1. Базовые действия с документом: создание, открытие, сохранение

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.Reflection;
// Требуется подключение сборки
// Microsoft.Office.Interop.Word версии 12

namespace Microsoft.Office.Interop.Word
{

    public class MSWord : IDisposable
    {
        public Document wd;
        Application wa;
        string fname;

        static string FullName(string filename)
        {
            return Path.GetFullPath(filename);
        }

        static string DummyName()
        {
            return Directory.GetCurrentDirectory() + "\\$$Dummy$$.doc";
        }

        public string FileName
        {
            get { return fname; }
        }

        public Boolean Visible
        {
            set { wa.Visible = value; }
            get { return wa.Visible; }
        }

        public void SaveAs(string filename)
        {
            fname = FullName(filename);
            Object fileName0 = fname;
            Object fileFormat =
                Path.GetExtension(filename).ToLower() == ".docx" ?
                Word.WdSaveFormat.wdFormatXMLDocument :
                Word.WdSaveFormat.wdFormatDocument;
            Object lockComments = false;
            Object password = "";
            Object addToRecentFiles = false;
            Object writePassword = "";
            Object readOnlyRecommended = false;
            Object embedTrueTypeFonts = false;
            Object saveNativePictureFormat = false;
            Object saveFormsData = false;
```

```
            Object saveAsAOCELetter = Type.Missing;
            Object encoding = Type.Missing;
            Object insertLineBreaks = Type.Missing;
            Object allowSubstitutions = Type.Missing;
            Object lineEnding = Type.Missing;
            Object addBiDiMarks = Type.Missing;
            wd.SaveAs(ref fileName0,
                ref fileFormat, ref lockComments,
                ref password, ref addToRecentFiles, ref writePassword,
                ref readOnlyRecommended, ref embedTrueTypeFonts,
                ref saveNativePictureFormat, ref saveFormsData,
                ref saveAsAOCELetter, ref encoding, ref insertLineBreaks,
                ref allowSubstitutions, ref lineEnding, ref addBiDiMarks);
        }

        public void Save()
        {
            if (!wd.Saved)
                SaveAs(fname);
        }

        void Load0(string filename)
        {
            fname = FullName(filename);
            if (!File.Exists(fname))
            {
                Object template = Type.Missing;
                Object newTemplate = false;
                Object documentType = Word.WdNewDocumentType.wdNewBlankDocument;
                Object visible = false;
                wd = wa.Documents.Add(
                    ref template, ref newTemplate, ref documentType, ref visible);
            }
            else
            {
                Object filename0 = fname;
                Object confirmConversions = true;
                Object readOnly = false;
                Object addToRecentFiles = true;
                Object passwordDocument = Type.Missing;
                Object passwordTemplate = Type.Missing;
                Object revert = false;
                Object writePasswordDocument = Type.Missing;
                Object writePasswordTemplate = Type.Missing;
                Object format = Type.Missing;
                Object encoding = Type.Missing; ;
                Object oVisible = Type.Missing;
                Object openConflictDocument = Type.Missing;
                Object openAndRepair = Type.Missing;
                Object documentDirection = Type.Missing;
                Object noEncodingDialog = false;
                Object xmlTransform = Type.Missing;
                wd = wa.Documents.Open(ref filename0,
                    ref confirmConversions, ref readOnly, ref addToRecentFiles,
                    ref passwordDocument, ref passwordTemplate, ref revert,
                    ref writePasswordDocument, ref writePasswordTemplate,
                    ref format, ref encoding, ref oVisible,
                    ref openAndRepair, ref documentDirection, ref noEncodingDialog,
                    ref xmlTransform);
            }
        }
```

```
        public void Load(string filename, bool savePreviousDoc)
        {
            if (wd != null)
            {
                if (savePreviousDoc)
                    Save();
                Object saveChanges = WdSaveOptions.wdDoNotSaveChanges;
                wd.Close(ref saveChanges);
                wd = null;
            }
            Load0(filename);
        }

        public void Load(string filename) => Load(filename, fname != DummyName());

        public void Load(bool savePreviousDoc) => Load(DummyName(), savePreviousDoc);

        public void Load() => Load(DummyName());

        public MSWord(string filename, bool visible = false)
        {
            wa = new Application();
            Load0(filename);
            Visible = visible;
        }

        public MSWord(bool visible = false) : this(DummyName(), visible) { }

        public void Dispose()
        {
            Dispose(true);
            GC.SuppressFinalize(this);
        }

        protected virtual void Dispose(bool disposeIsCalled)
        {
            try
            {
                if (fname != DummyName())
                    Save();
                wd.Close();
                object saveChanges = Word.WdSaveOptions.wdDoNotSaveChanges;
                object originalFormat = Word.WdOriginalFormat.wdWordDocument;
                object routeDocument = Type.Missing;
                wa.Quit(ref saveChanges, ref originalFormat, ref routeDocument);
            }
            catch
            { }
        }

        ~MSWord()
        {
            Dispose(false);
        }

    }
}
```

```
static void Main(string[] args)
      {
          MSWord w = new MSWord();
          for (int i = 1; i < 5; ++i)
              w.SaveAs(i + ".doc");
          Console.WriteLine("OK");
          Console.ReadLine();
      }
```

Будут созданы пустые файлы 1.doc, 2.doc, 3.doc, 4.doc.

---

2. Выделение фрагментов текста

```
public const string ParMark = "\r";
public const string CellMark = "\u0007";
public const string TabMark = "\t";
public const string LineBreak = "\u000B";
public const string PageBreak = "\u000C";
public const string ColBreak = "\u000E";
public const string NBSp = "\u00A0";
public const string LDash = "\u2014";
public const string MDash = "\u2013";
public const string LQuot = "\u00AB";
public const string RQuot = "\u00BB";

public Selection SelectRange(int pos, int count)
{
    Object a1 = pos;
    Object a2 = pos + count;
    wd.Range(ref a1, ref a2).Select();
    return wa.Selection;
}

public Selection SelectPos(int pos) => SelectRange(pos, 0);

public int LastPos
{
    get { return wd.Range().End - 1; }
}

public Selection Selection
{
    get => wa.Selection;
}

// Свойства SelLength, SelStart и SelText
//   могут изменяться только в том случае, если
//   в текущем выделении НЕ СОДЕРЖИТСЯ символ CellMark
//   (иными словами, изменения допускаются только
//   в пределах одной ячейки таблицы)
// Кроме того, свойство SelLength в случае,
//   если в выделении СОДЕРЖИТСЯ символ CellMark,
//   всегда возвращает особое значение -1.

public int SelLength
{
    get => SelText.Contains(CellMark) ? -1 :
              Selection.End - Selection.Start;
```

```
            set
            {
                if (SelText.Contains(CellMark))
                    return;
                if (value < 0)
                    value = 0;
                Selection.End = Selection.Start + value;
            }
        }

        public int SelStart
        {
            get => Selection.Start;
            set
            {
                if (SelText.Contains(CellMark))
                    return;
                if (value < 0)
                    value = 0;
                int len = SelLength;
                Selection.Start = value;
                SelLength = len;
            }
        }

        // Особая обработка двух случаев:
        // 1) если длина выделения равна 0, то возвращается
        //      пустая строка (стандартный Text возвращает
        //      ОДИН символ за позицией курсора)
        // 2) если вставляется текст в конец документа
        //      и он оканчивается символом абзаца,
        //      то перед вставкой этот символ убирается
        //      (в противном случае добавляется НА ОДИН СИМВОЛ
        //      АБЗАЦА БОЛЬШЕ)

        public string SelText
        {
            get => Selection.End - Selection.Start == 0 || Selection.Text == null ?
                    "" : Selection.Text;
            set
            {
                if (SelText.Contains(CellMark))
                    return;
                if (value != "" && value[value.Length - 1] == ParMark[0]
                    && wd.Content.End == Selection.End)
                    value = value.Remove(value.Length - 1);
                Selection.Text = value;
            }
        }

        public Selection ToSelStart()
        {
            if (SelLength != -1)
                SelLength = 0;
            return Selection;
        }

        public Selection ToSelEnd()
        {
            if (SelLength != -1)
            {
```

```
            int l = SelLength;
            SelLength = 0;
            SelStart += l;
        }
        return Selection;
    }
```

В конец метода Load0 добавить оператор:

```
        SelectPos(0);
```

```
static void Main(string[] args)
    {
        MSWord w = new MSWord();
        for (int i = 1; i < 5; ++i)
        {
            w.Load(i + ".doc");
            w.SelText = "Пример " + i;
        }
        Console.WriteLine("OK");
        Console.ReadLine();
    }
```

В файлы будет добавлен указанный текст.

```
    static void Main(string[] args)
    {
        MSWord w = new MSWord();
        for (int i = 1; i < 5; ++i)
        {
            w.Load(i + ".doc");
            w.SelLength = 6;
            w.SelText = "Example";
        }
        Console.WriteLine("OK");
        Console.ReadLine();
    }
```

Текст «Пример» в файлах будет заменен на «Example».

---

3. Работа с абзацами

```
    public int ParCount
    {
        get => wd.Paragraphs.Count;
    }

    public Selection SelectPars(int start, int count)
    {
        if (start < 0)
            start = 0;
        if (start >= ParCount)
            start = ParCount - 1;
        if (start < 0)
```

```
            throw new ArgumentOutOfRangeException();
        if (count < 1)
            count = 1;
        if (start + count - 1 >= ParCount)
            count = ParCount - start;

        Object a1 = wd.Paragraphs[start + 1].Range.Start;
        Object a2 = wd.Paragraphs[start + count].Range.End;
        wd.Range(ref a1, ref a2).Select();
        return wa.Selection;

    }

    public Selection SelectPars(int start) => SelectPars(start, ParCount);

    public Selection SelectAll() => SelectPars(0, ParCount);

    public Selection SelectPar(int start) => SelectPars(start, 1);

    int GetParIndexFromStartRec(int start, int a, int b, int sa, int sb)
    {
        // бинарный поиск
        int c = (a + b) / 2;
        if (c == a)
            return a - 1;
        if (c == b)
            return b - 1;
        int sc = wd.Paragraphs[c].Range.Start;
        if (sc == start)
            return c - 1;
        if (sc < start)
            return GetParIndexFromStartRec(start, c, b, sc, sb);
        else
            return GetParIndexFromStartRec(start, a, c, sa, sc);
    }

    public int PosToPar(int start)
    {
        if (start <= 0)
            return 0;
        int b = ParCount;
        int sb = wd.Paragraphs[b].Range.Start;
        if (start >= sb)
            return b - 1;
        return GetParIndexFromStartRec(start, 1, b, 0, sb);
    }

    public int SelStartToPar() => PosToPar(Selection.Start);

    public int SelEndToPar() => PosToPar(Selection.End - 1);

    public Selection ExpandSelection()
    {
        int p1 = SelStartToPar();
        int p2 = SelEndToPar();
        return SelectPars(p1, p2 - p1 + 1);
    }

    public Selection RemovePar(int start)
    {
        SelectPar(start);
```

```
            SelText = "";
            return Selection;
        }

        public Selection RemovePars(int start, int count)
        {
            SelectPars(start, count);
            SelText = "";
            return Selection;
        }

        public Selection InsertPars(IEnumerable<string> lines)
        {
            if (SelLength != -1)
            {
                string text = "";
                foreach (string e in lines)
                {
                    if (e != null)
                        text += e;
                    text += ParMark;
                }
                SelText = text;
            }
            return Selection;
        }

        public Selection InsertPars(params string[] lines) =>
            InsertPars(lines.AsEnumerable());

        public Selection InsertPars(int count) =>
            InsertPars(Enumerable.Repeat<string>(null, count));

        public Selection InsertPar() => InsertPars(1);

        public Selection AddPars(IEnumerable<string> lines)
        {
            SelectPos(wd.Range().End - 1);
            return InsertPars(lines);
        }

        public Selection AddPars(params string[] lines) =>
            AddPars(lines.AsEnumerable());

        public Selection AddPars(int count) =>
            AddPars(Enumerable.Repeat<string>(null, count));

        public Selection AddPar() => AddPars(1);
```

```
        static void Main(string[] args)
        {
            MSWord w = new MSWord("b.doc");
            w.AddPars(Enumerable.Range(1, 10).Select(e => "Абзац " + e));
            Console.WriteLine("OK");
            Console.ReadLine();
        }
```

При каждом запуске программы в файл b.doc добавляются 10 абзацев с текстом «Абзац …».

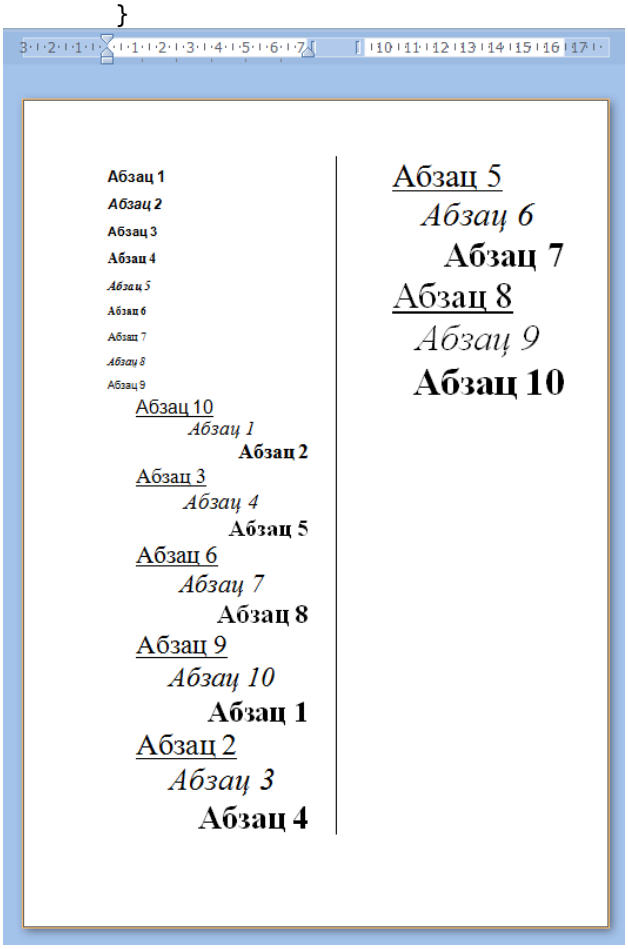
---

4. Форматирование и работа с буфером обмена

```
        public void ParAlign(int alignment0_3, double mmFirst, double mmLeft,
          double mmRight, int ptTop, int ptBottom, int lineSpacing0_2)
        {
            switch (alignment0_3)
            {
                case 0:
                    Selection.ParagraphFormat.Alignment =
                      WdParagraphAlignment.wdAlignParagraphLeft;
                    break;
                case 1:
                    Selection.ParagraphFormat.Alignment =
                      WdParagraphAlignment.wdAlignParagraphCenter;
                    break;
                case 2:
                    Selection.ParagraphFormat.Alignment =
                      WdParagraphAlignment.wdAlignParagraphRight;
                    break;
                case 3:
                    Selection.ParagraphFormat.Alignment =
                      WdParagraphAlignment.wdAlignParagraphJustify;
                    break;
            }
            switch (lineSpacing0_2)
            {
                case 0:
                    Selection.ParagraphFormat.LineSpacingRule =
                      WdLineSpacing.wdLineSpaceSingle;
                    break;
                case 1:
                    Selection.ParagraphFormat.LineSpacingRule =
                      WdLineSpacing.wdLineSpace1pt5;
                    break;
                case 2:
                    Selection.ParagraphFormat.LineSpacingRule =
                      WdLineSpacing.wdLineSpaceDouble;
                    break;
            }
            Selection.ParagraphFormat.FirstLineIndent =
              wa.MillimetersToPoints((float)mmFirst);
            Selection.ParagraphFormat.LeftIndent = wa.MillimetersToPoints((float)mmLeft);
            Selection.ParagraphFormat.RightIndent = wa.MillimetersToPoints((float)mmRight);
            Selection.ParagraphFormat.SpaceBefore = ptTop;
            Selection.ParagraphFormat.SpaceAfter = ptBottom;
        }

        public void ParAlign(int alignment0_3, double mmFirst, double mmLeft, double
mmRight) =>
            ParAlign(alignment0_3, mmFirst, mmLeft, mmRight, 0, 0, 0);

        public void ParAlign(int alignment0_3, double mmFirst) =>
            ParAlign(alignment0_3, mmFirst, 0, 0, 0, 0, 0);

        public void ParStyle(string style)
        {
            Object s = style;
            Object s1 = wd.Styles[ref s];
            Selection.set_Style(ref s);
        }
```

```
        public void ParNormal() => ParStyle("Обычный");

        public void ParHeader(int level1_9)
        {
            if (level1_9 < 1)
                level1_9 = 1;
            if (level1_9 > 9)
                level1_9 = 9;
            ParStyle("Заголовок " + level1_9);
        }

        public void Columns(int count, double mmSpacing, bool lineBetween)
        {
            wd.PageSetup.TextColumns.SetCount(count);
            wd.PageSetup.TextColumns.Spacing = wa.MillimetersToPoints((float)mmSpacing);
            wd.PageSetup.TextColumns.LineBetween = lineBetween ? -1 : 0;
        }

        public void Copy() => Selection.Range.Copy();

        public void Cut() => Selection.Range.Cut();

        public void Paste()
        {
            try
            {
                Selection.Range.Paste();
            }
            catch
            { }
        }

        public int SelEndPageNumber
        {
            get
            {
                return (int)Selection.Information[WdInformation.wdActiveEndPageNumber];
                // вариант, основанный на технологии Reflection:
                Object o = wa.Selection;
                Type type = o.GetType();
                object oProp = type.InvokeMember("Information",
                                        BindingFlags.Default | BindingFlags.GetProperty,
                                        null, o,
                                        new object[] { WdInformation.wdActiveEndPageNumber });
                return (int)oProp;
            }
        }

        public int SelEndAdjPageNumber
        {
            // отличается от предыдущего тем, что учитывает настройки
            // пользователя (например, первая страница может иметь номер 3)
            get
            {
                return
                  (int)Selection.Information[WdInformation.wdActiveEndAdjustedPageNumber];
            }
        }
```

```
static void Main(string[] args)
{
    MSWord w = new MSWord("b.doc");
    w.Columns(2, 20, true);
    for (int i = 0; i < w.ParCount; ++i)
    {
        w.SelectPar(i);
        w.Selection.Font.Size = 10 + i;
        switch (i % 3)
        {
            case 0:
                w.Selection.Font.Underline = WdUnderline.wdUnderlineSingle;
                break;
            case 1:
                w.Selection.Font.Italic = 1;
                break;
            case 2:
                w.Selection.Font.Bold = 1;
                break;
        }
        if (i < 9)
            w.ParHeader(i+1);
        else
            w.ParAlign(i % 3, 10);
    }
    Console.WriteLine(w.SelEndPageNumber);
    Console.WriteLine("OK");
    Console.ReadLine();
}
```



5. Дополнительные возможности

5.1. Доступ к полному тексту документа

```
// Свойство AllText специально предназначено для поиска в пределах всего документа
// В нем учитывается, что вставленные поля (в том числе изображения и гиперссылки)
// имеют диапазон, не соответствующий их реальным вхождениям в свойство Text.
// Для обеспечения соответствия удаляются метки ячеек таблицы и
// дополняются специальными символами все вхождения полей.
// После всех преобразований проверяется, что теперь строка имеет корректный
// размер (соответствующий значению Range().End. Если это не так,
// возбуждается исключение.

struct InsData
{
    public static char repl = '\u2000';
    public int start;
    public int len;
    public InsData(int s, int l)
    {
        start = s;
        len = l;
    }
}

public class NotSupportedDocumentException : Exception
{
    public NotSupportedDocumentException() : base()
    { }
    public NotSupportedDocumentException(string message) : base(message)
    { }
}

public string AllText
{
    get
    {
        string s = TryAllText;
        if (s == null)
            throw new NotSupportedDocumentException
                ("Особенности документа не позволяют использовать для него" +
                " свойство AllText");
        return s;
    }
}

public string TryAllText
{
    get
    {
        foreach (Field f in wd.Fields)
            f.ShowCodes = false;
        List<InsData> id = new List<InsData>();
        string s = wd.Range().Text.Replace(MSWord.CellMark, "");
        for (int i = 1; i <= wd.OMaths.Count; i++)
        {
            Range h = wd.OMaths[i].Range;
            int p = h.Start - 1;
            id.Add(new InsData(p, 1));
        }
```

```
            for (int i = 1; i <= wd.Shapes.Count; i++)
            {
                Range h = wd.Shapes[i].Anchor;
                int p = h.Start;
                id.Add(new InsData(p, h.End - p));
            }
            for (int i = 1; i <= wd.Fields.Count; i++)
            {
                Field h = wd.Fields[i];
                int p = h.Code.Start - 1;
                if (h.Result.End == h.Result.Start)
                    id.Add(new InsData(p, h.Code.End + 1 - p));
                else
                    id.Add(new InsData(p, h.Code.End + 2 - p));
            }
            foreach (InsData e in id.OrderBy(e1 => e1.start))
                s = s.Insert(e.start, new string(InsData.repl, e.len));
            if (s.Length != wd.Range().End)
                s = null;
            return s;
        }
    }
```

5.2. Преобразование таблиц в текст и обратно

```
    public int TableCount
    {
        get => wd.Tables.Count;
    }

    public Table TextToTable(int rows, int cols, bool borders)
    {
        if (SelLength != -1)
        {
            Object separator = WdTableFieldSeparator.wdSeparateByTabs;

            Object r = rows;
            if (rows == 0)
                r = Type.Missing;
            Object c = cols;
            if (cols == 0)
                c = Type.Missing;
            Table t = Selection.Range.ConvertToTable(ref separator,
                ref r, ref c);
            t.Range.Select();
            object s = "Сетка таблицы";
            if (borders)
                Selection.set_Style(ref s);
            return t;
        }
        return null;
    }

    public Table TextToTable(int rows, int cols) => TextToTable(rows, cols, true);

    public Table TextToTable(int cols) => TextToTable(0, cols, true);

    public Table TextToTable(bool borders) => TextToTable(0, 0, borders);

    public Table TextToTable() => TextToTable(0, 0, true);
```

```
public Selection TableToText(Table table)
{
    Object separator = WdTableFieldSeparator.wdSeparateByTabs;
    table.ConvertToText(ref separator).Select();
    return Selection;
}

public Selection TableToText(int tableInd)
{
    if (tableInd < 0)
        tableInd = 0;
    if (tableInd >= TableCount)
        tableInd = TableCount - 1;
    if (tableInd < 0)
        throw new ArgumentOutOfRangeException();
    return TableToText(wd.Tables[tableInd + 1]);
}
```

5.3. Добавление гиперссылок

```
public void AddHyperlink(string address, string screentip = "")
{
    Object sel = Selection.Range;
    Object addr = address;
    Object subaddr = Type.Missing;
    Object tip = screentip;
    Object display = SelText;
    wd.Hyperlinks.Add(sel, ref addr,
        ref subaddr, ref tip, ref display);
}
```

6. Программа MSWordFind

Основные возможности

Программа MSWordFind предназначена для поиска фрагментов текста в наборе doc-файлов. Перечислим ее основные особенности:

- возможность организации поиска во всех файлах, входящих в некоторый каталог и его подкаталоги (или в части файлов, имя которых удовлетворяет указанной маске);
- одновременный поиск нескольких требуемых фрагментов;
- возможность поиска с применением и без применения механизма регулярных выражений;
- оформление результатов поиска в виде doc-файла, содержащего все найденные фрагменты и позволяющего быстро загрузить любой из файлов с найденными фрагментами.

При запуске программа MSWordFind ищет в текущем каталоге текстовый файл с расширением fdat (файл должен иметь однобайтную кодировку Windows). Если такой файл единственный, то он обрабатывается, если файлов с расширением fdat несколько, то выводится список этих файлов и предлагается указать номер того, который надо обработать. Можно также запускать программу MSWordFind с параметрами командной строки; каждый параметр определяет файл с настройками синхронизации (в этом случае файл с настройками может иметь и другие расширения). При указании параметров командной строки можно указывать полный путь к файлу и допустимо не указывать расширение; по умолчанию будет использовано расширение fdat.

Формат файла fdat и порядок работы программы

Файл fdat может содержать необязательную часть (комментарии, общие настройки поиска) и обязательную часть (список шаблонов поиска).

Начальные и конечные пробелы в строках файла fdat не учитываются.

Общие настройки указываются в виде «ключ=значение»; в версии 1.0 имеются 4 вида настроек со следующими ключами:

- updir — имя каталога, в котором выполняется поиск (обрабатываются также файлы в его подкаталогах);
- mask — маска файлов, в которых выполняется поиск (расширение в маске можно не указывать; при его наличии оно не учитывается, так как всегда обрабатываются файлы с расширениями doc и docx);
- detailcount — количество детализированных описаний найденных вхождений, которые заносятся в итоговый doc-файл; должно быть неотрицательным;
- pagecount — максимальное количество страниц, занимаемое найденным вхождением, при котором его текст записывается в итоговый файл; должно быть не меньше 2.

Если настройка updir не указана, то поиск выполняется в текущем каталоге (и его подкаталогах), если не указана маска mask, то она по умолчанию считается равной «*», т. е. охватывает все возможные имена файлов с расширениями doc и docx. Значение настройки detailcount по умолчанию равно 10, а настройки pagecount — 2.

Шаблоны поиска задаются в одном из двух видов:

```
=шаблон, не использующий регулярные выражения
-шаблон, использующий регулярные выражения
```

Таким образом, если строка начинается с символа «=», то она определяет шаблон поиска *в точности* того текста, который указан, если строка начинается с символа «-», то в шаблоне могут использоваться команды языка регулярных выражений.

При использовании варианта шаблона без регулярных выражений имеется дополнительная возможность указания нескольких вариантов поиска; для этого варианты должны разделяться символами @, например:

```
=ОК-@ОПК-@ПК-
```

Данный вариант будет искать варианты текста «ОК-», «ОПК-» и «ПК-».

Замечание. Если необходимо включить в текст шаблона поиска символ @, то его надо заключить в фигурные скобки: {@}.

Заметим, что поиск этих же вариантов можно организовать, указав три различных шаблона:

```
=ОК-
=ОПК-
=ПК-
```

Однако в случае трех шаблонов информация о результатах поиска также будет оформлена в виде трех разделов, каждый из которых будет содержать результаты поиска для одного из шаблонов. В случае указания комбинированного шаблона «ОК-@ОПК-@ПК-» все результаты будут выведены в одном общем разделе. Кроме того, в случае комбинированного шаблона при обнаружении фрагмента «ОПК-» данный фрагмент уже не будет анализироваться повторно для последующих вариантов шаблона и, таким образом, для него не будет обнаружено «лишнее» вхождение фрагмента «ПК-».

Поиск этих же фрагментов с применением средств языка регулярных выражений может быть организован с помощью следующей настройки:

```
=ОК-|О?ПК-
```

Здесь были использованы две команды языка регулярных выражений: |, означающая выбор из нескольких вариантов (полный аналог символа @ для шаблона, не использующего регулярные выражения), и ? — квантификатор, означающий, что предыдущий символ может встретиться 0 или 1 раз.

Программа проверяет правильность указанных регулярных выражений, и при обнаружении ошибок выводит об этом сообщение и немедленно завершает работу.

Кроме того, программа немедленно завершает работу, если указанный каталог для анализа файлов не существует или не содержит файлы, удовлетворяющие указанной маске.

Если требуемые файлы обнаружены, то они последовательно анализируются. Порядок их анализа соответствует алфавитному порядку их имен. Программа выводит в консольное окно имя анализируемого файла и количество найденных в нем вхождений для каждого шаблона поиска. После завершения обработки всех файлов выводится итоговая информация: количество обработанных файлов и общее число найденных вхождений для каждого шаблона поиска, например:

```
MSWordFind 1.0

Анализ файла Абрамян_М\b16-ANN-CS321_Параллельное_и_многопоточное_программирование(Абрамян_М).doc: 1
Анализ файла Абрамян_М\b16-FOS-CS321_Параллельное_и_многопоточное_программирование(Абрамян_М).doc: 4
Анализ файла Абрамян_М\b16-RPD-CS321_Параллельное_и_многопоточное_программирование(Абрамян_М).doc: 1
Анализ файла Абрамян_М\b16-UKD-CS321_Параллельное_и_многопоточное_программирование(Абрамян_М).doc: 0

Просмотрено файлов: 4
Найдено вхождений шаблона 1: 6
Подробная информация о результатах поиска приведена в файле
D:\Generators\demo2\$find\fiit-found.doc

Для завершения программы нажмите Enter.
```

Замечание 1. В течение работы программы MSWordFind не следует выполнять редактирование других файлов в редакторе Word.

Замечание 2. В некоторых, достаточно редких, случаях программе MSWordFind не удается выполнить поиск в указанном файле. Это происходит из-за наличия в файле сложных компонентов, например, *вложенных* объектов-полей (fields), или *полотна* (canvas), вставленного в текст *без режима обтекания*, или формул, содержащих многострочные матрицы. В этих случаях программа выводит после имени файла не количество найденных вхождений, а символ «?». Кроме того, в файл с итоговой информацией добавляется соответствующее сообщение, например:

Файл: **Кряквин_ВД\b14-FOS-Цифровая_обработка_сигналов(Кряквин_В).doc**
Особенности документа не позволили организовать в нем полнотекстовый поиск.

Отметим, что просмотреть объекты-поля, содержащиеся в документе, можно, нажав комбинацию Alt+F9 (при этом объекты отображаются в фигурных скобках). Вложенные объекты, как правило, возникают в результате ошибочных действий. Что касается полотна, используемого обычно для размещения на нем графических элементов, то для него рекомендуется устанавливать *режим обтекания* «Вокруг рамки». Обычные рисунки (размещенные в любом режиме), полотно, размещенное в режиме обтекания, «обычные» (не вложенные) поля, а также однострочные формулы обрабатываются программой корректно.

Файл с итоговой информацией

Подробная информация о найденных вхождениях сохраняется в doc-файле, имя которого получается из имени fdat-файла добавлением суффикса «-found». Например, если был использован файл fiit.fdat, то информация о результатах поиска будет сохранена в файле

fiit-found.doc. Заметим, что при поиске программа MSWordFind не обрабатывает doc-файлы с префиксом «-found».

В качестве примера приведем содержимое файла с результатами поиска приведенного выше шаблона в каталоге $sourcefiles, содержащем данные о единственной РПД:

Шаблон поиска (без применения механизма регулярных выражений):
ОК-@ОПК-@ПК-
Каталог верхнего уровня:
D:\Generators\demo2\$sourcefiles
Файл: **Абрамян_М\b16-ANN-CS321_Параллельное_и_многопоточное_программирование(Абрамян_М).doc**
Количество найденных вхождений: 1
(1) Стр. 2:

- ПК-3: способность использовать современные инструментальные и вычислительные средства

Файл: **Абрамян_М\b16-FOS-CS321_Параллельное_и_многопоточное_программирование(Абрамян_М).doc**
Количество найденных вхождений: 4
(1) Стр. 1:

ПК-3

(2) Стр. 1:

ПК-3

(3) Стр. 1:

ПК-3

(4) Стр. 1:

ПК-3

Файл: **Абрамян_М\b16-RPD-CS321_Параллельное_и_многопоточное_программирование(Абрамян_М).doc**
Количество найденных вхождений: 1
(1) Стр. 2:
ПК-3: способность использовать современные инструментальные и вычислительные средства

В начальной части документа указывается шаблон поиска и каталог верхнего уровня. Затем перечисляются обработанные файлы, в которых были найдены вхождения данного шаблона. После имени файла указывается количество найденных вхождений, а затем приводятся фрагменты с найденными вхождениями, причем текст вхождения в них выделяется зеленым цветом. Перед каждым фрагментом указывается порядковый номер найденного вхождения и номер (или диапазон номеров) страниц, на которых данный фрагмент располагается. Фрагмент всегда представляет собой один или несколько полных абзацев.

Если фрагмент представляет собой ячейку таблицы, то он выводится также в виде ячейки таблицы.

Имена обработанных файлов, указанные в сформированном документе, представляют собой гиперссылки, позволяющие быстро открыть требуемый файл в редакторе Word (для этого достаточно щелкнуть на гиперссылке мышью, держа нажатой клавишу Ctrl).

Если в fdat-файле указано несколько шаблонов поиска, то информация о каждом шаблоне приводится в итоговом dat-файле с новой страницы.

Подстановочные команды

Для указания в fdat-файлах некоторых специальных символов можно использовать подстановочные команды:

~ символ неразрывного пробела;
<< символ открывающей угловой кавычки «;
>> символ закрывающей угловой кавычки »;
-- символ короткого тире –;
--- символ длинного тире —.

Если текст содержит последовательности символов, совпадающие с подстановочными командами, то эти последовательности надо заключать в фигурные скобки. Например, для вставки текста «a >> b» необходимо представить его в следующем виде: «a {>>} b». Заметим, что преобразование всех подобных команд в соответствующие символы выполняется до запуска механизма поиска, поэтому данные команды не будут влиять на разбор регулярных выражений.

При определении шаблона поиска без использования регулярных выражений можно также указывать подстановочную команду |, означающую символ табуляции. В шаблонах поиска с применением регулярных выражений символ | имеет другой смысл: это операция, задающая варианты поиска. Символ табуляции в регулярном выражении можно указать с помощью команды \t.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using System.IO;
using Microsoft.Office.Interop.Word;

namespace MSWordGen
{
    class Program
    {

        static bool ErrorsFound = false;
        static bool ShowWord = false; // для отладочных целей = true

        static void Err(string s)
        {
            Console.WriteLine("Error: " + s);
            ErrorsFound = true;
        }

        static MSWord w, w1;
        static MSWord[] w0;
        static string[] w0name;
        static int[] w0count;
        static int totalcount;

        static string ProcessStr(string s)
        {
            return s
                .Replace("{~}", "\u9990").Replace("{<<}", "\u9991")
```

```
            .Replace("{>>}", "\u9992")
            .Replace("{---}", "\u9994").Replace("{--}", "\u9995")

            .Replace("~", MSWord.NBSp).Replace("<<", MSWord.LQuot)
            .Replace(">>", MSWord.RQuot)
            .Replace("---", MSWord.LDash).Replace("--", MSWord.MDash)

            .Replace("\u9990", "~").Replace("\u9991", "<<")
            .Replace("\u9992", ">>")
            .Replace("\u9994", "---").Replace("\u9995", "--");
    }

    static string updir = "";
    static string mask = "";
    static int detailcount = 10;
    static int pagecount = 2;
    static int errdocs = 0;
    static List<string> otempl = new List<string>();
    static List<string> templ = new List<string>();

    static void ClearMSWord()
    {
        if (w0 != null)
        {
            foreach (MSWord e in w0)
            {
                try
                {
                    if (e != null)
                    {
                        e.Load(false);
                        e.Dispose();
                    }
                }
                catch
                { }
            }
            w0 = null;
        }
        if (w != null)
            try
            {
                w.Dispose();
            }
            catch
            { }
        w = null;
        if (w1 != null)
            try
            {
                w1.Dispose();
            }
            catch
            { }
        w1 = null;
    }

    static void ProcessTables(MSWord w)
    // чтобы размеры таблиц соответствовали размеру страницы
    {
        for (int i = w.TableCount - 1; i >= 0; --i)
```

```
            {
                w.TableToText(i);
                w.TextToTable();
            }
        }

        static void ProcessFile(string name, string fdatname)
        {
            string sname = name.Substring(updir.Length + 1);
            Console.Write($"Анализ файла {sname}: ");
            if (w == null)
                w = new MSWord(name, ShowWord);
            else
                w.Load(name);
            if (w1 == null)
            {
                File.Delete(fdatname);
                w1 = new MSWord(fdatname, ShowWord);
            }

            string s = w.TryAllText;
            for (int i = 0; i < templ.Count; ++i)
            {
                string t = templ[i];
                MatchCollection mm = null;
                if (s != null)
                {
                    mm = Regex.Matches(s, t);
                    Console.Write(mm.Count + " ");
                    if (mm.Count == 0)
                        continue;
                }
                if (w0[i] == null)
                {
                    w0[i] = new MSWord(w0name[i], ShowWord);
                    w0[i].SelectAll();
                    string s2 = otempl[i][0] == '=' ?
                        "(без применения механизма регулярных выражений):" :
                        "(с применением механизма регулярных выражений):";
                    w0[i].SelText = $"Шаблон поиска {s2}{MSWord.ParMark}{MSWord.ParMark}";
                    w0[i].SelLength -= 1;
                    w0[i].ParNormal();
                    w0[i].AddPars(otempl[i].Substring(1));
                    w0[i].ParNormal();
                    w0[i].Selection.Font.Bold = 1;
                    w0[i].AddPars("Каталог верхнего уровня:");
                    w0[i].ParNormal();
                    w0[i].AddPars(Path.GetFullPath(updir));
                    w0[i].ParNormal();
                    w0[i].Selection.Font.Bold = 1;
                }
                w0[i].AddPars($"Файл: {sname}");
                w0[i].ParNormal();
                w0[i].SelLength -= 7;
                w0[i].SelStart += 6;
                w0[i].Selection.Font.Bold = 1;
                w0[i].Selection.Font.Underline = WdUnderline.wdUnderlineSingle;
                w0[i].AddHyperlink(Path.GetFullPath(name));
                if (mm == null)
                {
                    Console.Write("?");
```

```
                    w0[i].AddPars("Особенности документа не позволили организовать в нем
полнотекстовый поиск.");
                    w0[i].ParNormal();
                    w0[i].Selection.Font.Color = WdColor.wdColorRed;
                    errdocs++;
                    continue;
                }
                w0[i].AddPars($"Количество найденных вхождений: {mm.Count}");
                w0count[i] += mm.Count;
                w0[i].ParNormal();
                for (int j = 0; j < Math.Min(mm.Count, detailcount); ++j)
                {
                    w.SelectRange(mm[j].Index, mm[j].Length);
                    w.ExpandSelection();
                    int start1 = mm[j].Index - w.SelStart;
                    w.Copy();
                    int pg2 = w.SelEndAdjPageNumber;
                    w.SelectPos(w.SelStart);
                    int pg1 = w.SelEndAdjPageNumber;

                    if (pg2 - pg1 <= pagecount - 1)
                    {
                        w1.SelectAll();
                        w1.Paste();
                        ProcessTables(w1);
                        w1.SelectRange(start1, mm[j].Length);
                        w1.Selection.Font.Color = WdColor.wdColorGreen;
                        w1.SelectAll();
                        w1.SelLength -= 1;
                        w1.Cut();
                    }
                    string pg = $"Стр.{MSWord.NBSp}{pg1}";
                    if (pg1 != pg2)
                        pg += MSWord.MDash + pg2;
                    if (pg2 - pg1 > 1)
                        pg = $"({j + 1}) {pg}.";
                    else
                        pg = $"({j + 1}) {pg}:";
                    w0[i].AddPars(pg);
                    w0[i].ParNormal();
                    w0[i].Selection.Font.Italic = 1;
                    w0[i].Selection.Font.Bold = 1;
                    w0[i].ToSelEnd();
                    if (pg2 - pg1 <= pagecount - 1)
                        w0[i].Paste();
                }
            }
            Console.WriteLine();
        }

        static void FinalProcessing(string name)
        {
            w.Load(false);
            if (w1 == null)
                return;
            w1.SelectPos(0);
            for (int i = 0; i < w0.Length; ++i)
                if (w0[i] != null)
                {
                    w0[i].SelectAll();
                    w0[i].SelLength -= 1;
```

```
                    w0[i].Cut();
                    w1.Paste();
                    w1.SelectPos(w1.LastPos);
                    if (i < w0.Length - 1)
                    {
                        w1.SelText = MSWord.PageBreak;
                        w1.ToSelEnd();
                    }
                }
            Console.WriteLine($"\nПросмотрено файлов: {totalcount}");
            int n = 0;
            for (int i = 0; i < w0count.Length; ++i)
            {
                Console.WriteLine($"Найдено вхождений шаблона {i + 1}: {w0count[i]}");
                n += w0count[i];
            }
            if (errdocs > 0)
                Console.WriteLine($"Количество необработанных файлов: {errdocs}");
            if (n + errdocs > 0)
                Console.WriteLine($"Подробная информация о результатах поиска приведена в
файле \n{name}");
        }


        static void DoFind(string name)
        {
            if (!Directory.Exists(updir))
            {
                Err($"Каталог {updir} не найден.");
                return;
            }
            string[] ff = Directory.GetFiles(updir, mask + ".doc?",
SearchOption.AllDirectories)
                .Where(e => ((Path.GetExtension(e).ToLower() == ".doc" ||
                Path.GetExtension(e).ToLower() == ".docx") &&
                ((File.GetAttributes(e) & FileAttributes.Hidden) != FileAttributes.Hidden)
                && (!Path.GetFileNameWithoutExtension(e).ToLower().EndsWith("-found"))))
                .OrderBy(e => e).ToArray();
            if (ff.Length == 0)
            {
                Err($@"В каталоге {updir} и его подкаталогах
не найдены файлы с расширениями doc и docx.");
                return;
            }
            ClearMSWord();
            w0 = new MSWord[templ.Count];
            w0name = new string[templ.Count];
            w0count = new int[templ.Count];
            for (int i = 1; i <= templ.Count; ++i)
            {
                w0name[i - 1] = Path.GetTempPath() + Path.GetRandomFileName() + ".doc";
            }
            totalcount = ff.Length;
            foreach (string f in ff)
            {
                ProcessFile(f, name);
            }
            FinalProcessing(name);
        }

        static void ProcessFDat(string name)
```

```
        {
            bool res = true;
            name = Path.GetFullPath(name);
            if (Path.GetExtension(name) == "")
                name = Path.ChangeExtension(name, ".fdat");
            if (!File.Exists(name))
            {
                Err($"Файл {name} не найден.");
                return;
            }
            updir = Directory.GetCurrentDirectory();
            mask = "*";
            detailcount = 10;
            pagecount = 2;
            templ.Clear();
            otempl.Clear();
            foreach (string s0 in File.ReadLines(name, Encoding.Default))
            {
                string s = s0.Trim();
                if (s == "" || s.StartsWith("%"))
                    continue;
                if (s.StartsWith("updir="))
                {
                    updir = s.Substring(6).Trim();
                    continue;
                }
                if (s.StartsWith("mask="))
                {
                    mask = s.Substring(5).Trim();
                    mask = Path.GetFileNameWithoutExtension(mask);
                    continue;
                }
                if (s.StartsWith("detailcount="))
                {
                    s = s.Substring(12).Trim();
                    int n;
                    if (int.TryParse(s, out n) && n >= 0)
                        detailcount = n;
                    continue;
                }
                if (s.StartsWith("pagecount="))
                {
                    s = s.Substring(10).Trim();
                    int n;
                    if (int.TryParse(s, out n) && n >= 2)
                        pagecount = n;
                    continue;
                }
                if (s.StartsWith("="))
                {
                    otempl.Add(s);
                    s = Regex.Escape(ProcessStr(s.Substring(1).TrimEnd())
                        .Replace("{|}", "\u9993").Replace("|", MSWord.TabMark)
                        .Replace("\u9993", "|"));
                    s = s.Replace(@"\{@}", "\u9990").Replace("@", "|")
                        .Replace("\u9990", "@");
                    templ.Add(s);
                    continue;
                }
                if (s.StartsWith("-"))
                {
```

```
                    otempl.Add(s);
                    s = ProcessStr(s.Substring(1).TrimEnd());
                    templ.Add(s);
                    try
                    {
                        Regex.IsMatch("", s);
                    }
                    catch (Exception ex)
                    {
                        Err(ex.Message);
                        res = false;
                    }
                    continue;
                }
            }
            if (res)
                DoFind(Path.GetDirectoryName(name) + "\\" +
                Path.GetFileNameWithoutExtension(name) + "-found.doc");
        }

        static void Main(string[] args)
        {
            Console.WriteLine("MSWordFind 1.0\n");
            try
            {
                System.Threading.Thread.CurrentThread.CurrentCulture =
                    new System.Globalization.CultureInfo("en-US");
                if (args.Length > 0)
                    foreach (string e in args)
                        ProcessFDat(e);
                else
                {
                    string[] f = Directory.GetFiles(Directory.GetCurrentDirectory(),
"*.fdat");
                    if (f.Length == 0)
                        Err("В каталоге не найдены файлы с расширением fdat.");
                    else
                    if (f.Length == 1)
                        ProcessFDat(f[0]);
                    else
                    {
                        Console.WriteLine("Выберите файл для обработки или нажмите Enter для
завершения программы:");
                        for (int i = 0; i < f.Length; ++i)
                            Console.WriteLine((i + 1) + " - " + Path.GetFileName(f[i]));
                        Console.Write("Номер файла: ");
                        string res = Console.ReadLine();
                        int n;
                        if (int.TryParse(res, out n) && n > 0 && n <= f.Length)
                            ProcessFDat(f[n - 1]);
                    }
                }
            }
            catch (Exception ex)
            {
                Err(ex.Message);
            }
            ClearMSWord();
            if (ErrorsFound)
                Console.Write("\nБыли обнаружены ошибки. ");
            Console.Write("\nДля завершения программы нажмите Enter. ");
```

```
            Console.ReadLine();
        }
    }
}
```

Файл test.fdat

```
=Абзац 1@Абзац 2
```

Консольный вывод

```
D:\...!2019-Материалы\NET\2 семестр\02-MSWordFind\05-
MSWord\bin\Debug>MSWordFind.exe
MSWordFind 1.0

Анализ файла 1.doc: 0
Анализ файла 2.doc: 0
Анализ файла 3.doc: 0
Анализ файла 4.doc: 0
Анализ файла a.doc: 0
Анализ файла b.doc: 9

Просмотрено файлов: 6
Найдено вхождений шаблона 1: 9
Подробная информация о результатах поиска приведена в файле
D:\!!!!!!!!!2019-Материалы\NET\2 семестр\02-MSWordFind\05-
MSWord\bin\Debug\test-found.doc

Для завершения программы нажмите Enter.
```

Файл test-found.doc