfo(path);

DirectoryInfo[] dir = dInfo.GetDirectories();

FileInfo[] spisok = dInfo.GetFiles();

dirList = new string[1000];

textBoxOut.Text += ($"\r\n№\t{"Имя файла",-50}\tРазмер (байт)");

if (path != @"C:\" && path != @"c:\" && path != @"C:" && path != @"c:") textBoxOut.Text += ($"\r\n0.\tНазад\r\n");

foreach (var d in dir)

{

dirList[dirIndex++] = d.Name;

textBoxOut.Text += ($"\r\n{dirIndex}.\t[{d.Name.ToUpper() + "]",-50}");

}

foreach (var f in spisok)

{

dirList[dirIndex++] = f.Name;

textBoxOut.Text += ($"\r\n{dirIndex}.\t{f.Name,-50}\t{f.Length:#,#}");

}

textBoxOut.Text += ("\r\n<Введите номер или команду>");

}

public static string PathLinker(int i) //собирает путь воедино

{

string pathLinked = "";

for (int j = 0; j <= i; j++) pathLinked = pathLinked + path[j] + @"\";

return pathLinked;

}

public static void Request(string command, TextBox textBoxOut) //обрабатывает введенную команду

{

textBoxOut.ReadOnly = true;

try

{

char symbol = command[0];

if (symbol == '-') //обрабока команд

{

if (command.StartsWith("-open"))

{

string[] tmp = command.Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);

if (tmp.Length > 2) //если задана кодировка

{

try //если задан файл числом

{

Form1.encodingPage = tmp[2];

Open(textBoxOut, Convert.ToInt32(tmp[1]) - 1, null, Form1.encodingPage);

}

catch //если задан путь

{

Form1.encodingPage = tmp[2];

Open(textBoxOut , - 1, tmp[1], Form1.encodingPage);

}

return;

}

else //если кодировка не задана

{

Open(textBoxOut, - 1, tmp[1], Form1.encodingPage);

return;

}

}

else if (command.StartsWith("-top"))

{

Open(textBoxOut, -1);

return;

}

else if (command.StartsWith("-cur"))

{

Open(textBoxOut, -1, Directory.GetCurrentDirectory());

return;

}

else if (command.StartsWith("-edit"))

{

string[] tmp = command.Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);

if (Convert.ToInt32(tmp[1]) <= dirIndex && Convert.ToInt32(tmp[1]) > 0)

{

if (tmp.Length > 2) //если задана кодировка

{

try //если задан файл числом

{

Form1.encodingPage = tmp[2];

EditFile(textBoxOut, Convert.ToInt32(tmp[1]) - 1, null, Form1.encodingPage);

}

catch //если задан путь

{

Form1.encodingPage = tmp[2];

EditFile(textBoxOut, -1, tmp[1], Form1.encodingPage);

}

return;

}

else //если кодировка не задана

{

try //если задан файл числом

{

EditFile(textBoxOut, Convert.ToInt32(tmp[1]) - 1, null, Form1.encodingPage);

}

catch //если задан путь

{

EditFile(textBoxOut, -1, tmp[1], Form1.encodingPage);

}

return;

}

}

else

{

textBoxOut.Text += ("\r\nОшибка. Попробуйте снова.");

FileList(PathLinker(pathIndex), textBoxOut);

return;

}

}

else if (command.StartsWith("-info"))

{

Intro(textBoxOut);

return;

}

else

{

textBoxOut.Text += ("\r\nТакой команды не существует. Попробуйте снова.");

FileList(PathLinker(pathIndex), textBoxOut);

return;

}

}

else //обработка числа (выбор файла)

{

try

{

int num = Convert.ToInt32(command);

if (num <= dirIndex && num > 0)

{

Open(textBoxOut, num - 1, null, Form1.encodingPage);

return;

}

else if (num == 0 && PathLinker(pathIndex) != @"C:\" && PathLinker(pathIndex) != @"c:\" && PathLinker(pathIndex) != @"C:" && PathLinker(pathIndex) != @"c:") // "назад"

{

pathIndex--;

FileList(PathLinker(pathIndex), textBoxOut);

}

else

{

textBoxOut.Text += ("\r\nОшибка. Попробуйте снова.");

FileList(PathLinker(pathIndex), textBoxOut);

return;

}

}

catch

{

textBoxOut.Text += ("\r\nОшибка. Попробуйте снова.");

FileList(PathLinker(pathIndex), textBoxOut);

return;

}

}

}

catch

{

textBoxOut.Text += ("\r\nОшибка. Попробуйте снова.");

FileList(PathLinker(pathIndex), textBoxOut);

return;

}

}

private static void Open(TextBox textBoxOut, int num = -1, string p = @"C:", string codePage = "65001") //Открывает файлы и папки

{

if (num != -1) //открытие файла по числу

{

try

{

path[++pathIndex] = dirList[num];

textBoxOut.Text += ($"\r\nОткрытие {PathLinker(pathIndex)}...");

// проверяем на поддерживаемый файл для открытия

if (path[pathIndex].ToLower().EndsWith(".txt") ||

path[pathIndex].ToLower().EndsWith(".log") ||

path[pathIndex].ToLower().EndsWith(".md") ||

path[pathIndex].ToLower().EndsWith(".cs") ||

path[pathIndex].ToLower().EndsWith(".c") ||

path[pathIndex].ToLower().EndsWith(".cpp") ||

path[pathIndex].ToLower().EndsWith(".html") ||

path[pathIndex].ToLower().EndsWith(".xml") ||

path[pathIndex].ToLower().EndsWith(".fb2"))

{

OpenFile(textBoxOut, codePage);

FileList(PathLinker(--pathIndex), textBoxOut);

return;

}

else

{

FileList(PathLinker(pathIndex), textBoxOut);

}

}

catch

{

textBoxOut.Text += ($"\r\nОшибка: невозможно открыть {PathLinker(pathIndex)}");

--pathIndex;

FileList(PathLinker(pathIndex), textBoxOut);

}

}

else //открытие файла по пути

{

try

{

// сделать резервную копию "path[]" и "pathIndex" на случай ошибки открытия

pathBackup = path;

pathIndexBackup = pathIndex;

// разделить путь "р" на состовляющие и узнать новый "pathIndex"

// записать поверх путь из "р" в "path[]"

path = new string[1000];

int tempIndex = 0;

string[] temp = p.Split(new char[] { '\\' }, StringSplitOptions.RemoveEmptyEntries);

pathIndex = temp.Length - 1;

foreach (string s in temp)

{

path[tempIndex++] = s;

}

textBoxOut.Text += ($"\r\nОткрытие {PathLinker(pathIndex)}...");

// проверяем на поддерживаемый файл для открытия

if (path[pathIndex].ToLower().EndsWith(".txt") ||

path[pathIndex].ToLower().EndsWith(".log") ||

path[pathIndex].ToLower().EndsWith(".md") ||

path[pathIndex].ToLower().EndsWith(".cs") ||

path[pathIndex].ToLower().EndsWith(".c") ||

path[pathIndex].ToLower().EndsWith(".cpp") ||

path[pathIndex].ToLower().EndsWith(".html") ||

path[pathIndex].ToLower().EndsWith(".xml") ||

path[pathIndex].ToLower().EndsWith(".fb2"))

{

OpenFile(textBoxOut, codePage);

path = pathBackup;

pathIndex = pathIndexBackup;

FileList(PathLinker(pathIndex), textBoxOut);

return;

}

else

{

FileList(PathLinker(pathIndex), textBoxOut);

}

}

catch

{

textBoxOut.Text += ($"\r\nОшибка: невозможно открыть {p}");

path = pathBackup;

pathIndex = pathIndexBackup;

FileList(PathLinker(pathIndex), textBoxOut);

}

}

}

private static void OpenFile(TextBox textBoxOut, string codePage = "65001") //Выводит текстовые файлы на консоль

{

string fName = path[pathIndex];

string[] fFormat = fName.Split(new char[] { '.' }, StringSplitOptions.RemoveEmptyEntries);

string filePath = PathLinker(pathIndex).TrimEnd('\\');

// страница 1251 имеет имя "windows-1251", а страница 866 – "cp866" и, соответственно, 65001 – "UTF-8".

textBoxOut.Text += ($"\r\n\n>>>\tФайл: {fName}\t|\tКодировка: {codePage}\t<<<\r\n");

try

{

switch (fFormat.Last())

{

case "xml":

XDocument xmlBook = XDocument.Load(filePath);

textBoxOut.Text += ($"\r\n{xmlBook.Declaration}");

foreach (XNode element in xmlBook.Nodes()) OpenXML(textBoxOut, element);

break;

case "fb2":

XElement doc = XElement.Load(filePath);

//Вывод "шапки"

string lastName = null, firstName = null,

title = null, sequence = null, annotation = null,

year = null, id = null;

string[] genre = new string[10];

int genreIndex = 0;

var titleinfo = doc.Elements().First(x =>

x.Name.LocalName == "description").Elements().First(

x => x.Name.LocalName == "title-info").Elements();

foreach (var element in titleinfo)

{

switch (element.Name.LocalName)

{

case "genre": genre[genreIndex++] = element.Value; break;

case "author"://Извлекаем элементы этого узла

foreach (var el in element.Elements())

{

switch (el.Name.LocalName)

{

case "last-name": lastName = el.Value; break;

case "first-name": firstName = el.Value; break;

default: break;

}

}

break;

case "book-title": title = element.Value; break;

case "date": year = element.Value; break;

//Наименование серии – атрибут элемента sequence

case "sequence":

sequence = element.Attribute("name").Value;

break;

case "annotation": annotation = element.Value; break;

default: break;

}

}

var documentinfo = doc.Elements().First(x =>

x.Name.LocalName == "description").Elements().First(

x => x.Name.LocalName == "document-info").Elements();

foreach (var element in documentinfo)

{

switch (element.Name.LocalName)

{

case "id": id = element.Value; break;

}

}

textBoxOut.Text += ($"Тематика: ");

for (int g = 0; g < genreIndex; g++)

textBoxOut.Text += ($"{genre[g]} ");

textBoxOut.Text += ($"\r\n\nАвтор: " +

$"{lastName} {firstName}\nНазвание: {title}\n" +

$"Серия: {sequence}\nАннотация: {annotation}\nДата: {year}\nid: {id}\n");

//Вывод "тушки"

int loop = 0; //для вывода документа по частям

var body = doc.Elements().First(x =>

x.Name.LocalName == "body").Elements();

foreach (var item in body)

{

if (item.Name.LocalName == "section") //section

foreach (var el in item.Elements())

{

switch (el.Name.LocalName)

{

case "title":

foreach (var el\_title in el.Elements())

{

if (el\_title.Name.LocalName == "p")

{

textBoxOut.Text += ($"\r\n\t{el\_title.Value.ToUpper()}");

loop += (Convert.ToString(el\_title.Value)).Length;

}

}

break;

case "epigraph":

foreach (var el\_epigraph in el.Elements())

{

if (el\_epigraph.Name.LocalName == "p")

{

textBoxOut.Text += ($"\r\n{el\_epigraph.Value}");

loop += (Convert.ToString(el\_epigraph.Value)).Length;

}

}

break;

case "cite":

foreach (var el\_cite in el.Elements())

{

if (el\_cite.Name.LocalName == "p")

{

textBoxOut.Text += ($"\r\n{el\_cite.Value}");

loop += (Convert.ToString(el\_cite.Value)).Length;

}

}

break;

case "p":

textBoxOut.Text += ($"\r\n {el.Value}");

loop += (Convert.ToString(el.Value)).Length;

break;

case "empty-line":

textBoxOut.Text += ($"\r\n\n");

loop += (Convert.ToString(el.Value)).Length;

break;

case "empty-line/":

textBoxOut.Text += ($"\r\n\n");

loop += (Convert.ToString(el.Value)).Length;

break;

default: break;

}

if (loop > 1000)

{

loop = 0;

//break;

}

}

if (item.Name.LocalName == "title")

{

foreach (var el\_title in item.Elements())

{

if (el\_title.Name.LocalName == "p")

{

textBoxOut.Text += ($"\r\n\t{el\_title.Value.ToUpper()}");

loop += (Convert.ToString(el\_title.Value)).Length;

}

}

}

if (item.Name.LocalName == "epigraph")

{

foreach (var el\_epigraph in item.Elements())

{

if (el\_epigraph.Name.LocalName == "p")

{

textBoxOut.Text += ($"\r\n{el\_epigraph.Value}");

loop += (Convert.ToString(el\_epigraph.Value)).Length;

}

}

}

if (item.Name.LocalName == "cite")

{

foreach (var el\_cite in item.Elements())

{

if (el\_cite.Name.LocalName == "p")

{

textBoxOut.Text += ($"\r\n{el\_cite.Value}");

loop += (Convert.ToString(el\_cite.Value)).Length;

}

}

}

if (item.Name.LocalName == "p")

{

textBoxOut.Text += ($"\r\n{item.Value}");

loop += (Convert.ToString(item.Value)).Length;

}

if (item.Name.LocalName == "empty-line")

{

textBoxOut.Text += ($"\r\n{item.Value}");

loop += (Convert.ToString(item.Value)).Length;

}

if (item.Name.LocalName == "empty-line/")

{

textBoxOut.Text += ($"\r\n{item.Value}");

loop += (Convert.ToString(item.Value)).Length;

}

if (loop > 1000)

{

loop = 0;

//break;

}

}

break;

case "html":

XElement dochtml = XElement.Load(filePath);

var html = dochtml.Elements();

foreach (var item in html)

{

if (item.Name.LocalName == "head")

{

textBoxOut.Text += ($"Заголовок: ");

foreach (var el\_title in item.Elements())

{

if (el\_title.Name.LocalName == "title")

textBoxOut.Text += ($"\r\n{el\_title.Value}");

}

}

textBoxOut.Text += ($"\r\n\n\n");

if (item.Name.LocalName == "body")

{

foreach (var el in item.Elements())

{

switch (el.Name.LocalName)

{

case "section":

foreach (var el\_section in el.Elements())

{

if (el\_section.Name.LocalName == "p")

textBoxOut.Text += ($"\r\n {el\_section.Value}");

if (el\_section.Name.LocalName == "h1")

textBoxOut.Text += ($"\r\n\t\t{el\_section.Value.ToUpper()}");

if (el\_section.Name.LocalName == "h2")

textBoxOut.Text += ($"\r\n\t{el\_section.Value.ToUpper()}");

if (el\_section.Name.LocalName == "h3")

textBoxOut.Text += ($"\r\n\t{el\_section.Value}");

}

break;

case "table":

foreach (var el\_table in el.Elements()) // строчки

{

foreach (var el\_tr in el\_table.Elements()) // столбцы

{

textBoxOut.Text += ($"{el\_tr.Value}\t");

}

textBoxOut.Text += ($"\r\n\n");

}

break;

case "ul":

foreach (var el\_ul in el.Elements())

{

if (el\_ul.Name.LocalName == "li")

textBoxOut.Text += ($"\r\n · {el\_ul.Value}");

}

break;

case "p": textBoxOut.Text += ($"\r\n {el.Value}"); break;

case "h1": textBoxOut.Text += ($"\r\n\t\t{el.Value.ToUpper()}"); break;

case "h2": textBoxOut.Text += ($"\r\n\t{el.Value.ToUpper()}"); break;

case "h3": textBoxOut.Text += ($"\r\n\t{el.Value}"); break;

default: break;

}

}

}

if (item.Name.LocalName == "p")

textBoxOut.Text += ($"\r\n {item.Value}");

if (item.Name.LocalName == "h1")

textBoxOut.Text += ($"\r\n\t\t{item.Value.ToUpper()}");

if (item.Name.LocalName == "h2")

textBoxOut.Text += ($"\r\n\t{item.Value.ToUpper()}");

if (item.Name.LocalName == "h3")

textBoxOut.Text += ($"\r\n\t{item.Value}");

}

break;

default:

try

{

StreamReader fStr = new StreamReader(filePath, Encoding.GetEncoding(codePage));

string s;

while ((s = fStr.ReadLine()) != null) textBoxOut.Text += ($"\r\n{s}");

fStr.Close();

}

catch

{

StreamReader fStr = new StreamReader(filePath, Encoding.GetEncoding(Convert.ToInt32(codePage)));

string s;

while ((s = fStr.ReadLine()) != null) textBoxOut.Text += ($"\r\n{s}");

fStr.Close();

}

break;

}

}

catch (Exception e)

{

textBoxOut.Text += ($"\r\nОшибка открытия файла: {e.Message}");

}

}

private static void OpenXML(TextBox textBoxOut, XNode node)

{

if (node.NodeType == XmlNodeType.Comment)

{

textBoxOut.Text += ($"\r\n{node}"); return; }

XElement e = (XElement)node;

textBoxOut.Text += ($"{e.Name} : ");

if (!e.HasElements)//Если элемент не имеет дочерних элементов

{

textBoxOut.Text += ($"\r\n{e.Value}"); //вывод элемента

if (e.HasAttributes) //и его атрибутов

foreach (var at in e.Attributes()) textBoxOut.Text += ($"\r\n{at}");

}

else

{

textBoxOut.Text += ($"\r\n");

foreach (var nd in e.Nodes()) OpenXML(textBoxOut, nd);

}

}

private static void EditFile(TextBox textBoxOut, int num = -1, string p = @"C:", string codePage = "65001") //Открывает файлы для редактирования

{

if (num != -1) //открытие файла по числу

{

try

{

path[++pathIndex] = dirList[num];

textBoxOut.Text += ($"\r\nОткрытие {PathLinker(pathIndex)}...");

// проверяем на поддерживаемый файл для открытия

if (path[pathIndex].ToLower().EndsWith(".txt") ||

path[pathIndex].ToLower().EndsWith(".log") ||

path[pathIndex].ToLower().EndsWith(".md") ||

path[pathIndex].ToLower().EndsWith(".cs") ||

path[pathIndex].ToLower().EndsWith(".c") ||

path[pathIndex].ToLower().EndsWith(".cpp"))

{

textBoxOut.ReadOnly = false;

string fileName = PathLinker(pathIndex);

try // если кодировка задана текстом

{

StreamReader fStr = new StreamReader(fileName.Remove(fileName.Length - 1), Encoding.GetEncoding(codePage));

textBoxOut.Text = fStr.ReadToEnd();

fStr.Close();

}

catch // если кодировка задана числом

{

StreamReader fStr = new StreamReader(fileName.Remove(fileName.Length - 1), Encoding.GetEncoding(Convert.ToInt32(codePage)));

textBoxOut.Text = fStr.ReadToEnd();

fStr.Close();

}

--pathIndex;

}

else

{

--pathIndex;

textBoxOut.Text += ($"\r\nОшибка: невозможно открыть для редактирования {PathLinker(pathIndex)}");

FileList(PathLinker(pathIndex), textBoxOut);

}

}

catch

{

textBoxOut.Text += ($"\r\nОшибка: невозможно открыть для редактирования {PathLinker(pathIndex)}");

--pathIndex;

FileList(PathLinker(pathIndex), textBoxOut);

}

}

else //открытие файла по пути

{

try

{

// сделать резервную копию "path[]" и "pathIndex" на случай ошибки открытия

pathBackup = path;

pathIndexBackup = pathIndex;

// разделить путь "р" на состовляющие и узнать новый "pathIndex"

// записать поверх путь из "р" в "path[]"

path = new string[1000];

int tempIndex = 0;

string[] temp = p.Split(new char[] { '\\' }, StringSplitOptions.RemoveEmptyEntries);

pathIndex = temp.Length - 1;

foreach (string s in temp)

{

path[tempIndex++] = s;

}

textBoxOut.Text += ($"\r\nОткрытие {PathLinker(pathIndex)}...");

// проверяем на поддерживаемый файл для открытия

if (path[pathIndex].ToLower().EndsWith(".txt") ||

path[pathIndex].ToLower().EndsWith(".log") ||

path[pathIndex].ToLower().EndsWith(".md") ||

path[pathIndex].ToLower().EndsWith(".cs") ||

path[pathIndex].ToLower().EndsWith(".c") ||

path[pathIndex].ToLower().EndsWith(".cpp"))

{

textBoxOut.ReadOnly = false;

string fileName = PathLinker(pathIndex);

try // если кодировка задана текстом

{

StreamReader fStr = new StreamReader(fileName.Remove(fileName.Length - 1), Encoding.GetEncoding(codePage));

textBoxOut.Text = fStr.ReadToEnd();

fStr.Close();

}

catch // если кодировка задана числом

{

StreamReader fStr = new StreamReader(fileName.Remove(fileName.Length - 1), Encoding.GetEncoding(Convert.ToInt32(codePage)));

textBoxOut.Text = fStr.ReadToEnd();

fStr.Close();

}

path = pathBackup;

pathIndex = pathIndexBackup;

return;

}

else

{

textBoxOut.Text += ($"\r\nОшибка: невозможно открыть для редактирования {p}");

FileList(PathLinker(pathIndex), textBoxOut);

}

}

catch

{

textBoxOut.Text += ($"\r\nОшибка: невозможно открыть для редактирования {p}");

path = pathBackup;

pathIndex = pathIndexBackup;

FileList(PathLinker(pathIndex), textBoxOut);

}

}

}

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new Form1());

}

}

}

* 1. Form1.cs

using System;

using System.Windows.Forms;

using System.Text;

using System.IO;

namespace WindowsFormsApp2

{

public partial class Form1 : Form

{

private static string command; // хранит прописанную с командной строки команду

private string FileName { get; set; } = string.Empty;

public static string encodingPage = string.Empty;

public Form1()

{

InitializeComponent();

Program.path[0] = @"C:"; //начальный каталог

Program.Intro(textBoxOut);

}

private void buttonRun\_Click(object sender, EventArgs e) // Выполнить

{

textBoxOut.ReadOnly = true;

command = textBoxIn.Text;

textBoxOut.Clear();

textBoxIn.Clear();

encodingPage = (string)code.Items[code.SelectedIndex];

Program.Request(command, textBoxOut);

}

private void textBoxIn\_KeyPress(object sender, KeyPressEventArgs e) // обработка нажатия клавиши Enter для исполнения прописанной команды

{

if (e.KeyChar == Convert.ToChar(Keys.Return))

{

e.Handled = true; // убираем системный звук при нажатии Enter

buttonRun\_Click(sender, e);

}

}

private void buttonBack\_Click(object sender, EventArgs e) // Назад

{

textBoxOut.ReadOnly = true;

command = "0";

textBoxOut.Clear();

textBoxIn.Clear();

Program.Request(command, textBoxOut);

}

private void buttonTop\_Click(object sender, EventArgs e) // Корневой каталог

{

textBoxOut.ReadOnly = true;

command = "-top";

textBoxOut.Clear();

textBoxIn.Clear();

Program.Request(command, textBoxOut);

}

private void buttonCur\_Click(object sender, EventArgs e) // Текущий каталог

{

textBoxOut.ReadOnly = true;

command = "-cur";

textBoxOut.Clear();

textBoxIn.Clear();

Program.Request(command, textBoxOut);

}

private void справкаToolStripMenuItem\_Click(object sender, EventArgs e)

{

textBoxOut.ReadOnly = true;

command = "-info";

textBoxOut.Clear();

textBoxIn.Clear();

Program.Request(command, textBoxOut);

}

private void newFile(object sender, EventArgs e)

{

textBoxOut.Text = String.Empty;

textBoxOut.ReadOnly = false;

}

private void openFile(object sender, EventArgs e)

{

OpenFileDialog openFileDialog = new OpenFileDialog();

openFileDialog.InitialDirectory = Environment.CurrentDirectory;

openFileDialog.Filter = "Текстовые файлы (\*.txt)|\*.txt|" + "Все файлы (\*.\*)|\*.\*";

openFileDialog.RestoreDirectory = true;

if (openFileDialog.ShowDialog(this) == DialogResult.OK)

{

textBoxOut.ReadOnly = false;

FileName = openFileDialog.FileName;

encodingPage = (string)code.Items[code.SelectedIndex];

StreamReader fStr = new StreamReader(FileName, Encoding.GetEncoding(encodingPage));

textBoxOut.Text = fStr.ReadToEnd();

fStr.Close();

}

}

private void saveFile(object sender, EventArgs e)

{

SaveFileDialog saveFileDialog = new SaveFileDialog();

saveFileDialog.Filter = "Текстовые файлы (\*.txt)|\*.txt|" + "Все файлы (\*.\*)|\*.\*";

try

{

if (saveFileDialog.ShowDialog(this) == DialogResult.OK)

{

FileName = saveFileDialog.FileName;

}

encodingPage = (string)code.Items[code.SelectedIndex];

StreamWriter fStr = new StreamWriter(FileName, false, Encoding.GetEncoding(encodingPage));

fStr.Write(textBoxOut.Text);

fStr.Close();

}

catch { }

}

private void exit(object sender, EventArgs e)

{

textBoxOut.ReadOnly = true;

this.Close();

}

private void openFileList(object sender, EventArgs e)

{

textBoxOut.ReadOnly = true;

textBoxOut.Text = String.Empty;

Program.FileList(Program.PathLinker(Program.pathIndex), textBoxOut);

}

}

}