Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

|  |  |
| --- | --- |
| Факультет | Комп’ютерної інженерії та управління |
|  | (повна назва) |

|  |  |
| --- | --- |
| Кафедра | Електронних обчислювальних машин |
|  | (повна назва) |

**КУРСОВА РОБОТА**

|  |
| --- |
|  |
| (рівень вищої освіти) |

|  |  |
| --- | --- |
| з дисципліни: | Системне програмне забезпеыення |
|  |  |
| на тему: | Розробка програмного забезпечення з використанням |
|  | платформи .NET: «Редактор файлів формату JSON» |
|  |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Виконав: студент | | | 3-го | курсу, групи | | | КІУКІ-18-4 |
| спеціальності | | 123 «Комп’ютерна інженерія» | | | | | |
|  | | (код і повна назва спеціальності) | | | | | |
| Освітньо-професійної програми: «Комп’ютерна інженерія» | | | | | | | |
| (повна назва освітньої програми) | | | | | | | |
|  |  | | | |  | Акіменко Б. В. | |
|  | (підпис) | | | |  | (прізвище, ініціали) | |
|  |  | | | |  |  | |
|  |  | | | |  |  | |
|  |  | | | |  |  | |
| Голова комісії: |  | | | |  |  | |
| Члени комісії: | (підпис) | | | |  | (посада, прізвище, ініціали) | |
|  | (підпис) | | | |  | (посада, прізвище, ініціали) | |
|  | (підпис) | | | |  | (посада, прізвище, ініціали) | |

2020 р.

РЕФЕРАТ

Записка пояснювальна до курсової роботи: 28 с., 6 рис., 6 розділів, 1

додаток.

Мета роботи — розробка додатку редагування файлів формату JSON використовуючи мову С# та платформу .NET.

Методи вирішення задачі — використання стандартних компонентів Windows, розбробка алогоритмів, використання подій.

Розроблено додаток, який надає можливість редагувати, зберігати, перевіряти належність до формату JSON текстові файли. Додаток дозволяє збільшувати, зменшувати шрифт тексту, згортати, розгортати JSON вузли. Ключі, значення забарвлюються різними кольорами для кращого візуального сприйняття.

Програму складено мовою C#, використовуючи технологію Windows Forms для платформи .NET у середовищі програмування Visual Studio 2019.

JSON, КЛЮЧ, ЗНАЧЕННЯ, ФОРМАТ, FAST COLORED TEXTBOX, РІЗНОКОЛЬОРОВЕ ЗАБАРВЛЕННЯ, СТЕК, АЛГОРИТМ ДЕЙКСТРИ, СХЕМА ВАЛІДАЦІЇ, ПОШУК, ПІДРАХУНОК СТАТИСТИКИ.

ЗМІСТ

Вступ 4

1. Аналіз та постановка задачі 5
2. Теоретичні відомості 6
   1. JSON та його структура 6
   2. Недопустимі випадки використання JSON 7
3. Розробка структури та алгоритмів 9
   1. Структура програми 9
   2. Алгоритм перевірки відповідності тексту до формату JSON 10
4. Опис програми 16
   1. Опис компоненту компоненту Fast Colored TextBox 16
   2. Реалізація компоненту Fast Colored TextBox 16
   3. Різнокольоровий синтаксис за допомогою   
      компоненту Fast Colored TextBox 18
   4. Звуження/розширення вузлів JSON 20
   5. Обробники з затримкою 20
   6. Пріоритети стилів 21
   7. Автовідступ 21
   8. Автовідступ для символів 22
   9. Відкладене завантаження файлів 22
   10. Дії назад/вперед 23
   11. Продуктивність та обмеження 23
5. Ручне тестування програми 25
6. Інструкція користувача 27

Висновки 29

Перелік джерел посилань 30

ВСТУП

При розробці додатків виникає необхідність серіалізувати об'єкти для їх збереження на накопичувачі чи передачі до інших комп'ютерів через мережу. Стани об’єктів зберігаються в різні формати: бінарний, SOAP, XML, JSON.

З суб’єктивної точки зору автора JSON формат є найбільш зручним способом представлення об’єктів для перегляду людиною, тому редагування цього файлу безпосередньо програмістом цілком можливе. Однак, у результаті редагування може бути отриманий файл, який не буде правильно розпізнаний програмами, з причини забутої коми, круглої чи квадратної дужки, тому, засоби редагування повинні мати і засоби перевірки коректності відредагованого тексту.

Метою курсової роботи є створення редактора файлів формату JSON, що надає можливість редагувати, зберігати, переглядати та перевіряти відповідність тексту до JSON формату.

Для реалізації зазначеної мети передбачається вирішення таких завдань:

– забезпечити стандартні фалові операції (відкирити файл, закрити, створити новий, зберегти);

– забезпечити стандартні команди редагування (копіювання, переміщення, дії назад вперед);

– відображати ключі/значення різими кольорами;

– забезпечити можливість робити валідацію JSON файлу.

1 АНАЛІЗ ТА ПОСТАНОВКА ЗАДАЧІ

Необхідно створити зручний спосіб створення та редагування файлів формату JSON, тому необхідно реалізувати наступний функціонал:

1) відкиривати, закривати, створювати новий, зберігати файл;

2) копіювати, переміщувати, текст;

3) відміняти попередні зміни в тексті, робити зміни повторно;

4) відображати ключі/значення різими кольорами;

5) перевірка на належність тексту до формату JSON;

6) переглядати статистику про файл (кількість рядків, слів, символів);

7) здіснювати пошук у файлі;

8) показувати номер для кожного рядку файлу;

9) можливість збільшувати, зменшувати текст для зручності користувача;

10) можливість згортати/розгортати вміст JSON вузлів.

Для реалзації можливості відкиривати, зберігати файл необхідно віикористовувати стандартні діалогові вікна Windows.

Оскільки файли формату JSON призначені для передачі та зберігання даних, це, як правило, великі файли, тому необхідно використовувати більш швидкі компоненти .NET для роботи з текстом.

Статистику тексту (кількість рядків, слів, символів) можна виводити до елементу status bar внизу зліва основного вікна додатку. Оскільки функція аналізу тексту та підрахунку статистики є додатковою, то вона не повина забирати час виконання у основного потоку програми. Основна функція додатку – редагування тексту.

1. ТЕОРЕТИЧНІ ВІДОМОСТІ

2.1 JSON та його структура

JSON (JavaScript Object Notation) - це відкритий формат файлу та формат обміну даними, який використовує текст для зберігання та передачі об’єктів - даних, що складаються з пар ключ: значення та масивів. Як і XML придатний до читання людиною. Розробник формату – американський програміст Дуглас Крокфорд. Перетворення об’єкту в пам’ті у формат JSON називають процесом серіалізації. JSON відомий як один з найбільш широко використовуваних форматів для серіалізації.

JSON заснований як підмножина мови програмування JavaScript, визначеному в стандартах ECMA-262 3-ої версії видання у грудні 1999 р. JSON – це текстовий формат, повністю незалежний від мови, але він використовує згоди, знайомі програмістам C-подібних мов, таких як C, C++, C#, Java, JavaScript, Perl, Python та інших.

JSON файл будуються на двух базових структурах даних:

1) коллекція пар ключ/значення. У різних мовах, ця концепція реалізована як об'єкт, запис, структура, словник, хеш або асоціативний масив;

2) впорядкований список значень. У більшості мов це реалізовано як масив, вектор, список або послідовність.

Вищезгадані структури даних є універсальними. Майже всі сучасні мови програмування підтримують їх у тій чи іншій формі, тому, цілком закономірно, що формат даних, незалежний від мови програмування, повинен бути заснований на цих структурах

Перша структура об'єкт – це неупорядкований набір пар ключ - значення. Об'єкт починається з відкриваючої фігурної скобки ({) та закінчується закриваючою фігурною скобкою (}). Після кожного ключу ставиться двокрапка (:). Пари ключ/значення розділяються комою. За стандартом кожен ключ повинен бути в подвійних лапках. Ця властивість дає більш простий алгоритм розбору JSON файлу, на відміну від формату де ключі могли б бути без подвійних лапок.

Друга структура – масив – впорядкована сукупність значень. Масив починається із відкриваючої квадратної скобки ([) та закінчується закриваючою квадратною скобкою (]). Значення розділяються комою.

Значення може бути рядком в подвійних лапках, числом, true, false, null, об'єктом або масивом. Ці структури можуть бути вкладеними. Рядок - сукупність символів Unicode, поміщена в подвійні лапки. Символ зворотної косої риски (\) використовіється в якості символу екранування. Символ представляється як односимвольний рядок. Схожий синтаксис використовується в C# і Java. Число представляється так само, як в C# або Java, крім того, що використовується тільки десяткова система числення. JSON має три зарезвовані слова: true, false, null. Перші два відносяться до літералів булівого (логічного) типу, а null позначає, що поле порожнє.

2.2 Недопустимі випадки використання JSON

Оскільки серіалзацію використовує кожний другий додаток, необхідно розуміти межі застосування серіалізції.

Серіаліція неприпустима у випадку циклічних посилань. Наприклад, об’єкт A посилається на об’єкт В, а об’єкт В посилається на об’єкт А:

class A

{

public B RefToB;

}

class B

{

public A RefToA;

}

При серіалізції об’єкту А буде розкритий об’єкт В, який в свою чергу має об’єкт А і виникає рекрсивне вкладення об’єктів. Подібне рекрсивне вкладення може спостерігатися коли об’єкт має посилання на самого себе:

class A

{

public A RefToMyself;

}

У мові С# можна виправити цю проблему вказавши, що ці поля класів не поинні бути серіалозовані за допомогою атрибуту [NonSerialized]:

class A

{

[NonSerialized]

public B RefToB;

}

class B

{

[NonSerialized]

public A RefToA;

}

1. РОЗРОБКА СТРУКТУРИ ТА АЛГОРИТМІВ ПРОГРАМИ

3.1 Структура програми

Вхідною точкою програми є статичний метод Main() статичного класу Program. У тілі методу запускається основне вікно програми, що описане у класі MainWindow. У фаул TextProcessor.cs містить опис стукутур – типів що повертають статичні методи статичного класу TextProcessor. Приклад таких статичних методыв класу TextProcessor: функція перевірки належності тексту до формату JSON, функція, що рахує кількість рядків та слів файлу.

Клас MainWindow представляє основне вікно програми, він унаслідуваний від класу Form. Як показано на рис. 2.1, цей клас має меню, служку інструментів, область для тексту, випадаючее меню збільшення/зменшення розміру тексту.

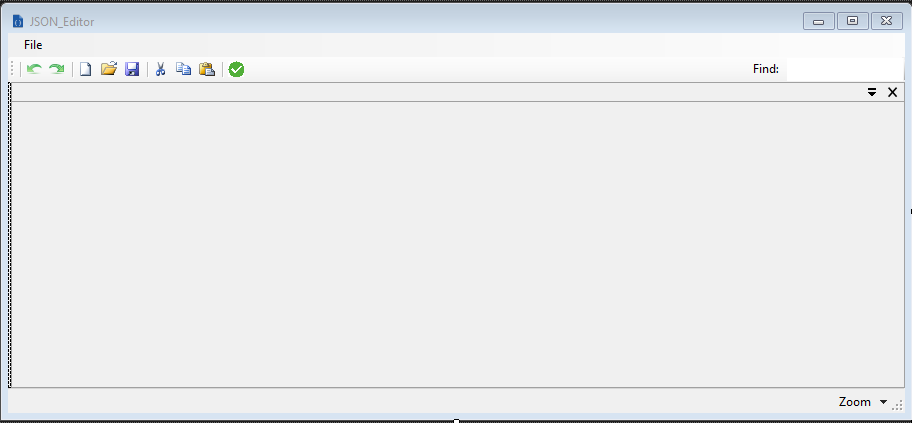


Рисунок 2.1 – Основне вікно, відкрите у кострукторі Visual Studio.

Клас має багато методів, що встановлені як обробники подій від елементів інтерфейсу, зокрема метод, TsbVerify\_Click(), що оброблює клік по іконці запиту верифікації, метод tb\_TextChangedDelayed(), що викликається з затримкою в 1 секунду після кожної модифікації тексту. Метод tb\_TextChangedDelayed() має перерахувати кількість слів, рядків, символів тексту після модифікації тексту.

Оскільки JSON файли можуть бути досить великими, перерахування рядків, слів, символів може викликати затримку реакції програми на дії користувача, тому функції з статичного класу TextProcessor, що аналізують текст виконуються в окремому потоці за допомоогою методу ThreadPool.QueueUserWorkItem().

3.2 Алгоритм перевірки відповідності тексту до формату JSON

У постановці завдання було зазначено, що необхідно забеззпечити механізм, який буде аналізувати текст та робити висновки про його відповідність або невідповідність до формату JSON.

Оскільки JSON об’єкт починається з відкриваючої фігурної дужки і закривається фігурною дужкою, JSON масив відкривається квадратною і закривається відповідною, це схоже не те, як відкриваються та закриваються круглі дужки у математичних виразах, тому для перевірки JSON формату було прийнято рішення використовувати ідею алгоритму перетворення математичного виразу з інфіксної форми до постфіксної форми (алгоритму Дейкстри). Основа робоити цього алгоритму – це використання стеку. Як у алгоритмі Дейкстри до стеку заносяться круглі дужки та знаки математичних операцій, так і в алгоритмі перевірки формату JSON фігурні дужки, квадратні дужки, коми, двокрапки заносяться до стеку.

Алгоритм перевіряє послідовно кожний символ тексту і в залежності від сиволу та стану стеку виконує ті чи інші дії. З цього складність алгоритму прямопропорційна кількості символів тексту.

Якщо буде натраплено на символи, що задають структуру формату JSON, то будуть виконані такі дії:

1) для відкриваючої фігурної дужки ({) – занесення її до стеку;

2) для закриваючої фігурної дужки (}): якщо стек порожній – текст не відповідає формату JSON. Якщо на вершині стеку відкриваюча фігурна дужка, або двокрапка, а за нею відкриваюча фігурна дужка – вони видаляються, якщо інші знаки на вершині стеку – текст не відповідає формату JSON.

3) для подвійних дужок (“), що всередині тримають значення: якщо у стеці є відкриваюча подвійна дужка – видалення її з стеку, якщо ні - занесення дужки до стеку;

4) для двокрапки: якщо стек порожній - текст не відповідає формату JSON, якщо ні – занесення двокрапки до стеку.

5) для коми: якщо стек порожній – текст не відповідає формату JSON. Якщо на вершині стеку двокрапка – видалення її зі стеку, якщо ні – занесення коми до стеку (це може бути кома при переразуванні елементів масиву).

6) для відкриваючої квадратної дужки – занесення її до стеку;

7) для закриваючої квадратної дужки – якщо стек порожній – текст не відповідає формату JSON. Усі коми на веришині стеку видалються до відкриваючої дужки. Після ком повинна бути відкриваюча дужка, яка також видаляється зі стеку, якщо після ком якийсь інший символ - текст не відповідає формату JSON.

Для того, аби виявити помилки при яких загублене значення або ключ потрібно:

1) для значення - після двокрапки (:) пропустити всі пробіли та перевірити символ, що слідує за пробілами. Якщо це символ, ‘t’ або ‘f’ або ‘”’ або ‘{‘ або ‘[‘ або цифра, то це початок значення, алгоритм далі робить валідацію. Якщо це якийсь інший символ, то після двокрапки слідує не значеня – текст не відповідає формату JSON.

2) для ключу – після коми або вікриваючої фігурної дужки ({) пропустити всі пробіли та симвoли нового рядку (‘\n’ aбо ”\r\n”) та перевірити символ, що слідує за ними. Якще цей символ є подвійною відкривачуєю лапкою ‘”’, то це початок ключу, алгоритм далі робить валідацію. Якщо це якийсь інший символ, то це ключ – текст не відповідає формату JSON.

Оскільки перевірка наявності ключу необхдідна у двох місцях (після відкриваючої фігурної дужки та після коми), ця перевірка була реалізована у статичному методі, який можна викликати у двох місцях:

private static bool isSomethingLikeKey(ref int index, string text)

{

//check if there is anytihing

// anything like key

char ch = '\0';

do

{

++index;

ch = text[index];

if (ch == ' '

|| ch == '\r'

|| ch == '\n')

{

continue;

}

else

if (ch == '"')

{

--index;

return true;

}

else

{

return false;

}

} while (true);

}

Після перебору кожного символу тексту, якщо стек порожній – текст відповідає формату JSON, якщо ні – текст не відповідає формату JSON. Мовою С# алгоритм реалізований наступним чином:

public static ValidationResult isValidJson(string text)

{

Stack<char> stack = new Stack<char>();

ValidationResult result;

//set initial values

result.Row = 1; result.At = 1; result.IsValid = false;

char last = '\0';

for (int i = 0; i < text.Length; ++i)

{

char ch = text[i];

// ignore symbols in double quotes

if (stack.Count != 0

&& '"' == stack.Peek()

&& '"' != ch

)

{

++result.At;

continue;

} else if (Char.IsLetter(ch)) {

//keywords can be without quotes

if (text.Substring(i, 4) == "null"

|| text.Substring(i, 4) == "true")

{

i += 3;

continue;

}

else if (text.Substring(i, 5) == "false")

{

i += 4;

continue;

}

else

{

return result;

}

}

switch (ch)

{

case '\"':

if (stack.Count == 0)

{

return result;

}

last = stack.Peek();

if (last == '\"')

{

// It's a closing quote

stack.Pop();

}

else

{

// It's an opening quote

stack.Push(ch);

}

break;

case ':':

if (stack.Count == 0)

{

return result;

}

last = stack.Peek();

if ('"' != last)

{

stack.Push(ch);

//check if there is anytihing

// anything like value

do

{

++i;

ch = text[i];

if (ch == ' ')

{

continue;

} else

if (ch == 'n'

|| ch == 't'

|| ch == 'f'

|| ch == '"'

|| ch == '{'

|| ch == '['

|| Char.IsDigit(ch))

{

break;

} else

{

return result;

}

} while (ch == ' ');

--i;

}

break;

case ',':

if (stack.Count == 0)

{

return result;

}

last = stack.Peek();

if (':' == last)

{

stack.Pop();

if (!isSomethingLikeKey(ref i, text))

{

return result;

}

} else

{

stack.Push(ch);

}

break;

case '{':

if (isSomethingLikeKey(ref i, text))

{

stack.Push(ch);

} else

{

return result;

}

break;

case '}':

if (stack.Count == 0)

{

return result;

}

last = stack.Peek();

if (last == '{')

{

stack.Pop();

}

else

{

if (last == ':')

{

stack.Pop();

last = stack.Peek();

if ('{' == last)

{

stack.Pop();

}

else

{

return result;

}

}

else

{

return result;

}

}

break;

case '[':

stack.Push(ch);

break;

case ']':

if (stack.Count == 0)

{

return result;

}

last = stack.Peek();

while (',' == last)

{

stack.Pop();

last = stack.Peek();

}

if (last == '[')

{

stack.Pop();

}

else

{

return result;

}

break;

case '\n':

++result.Row;

result.At = 1;

break;

}

++result.At;

}

result.IsValid = (0 == stack.Count);

return result;

}

1. ОПИС ПРОГРАМИ

4.1 Опис компоненту компоненту Fast Colored TextBox

Оскільки необхідно вирішити задчу відображення ключів/значеннь різими кольорами та забезпечити нормальну роботу з великими файлами (оскільки серіалізовані дані можуть бути досить великими), було вирішено використовувати в якості контейнеру для тексту компонент Fast Colored TextBox. Спочатку я використовувався компонент RichTextBox, але, використовуючи його для великої кількості тексту, було виявлено, що RichTextBox дуже повільно виділяє велику кількість кольорових фрагментів (від 200 і більше). Коли таке підсвічування має бути зроблено динамічно, це спричиняє серйозні проблеми.

Компонент Fast Colored TextBox працює досить швидко з великою кількістю тексту, а також має інструменти для комфортного динамічного підсвічування синтаксису. Він має такі налаштування, як колір переднього плану, стиль шрифту, колір заднього плану, які можна налаштувати для довільно вибраних символів тексту. Можна легко отримати доступ до тексту за допомогою регулярних виразів. Підтримуються також операції **Find/Replace**, а також **Undo/Redo**.

4.2 Реалізація компоненту Fast Colored TextBox

Для зберігання символів тексту використовується структура Char:

public struct Char

{

public char c;

public StyleIndex style;

}

Структура зберігає символ (символ, 2 байти) та маску стилю (StyleIndex, 2 байти). Таким чином, на кожен символ тексту витрачається 4 байти пам'яті. Символи згруповані в рядки, які реалізовані за допомогою List <Char>.

StyleIndex - це маска індексів стилів, застосована до цього символу. Кожен біт StyleIndex означає, що цей символ буде намальований відповідним стилем. Оскільки StyleIndex має 16 біт, елемент керування підтримує не більше 16 різних стилів.

public enum StyleIndex : ushort

{

None = 0,

Style0 = 0x1,

Style1 = 0x2,

Style2 = 0x4,

Style3 = 0x8,

Style4 = 0x10,

Style5 = 0x20,

Style6 = 0x40,

Style7 = 0x80,

Style8 = 0x100,

Style9 = 0x200,

Style10 = 0x400,

Style11 = 0x800,

Style12 = 0x1000,

Style13 = 0x2000,

Style14 = 0x4000,

Style15 = 0x8000,

All = 0xffff

}

Стилі зберігаються в окремому списку:

public readonly Style[] Styles = new Style[sizeof(ushort)\*8];

За своєю суттю клас Style - це візуалізатор символів, фонів, меж та інших елементів дизайну тексту. Нижче наведено типову реалізацію одного зі стилів для відтворення текстових символів:

public class TextStyle : Style

{

public Brush ForeBrush { get; set; }

public Brush BackgroundBrush { get; set; }

public FontStyle FontStyle { get; set; }

public override void Draw(Graphics gr, Point position, Range range)

{

//draw background

if (BackgroundBrush != null)

gr.FillRectangle(BackgroundBrush, position.X, position.Y,

(range.End.iChar - range.Start.iChar) \*

range.tb.CharWidth, range.tb.CharHeight);

//draw chars

Font f = new Font(range.tb.Font, FontStyle);

Line line = range.tb[range.Start.iLine];

float dx = range.tb.CharWidth;

float y = position.Y - 2f;

float x = position.X - 2f;

Brush foreBrush = this.ForeBrush ?? new SolidBrush(range.tb.ForeColor);

for (int i = range.Start.iChar; i < range.End.iChar; i++)

{

//draw char

gr.DrawString(line[i].c.ToString(), f, foreBrush, x, y);

x += dx;

}

}

}

TextStyle містить колір переднього плану, колір заднього плану та стиль шрифту тексту. Створюючи новий стиль, компонент перевіряє стиль у своєму списку, і якщо стилю немає, він створює новий стиль зі своїм індексом.

Можна створювати нові стил, успадковані від класу Style.

Для роботи з фрагментами тексту використовується клас Range, що представляє суцільний блок тексту з урахуванням початкової та кінцевої позицій:

public class Range

{

Place start;

Place end;

}

public struct Place

{

int iLine;

int iChar;

}

4.3 Різнокольоровий синтаксис за допомогою компоненту Fast Colored TextBox

На відміну від RichTextBox, компонент не використовує RTF. Інформація про колір та тип символів зберігається лише в компоненті. Це означає, що фарбування компонента повинно перероблятися кожного разу під час введення тексту. Для цього використовується подія TextChanged.

Об'єкт Range, що містить інформацію про діапазон тексту, що модифікується, переходить до події TextChanged. Це дозволяє виділити лише змінений фрагмент тексту.

Для пошуку фрагментів тексту, які потрібно забарвити, можна застосувати перевантажений метод Range.SetStyle (), який приймає шаблон пошуку (регулярний вираз):

Style GreenStyle = new TextStyle(Brushes.Green, null, FontStyle.Italic);

...

private void fastColoredTextBox1\_TextChanged(object sender, TextChangedEventArgs e)

{

//clear style of changed range

e.ChangedRange.ClearStyle(GreenStyle);

//comment highlighting

e.ChangedRange.SetStyle(GreenStyle, @"//.\*$", RegexOptions.Multiline);

}

Перед початком виклику забарвлення використовується метод Range.ClearStyle() для очищення та видалення попереднього стилю.

Метод SetStyle() виділяє фрагмент тексту, відповідний регулярному виразу. Однак, якщо вираз включає названу групу "діапазон", виділяється група з іменем "діапазон". Ім'я класу, яке стоїть після ключових слів "class", "struct" і "enum", виділено жирним шрифтом у наступному прикладі:

e.ChangedRange.SetStyle(BoldStyle, @"\b(class|struct|enum)\s+(?<range>[\w\_]+?)\b");

Обробник подій TextChanged, використовуваний для розфарбовування синтаксису JSON.

Окрім події TextChanged, корисними можуть бути події TextChanging, VisibleRangeChanged та SelectionChanged. Подія TextChanging з'являється до того, як текст починає змінюватися. Подія SelectionChanged відбувається після зміни позиції курсора в компоненті або під час модифікації виділеного фрагмента тексту.

4.4 Звуження/розширення вузлів JSON

Компонент Fast Colored Text Box дозволяє звужувати розширювати блоки тексту. Щоб приховати виділений текст, використовуйте метод CollapseBlock():

fastColoredTextBox1.CollapseBlock(fastColoredTextBox1.Selection.Start.iLine,

fastColoredTextBox1.Selection.End.iLine);

Наприклад, для пошуку блоків {..} використовується наступний обробник:

private void fastColoredTextBox1\_TextChanged(object sender, TextChangedEventArgs e)

{

//clear folding markers of changed range

e.ChangedRange.ClearFoldingMarkers();

//set folding markers

e.ChangedRange.SetFoldingMarkers("{", "}");

e.ChangedRange.SetFoldingMarkers(@"#region\b", @"#endregion\b");

}

4.5 Обробники з затримкою

Багато подій (TextChanged, SelectionChanged, VisibleRangeChanged) мають версію події з затримкою. Затримана подія спрацьовує через певний час після настання основної події.

Ця властивість поддій з затримкою використовується для використовується у додатку наступним чином: користувач швидко вводить текст, TextChanged спрацьовує при введенні кожного символу, а подія TextChangedDelayed працює лише після того, як користувач перестав вводити текст, наприклад через 3 секуднди. Подія TextChangedDelayed відбудеться лише один раз на відміну від події TextChanged, що призводить до запуску лише одного потоку, а не декількох. Це дозволяє аналізувати текст (підраховувати кількість слів, JSON ключів та значень) після зупинки редагування тексту.

Елемент керування підтримує наступні відкладені події: TextChangedDelayed, SelectionChangedDelayed, VisibleRangeChangedDelayed. За допомогою властивостей DelayedEventsInterval і DelayedTextChangedInterval можна встановити час очікування.

4.6 Пріоритети стилів

Кожен символ може містити до 16 різних стилів. Тому виникає питання, в якому порядку будуть намальовані ці стилі. Щоб чітко вказати порядок малювання, можна використати метод FastColoredTextBox.AddStyle():

fastColoredTextBox1.AddStyle(MyUndermostStyle);

fastColoredTextBox1.AddStyle(MyUpperStyle);

fastColoredTextBox1.AddStyle(MyTopmostStyle);

Ці методи потрібно викликати перед будь-якими викликами Range.SetStyle(). В іншому випадку порядок стилів визначатиметься порядком викликів методів Range.SetStyle ().

За замовчуванням елемент керування малює лише один стиль TextStyle (або успадкований) - найменший з усіх. Однак ви можете увімкнути малювання символу з багатьма стилями, використовуючи властивість FastColoredTextBox.AllowSeveralTextStyleDrawing. Це стосується лише стилів TextStyle (або успадкованих), інші стилі (успадковані від Style) у будь-якому випадку малюються.

Якщо Сhar не має жодного TextStyle, він буде намальований за допомогою FastColoredTextBox.DefaultStyle. Викличте метод ClearStyleBuffer(), якщо вам потрібно зняти накладені стилі. Для редагування вигляду тексту можна застосувати напівпрозорий колір у стилях.

4.7 Автовідступ

Коли користувач набирає текст і встановлений автовідступ, елемент керування автоматично визначає лівий відступ для вхідного рядка.

Можна встановлювати власні правила для відступу. Для цього необхідно використовувати подію AutoIndentNeeded. Обробник події повинен повертати два значення: Shift і ShiftNextLines.

Значення Shift вказує відступ цього рядка відносно попереднього рядка у символах (може бути від’ємним). Значення ShiftNextLines означає, що відступ буде застосовано до усіх наступних рядків. Крім того, ви можете вручну зробити AutoIndent для виділеного тексту для цього необхідно викликати метод DoAutoIndent().

4.8 Автовідступ для символів

Елемент керування може вирівнювати положення окремих символів у сусідніх рядках. Для цього використовується властивість AutoIndentCharsPatterns, яка містить набір шаблонів (у кожному рядку - один шаблон).

Коли користувач вводить символ, рядок перевіряється щодо кожного шаблону. Якщо рядок, придатний для деяких з шаблонів, перевіряється біля рядків (лівий відступ цих рядків повинен відповідати відступу поточного рядка). А потім ці символи знайдених рядків вирівнюються, вставляючи пробіли. Які символи будуть вирівняні, визначається іменованими групами регулярних виразів, назва групи має бути "діапазон".

Щоб деактивувати автодвідступ для символів необхідно встановити для властивості AutoIndentChars значення false (за замовчуванням це true).

Автовідступ може бути встановлений програмно за допомоогою методу DoAutoIndentChars(). Також користувач може викликати AutoIndentChars за допомогою гарячих клавіш Ctrl + I (це спрацює, навіть якщо AutoIndentChars = false).

4.9 Відкладене завантаження файлів

Для роботи з файлами надзвичайно великого розміру (100 тис. рядків і більше) може бути корисним відкладений режим завантаження. У цьому режимі елемент керування відкриває файл і за потреби зчитує його частини. Невикористані фрагменти тексту видаляються з пам'яті. Цей режим підтримується трьома методами:

OpenBindingFile() - відкриває файл для читання (в монопольному режимі).

CloseBindingFile() - закриває відкритий файл. Після виклику Fast Colored TextBox повертається до звичайного (не відкладеного) режиму.

SaveToFile() – зберігає текст до файлу. Після цього методу Fast Colored TextBox буде зв’язаний з новим файлом. Цей метод також можна використовувати в простому (не відкладеному) режимі.

4.10 Дії назад/вперед

Компонент Fast Colored TextBox запам'ятовує рядки і місця, які користувач відвідав, тому є можливість переміщувати курсор на попереднє місце і назад.

Для того щоб повернутись до попередньої позиції курсору необхідно натиснути Ctrl + -, що викличе метод NavigateBackward(). Для того щоб повернутись назад необхідно натиснути Ctrl + Shift + -, що викличе метод NavigateForward().

Також можете бути використана властивість Line.LastVisit для отримання інформації про останнє відвідування рядка.

4.11 Продуктивність та обмеження

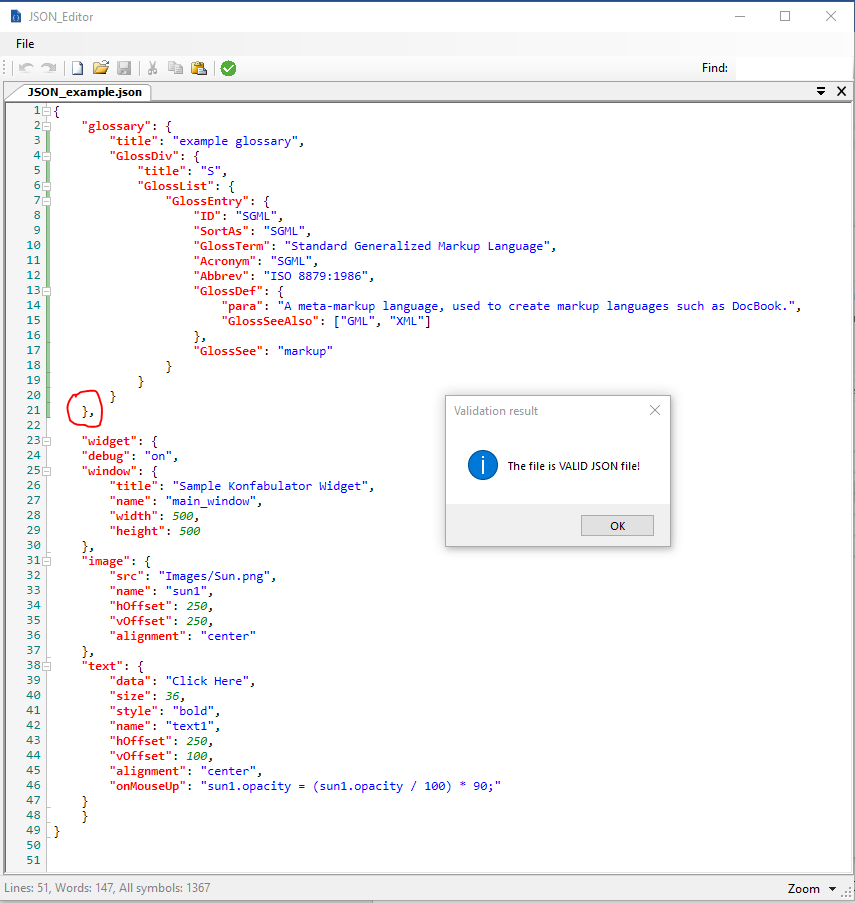
Для зберігання одного мегабайта тексту потрібно приблизно 6 МБ оперативної пам'яті (включаючи стеки скасування/повтору, які зберіють дії, виконані з текстом). Забарвлення тексту різними кольорами не споживає значних ресурсів.

Використання регулярних виразів та економія пам'яті дозволяють досягти високопродуктивного компонента. Для файлу з 50 000 рядків (близько 1,6 МБ) агальний час вставки та забарвлення синтаксису становив близько 3 секунд. Подальша робота з текстом пройшла без значних затримок.

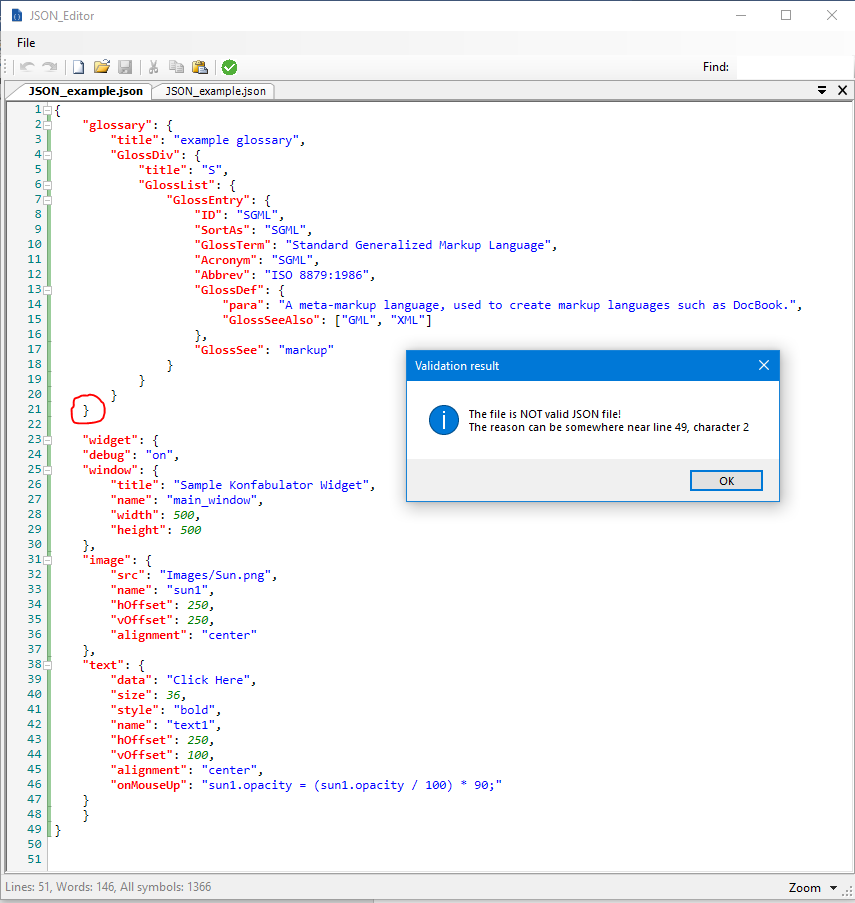
Компонент Fast Colored TextBox не підтримує вирівнювання по центру або праворуч і використовує лише моноширинні шрифти. Також символи табуляції завжди замінюються пробілами.

5 РУЧНЕ ТЕСТУВАННЯ ПРОГРАМИ

Для того, щоб перевірити коректність алгоритму верифікації формату JSON було запущено алгоритм на JSON файлі з правильною структурою та на файлі, що містить помилки. На рис. 4.1 а) показано результат роботи алгоритму на валідному JSON тексті. Програма повернула вірний результат, що файл валідний. На рис. 4.1 б) показано результат роботи алгоритму на тому самому JSON тексті, але у ньому загблена кома. Програма повернула вірний результат, що файл не відповідає формату JSON.



а)



б)

Рисунок 4.1 – Перевірка правильності алгоритму

перевірки на валідність JSON файлу:

а) валідний JSON файл; б) не валідний JSON файл

Таким же чином були перевірено роботу алгоритму на інших файлах і так само було отримано правильний результат.

6 ІНСТРУКЦІЯ КОРИСТУВАЧА

Програма складається з основного файлу виконання JSON\_Editor.exe, динамічних бібліотек FastColoredTextBox.dll та TabStrip.dll.exe, а також файлів FastColoredTextBox.pdb, JSON\_Editor.exe.config, JSON\_Editor.pdb. Інсталювати програму не потрібно.

На рис. 5.1 показані основні компоненти програми. Додаток містить такі компоненти: меню, служка інструментів, пошук, контейнер тексту (Fast Colored TextBox), смужка стану (status strip), випадаюче меню вибору розміру шрифту. У смужку стану виводиться чнформація про кількість рядків, слів, символів тексту. За допомогою випадаючого меню вибору шрифту тексту можна збільшувати, зменшувати шрифт, за допомогою якого відображається текст.

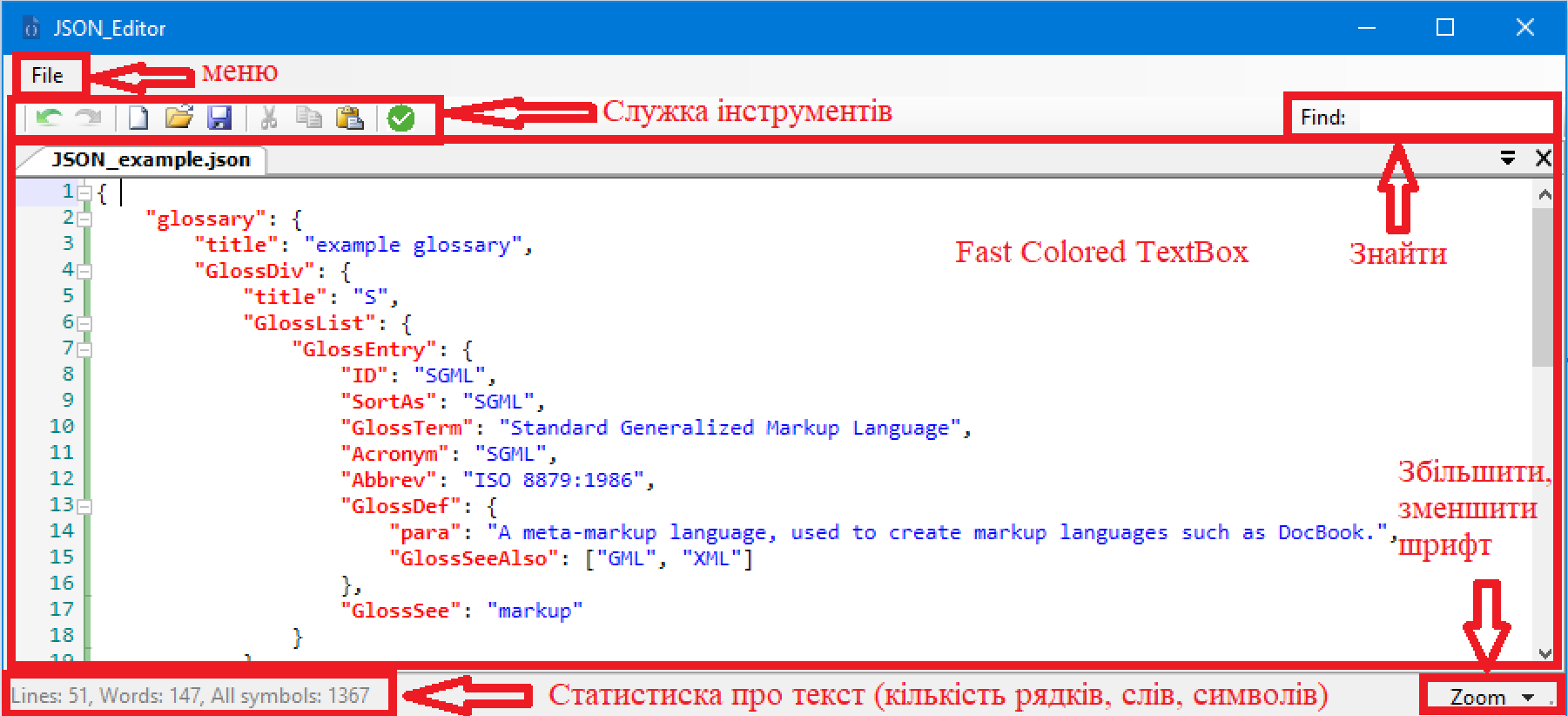


Рисунок 5.1 – Основні компоненти додатку

На рис. 5.2 показано меню програми. За допомогою меню програми можна створювати нові, відкривати, зберігати, зберігати в інших форматах файли. Дії з меню можна запустити за допомогою комбінацій клавіш як на рис. 5.2.

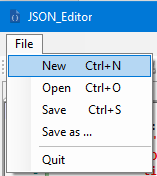


Рисунок 5.2 – Меню програми

На рис. 5.3 більш детально показані елементи смужки інструментів та їх назви.

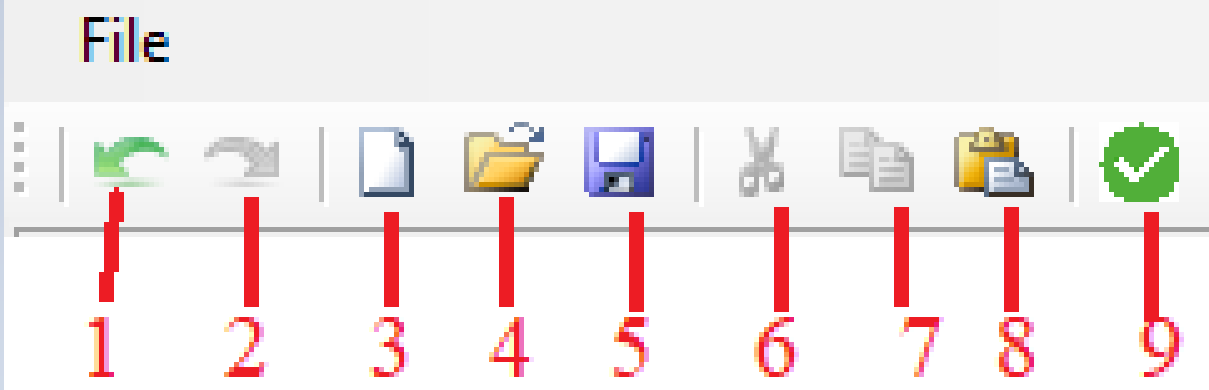


Рисунок 5.3 – Смужка інструментів:

1 – відмінити дію (undo); 2 – повторно зробити дію (redo);

3 – новий файл; 4 - вікрити файл; 5 - зберегти файл;

6 – вирізати текст; 7 – скопійувати текст; 8 – вставити текст;

9 – перевірити відповідність тексту до формату JSON

ВИСНОВКИ

У ході курсової роботи було створено редактор файлів формату JSON, що має таку функціональність: відкриття, редагування, збереження файлів, забарвлення ключів/значень різними кольорами, збільшення шрифту, дії назад/вперед, підрахунок статистики тексту, перевірка належності тексту до формату JSON, пошук тексту, згортання/розгортання вузлів JSON.

Заради збільшення швидкості редагування великих файлів було використано компонент Fast Colored TextBox, що забезпечив інтерфейс для програмування: подія DelayedTextChanged, властивість Text та ін. Роботу з декількома файлами забезпечив компонет TABStrip. Було складено алгоритм первірки тексту до формату JSON за ідеєю алгоритму перетворення математичного виразу з інфіксної форми до постфіксної форми (алгоритму Дейкстри). Алгоритм працює на основі стеку. Складність алгоритму перевірки належності тексту до формату JSON прямопропоріційна кількості символів тексту. Результати тестування алгоритму дали позитивні результати.

В цілому додаток виконує свої функції – редагування файлів формату JSON з ключами/значеннями забарвленими в різні кольори. Досить простий і зручний – усі задачі з постановки завдання були реалізовані. Окрім основного функціоналу було реалізовано також додатковий: підрахунок окремо ключів, окремо значень, пошук по ключам. Проте, створений додаток має ряд недоліків: при роботі з файлом на 76 тисяч рядків при відкритті файлу затримка все ж була, хоча і не великою, не має вибору темної версії, повідомлення про результат верифікації текстом (не дружньо до користувача). Однак, була створена ґрунтовна структура додатку, працюючи з якою в майбутньому можна усунути ці недоліки.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Коноваленко І.В., Федорів П.С. Системне програмування у Windows. Навчальний посібник для технічних спеціальностей вищих навчальних закладів. Тернопіль, - 2012, 152 с.

2. Рихтер Дж. Windows для профессианалов: создание эффективных Win32- приложений с учетом специфики 64-разрядной версии Windows. /Пер. с англ.- 6-е изд. – СПб: Питер; М.: Издательский торговый дом "Русская редакция", 2009. – 752с.

3. Руссинович М., Соломон Д. Внутреннее устройство Microsoft Windows: Windows Server 2003, Windows XP и Windows 2000 M.: Издательско-торговый дом "Русская редакция"; СПб.: Питер, 2008. -803с.