**TKOM – Projekt**

**Dokumentacja końcowa**

Student: Volodymyr Ostruk,

Numer albumu: 255356

Prowadzący: Piotr Gawkowski

**SPIS TREŚCI:**

|  |  |  |
| --- | --- | --- |
| 1. | [TREŚĆ ZADANIA:](#_1._TREŚĆ_ZADANIA:) | **3** |
| 2. | [OPIS FUNKCJONALNOŚCI:](#_2._OPIS_FUNKCJONALNOŚCI:) | **3** |
|  | - [SPECYFIKACJA I SKLADNIA JEZYKA XML](#_SPECYFIKACA_I_SKLADNIA) | **3** |
|  | - [SPECYFIKACJA I SKLADNIA JSON](#_Specyfikacja_i_skladnia) | **7** |
| 3. | [WYMAGANIA, OBSLUGA BLĘDÓW](#_3._WYMAGANIA,_OBSLUGA) | **8** |
| 4. | [ZAŁOŻENIA PROJEKTOWE i SPOSÓB DZIAŁANIA](#_4._ZAŁOŻENIA_PROJEKTOWE) | **12** |
| 5. | [SPOSÓB URUCHOMIENIA, WE/WY](#_5._SPOSÓB_URUCHOMIENIA,) | **13** |
| 6. | [PROJEKT REALIZACJI: OPIS MODUŁÓW I INTERFEJSÓW](#_6._PROJEKT_REALIZACJI:) | **13** |
|  | - [ANALIZA LEKSYKALNA](#_A._Analiza_leksykalna) | **13** |
|  | - [ANALIZA SKLADNIOWA I GRAMATYKA](#_B._Analiza_składniowa) | **14** |
|  | [- DZIALANIE PROGRAMU](#_C._Działanie_programu) | **14** |
|  | [- PODSTAWOWE KLASY PROGRAMU](#_D._Podstawowe_klasy) | **16** |
| 7. | [TESTOWANIE](#_7._PRZYKLADY_TESTOWE) | **17** |
| 8. | [DODATKI](#_Pełna_notacja_EBNF) | **21** |

## 1. TREŚĆ ZADANIA:

Napisać translator języka znaczników XML na format danych JSON.

## 2. OPIS FUNKCJONALNOŚCI:

Projekt jest implementowany w języku C++ 11. Program ma za zadanie wczytać plik XML i wygenerować plik JSON na podstawie XML bez utraty danych. W celu wykonania takiej translacji dokonywana jest analiza leksykalna i rozbiór składniowy pliku XML poprzez wyróżnienie tokenów oraz zbudowania na ich podstawie drzew rozbioru.

### SPECYFIKACA I SKLADNIA JEZYKA XML

Zakładamy, że przetwarzane pliki XML są “Dobrze odformatowanymi”("Well Formed") dokumentami.

Plik XML opisuje zbiór danych. Plik XML składa się z elementów.

Atomemami elementu XML może być:

* tekst
* atrybuty
* inne elementy
* kombinacje wszystkich trzech

Zasady nazywnictwa elementów XML(nazw elementów XML):

* case-sensitive
* powinne się zaczynać od litery lub podkreślenia
* nie mogą się zaczynać od ciągu xml ( XML, Xml, etc)
* mogą mieścić litery, cyfry, podkreślenia, myślniki I kropki
* nie mogą mieścić space
* użycie dwukropka uzasadnione tylko przy korzystaniu z przestrzeni nazw

Dokument XML ma drzewiastą structure, ktora musi zaczynać się w jednym elemencie “korzeniu”

### Struktura drzewa XML



Przykładowy plik XML dla powyższego drzewa:

<?xml version="1.0" encoding="UTF-8**"**?>  
<bookstore>  
  <book category="cooking">  
    <title lang="en">Everyday Italian</title>  
    <author>Giada De Laurentiis</author>  
    <year>2005</year>  
    <price>30.00</price>  
  </book>  
  <book category="children">  
    <title lang="en">Harry Potter</title>  
    <author>J K. Rowling</author>  
    <year>2005</year>  
    <price>29.99</price>  
  </book>  
  <book category="web">  
    <title lang="en">Learning XML</title>  
    <author>Erik T. Ray</author>  
    <year>2003</year>  
    <price>39.95</price>  
  </book>  
</bookstore>

Syntaks dokumentu XML:

* 1. xml prolog <?xml version="1.0" encoding="UTF-8**"**?> - deklaracja opcjonalna
  2. xml korzeń (root element)
  3. tagi , atrybuty XML Wartości atrybutów muszą być zacytowane.

**Problem Elementów I Atrybutów XML**

Rozpatrzmy następujący przykład:

<person gender="female">  
  <firstname>Anna</firstname>  
  <lastname>Smith</lastname>  
</person>

<person>  
  <gender>female</gender>  
  <firstname>Anna</firstname>  
  <lastname>Smith</lastname>  
</person>

W obu przypadkach dostajemy taką samą informacje. Nie ma żadnych zobowiązań i reguł na użycie któregoś z przypadków.

**Unikanie atrybutów XML**

Używając atrybutów do opisu danych musimy uważać na następujące ograniczenia:

* attributy nie mogą mieścić kilku elementów (elementy mogą)
* attributy nie mogą zawierać drzewiastej struktury(elementy mogą)
* atrybudy są trudno rozszerzalne w przyszlości

**Atrybuty XML uzywane dla Metadanych**

Najlepiej trzymać się konwencji:

- Używać elementów do opisu danych

- Używać atrybutów do opisu metadanych(danych o danych)

Naprzykład:

<messages>  
  <note id="501">  
    <to>Tove</to>  
    <from>Jani</from>  
    <heading>Reminder</heading>  
    <body>Don't forget me this weekend!</body>  
  </note>  
  <note id="502">  
    <to>Jani</to>  
    <from>Tove</from>  
    <heading>Re: Reminder</heading>  
    <body>I will not</body>  
  </note>  
</messages>

Atrybut id powyżej jest używany do identyfikaci różnych notatek I nie jest częścią notatki(nie niesię żadnej informacji dla użytkownika – pomocniczy)

**Przestrzeni nazw XML- atrybut xmlns**

Jeżeli wykorzystane zostały prefiksy do nazw XML, to do nich muszą być zdefiniowane odpowiednie przestrzeni nazw(namespaces) za pomocą atrybutu xmlns w następujący sposób:

xmlns:*prefix*="*URI*".

Naprzykład:

<root>  
<h:table xmlns:h="http://www.w3.org/TR/html4/">  
  <h:tr>  
    <h:td>Apples</h:td>  
    <h:td>Bananas</h:td>  
  </h:tr>  
</h:table>  
  
<f:table xmlns:f="http://www.w3schools.com/furniture">  
  <f:name>African Coffee Table</f:name>  
  <f:width>80</f:width>  
  <f:length>120</f:length>  
</f:table>  
  
</root>

Przestrzeni nazw mogą też być zadekłarowane w korzeniu element XML:

<root xmlns:h="http://www.w3.org/TR/html4/" xmlns:f="http://www.w3schools.com/furniture">  
<h:table>  
  <h:tr>  
    <h:td>Apples</h:td>  
    <h:td>Bananas</h:td>  
  </h:tr>  
</h:table>  
<f:table>  
  <f:name>African Coffee Table</f:name>  
  <f:width>80</f:width>  
  <f:length>120</f:length>  
</f:table>  
</root>

Wartośc przestrzeni nazw nie musi być URLem I może być dowolna.

## 

## Specyfikacja i skladnia JSON

JSON z kolei jest oparty o dwie wewnętrzne struktury:

* Kolekcja par nazwa/wartość z unikalnymi nazwami (*associative array*)
* Uporządkowana lista wartości (*array*)

**Syntaksyczne reguły JSONa**

JSON syntaks nasladue notacje obiektową z syntaksu JavaScript:

* Dane są reprezentowane parami: nazwa/wartość
* Dane odseparowane są przecinkami
* Nawiasy klamrowe opakowują obiekty
* Nawiasy kwadratowe opakowują tablice

**Wartości JSON**

Wartościami JSON mogą być:

* numer (integer or floating point)
* string (zacytowany)
* Boolean (true / false)
* tablice (kwadratowe nawiasy)
* obiekty (klamrowe nawiasy)
* null

Jak i JavaScript, obiekty JSON mogą zawierać wiele par nazwa/wartość:

### Przykład:

{"firstName":"John", "lastName":"Doe"}

Tablice jak I w javaScript mogą zawierać wiele obiektów:

### Przykład:

"employees":[  
    {"firstName":"John", "lastName":"Doe"},   
    {"firstName":"Anna", "lastName":"Smith"},   
    {"firstName":"Peter","lastName":"Jones"}  
]

Przykłady rzeczywiście używanych plików JSON można znałeżć tu: <http://www.sitepoint.com/10-example-json-files/>

## 

## 3. WYMAGANIA, OBSLUGA BLĘDÓW

**A. Funkcjonalne**

1. Program przyjmuje nazwę jednego skladniowo poprawnego pliku nazwa.XML(musi być składniowo poprawny zgodnie z <http://www.xmlvalidation.com>) i analizuje go.
2. W pliku nazwa.XML dopuszczalne są wszystkie konstrukcje zdefiniowane w standarcie, które program musi obsługiwać.
3. Program generuje składniowo poprawny plik nazwa.JSON zgodny z danymi pliku nazwa.XML. (sprawdzanie składniowej poprawności na stronie: <http://codebeautify.org/jsonvalidate>)
4. Mapowanie treści XML na JSON będzie przebiegać według przedstawionego dalej schematu:

Pojedynczy element pliku XML może się pojawić na jeden z siedmiu wzorców:

1. Element pusty
2. Element tylko z treścią tekstową
3. Element pusty z atrybutami
4. Element z treściąekstową I atrybutami
5. Element mieszczący w sobie inne elementy z różnymi imionami
6. Element mieszczący w sobie inne elementy identycznymi imionami
7. Element mieszczący w sobie tekst oraz inne elementy

Tablica poniżej pokazuje zasady konwersji 7 wzorców pomiędzy XML a JSON

|  |  |  |
| --- | --- | --- |
| **Wzorzec** | **XML** | **JSON** |
| 1 | <e/> | "e": null |
| 2 | <e>text</e> | "e": "text" |
| 3 | <e name="value" /> | "e":{"@name": "value"} |
| 4 | <e name="value">text</e> | "e": { "@name": "value", "#text": "text" } |
| 5 | <e> <a>text</a> <b>text</b> </e> | "e": { "a": "text", "b": "text" } |
| 6 | <e> <a>text</a> <a>text</a> </e> | "e": { "a": ["text", "text"] } |
| 7 | <e> text <a>text</a> </e> | "e": { "#text": "text", "a": "text" } |

Takie przypisanie obsługuje również przypadki związane z przestrzeniami nazw, które bez probłemu zmożemy potem odtworzyć. (np. nazwa h:name oraz atrybut xmlns:h="http://www.w3.org/1999/xhtml" podpadają pod punkt 3)

Oprócz tego musimy uwzgłędnić nastepujące “opcjonalne” elementy

1. Deklaracja XML(pierwsza linia, np. <?xml version="1.0" encoding="UTF-8" ?> )
2. Instrukcje przetwarzania (np. <?PITarget PIContent?>)
3. Komentarze XML
4. Sekcje CDATA oraz wyrażenia XML (np. &lt, <![CDATA[<sender>John Smith</sender>]]>)

Naprzyklad poniższy plik XML będzie przerobiony na następujący JSON:

<?xml version="1.0"?>

<?xml-stylesheet href="catalog.xsl" type="text/xsl"?>

<!DOCTYPE catalog SYSTEM "catalog.dtd">

<catalog>

<product description="Cardigan Sweater" product\_image="cardigan.jpg">

<catalog\_item gender="Men's">

<item\_number>QWZ5671</item\_number>

<price>39.95</price>

<size description="Medium">

<color\_swatch image="red\_cardigan.jpg">Red</color\_swatch>

<color\_swatch image="burgundy\_cardigan.jpg">Burgundy</color\_swatch>

</size>

<size description="Large">

<color\_swatch image="red\_cardigan.jpg">Red</color\_swatch>

<color\_swatch image="burgundy\_cardigan.jpg">Burgundy</color\_swatch>

</size>

</catalog\_item>

<catalog\_item gender="Women's">

<item\_number>RRX9856</item\_number>

<price>42.50</price>

<size description="Small">

<color\_swatch image="red\_cardigan.jpg">Red</color\_swatch>

<color\_swatch image="navy\_cardigan.jpg">Navy</color\_swatch>

<color\_swatch image="burgundy\_cardigan.jpg">Burgundy</color\_swatch>

</size>

<size description="Medium">

<color\_swatch image="red\_cardigan.jpg">Red</color\_swatch>

<color\_swatch image="navy\_cardigan.jpg">Navy</color\_swatch>

<color\_swatch image="burgundy\_cardigan.jpg">Burgundy</color\_swatch>

<color\_swatch image="black\_cardigan.jpg">Black</color\_swatch>

</size>

<size description="Large">

<color\_swatch image="navy\_cardigan.jpg">Navy</color\_swatch>

<color\_swatch image="black\_cardigan.jpg">Black</color\_swatch>

</size>

<size description="Extra Large">

<color\_swatch image="burgundy\_cardigan.jpg">Burgundy</color\_swatch>

<color\_swatch image="black\_cardigan.jpg">Black</color\_swatch>

</size>

</catalog\_item>

</product>

</catalog>

Odpowiadający JSON:

{

"catalog": {

"product": {

"@description": "Cardigan Sweater",

"@product\_image": "cardigan.jpg",

"catalog\_item": [

{

"@gender": "Men's",

"item\_number": "QWZ5671",

"price": "39.95",

"size": [

{

"@description": "Medium",

"color\_swatch": [

{

"@image": "red\_cardigan.jpg",

"#text": "Red"

},

{

"@image": "burgundy\_cardigan.jpg",

"#text": "Burgundy"

}]},

{

"@description": "Large",

"color\_swatch": [

{

"@image": "red\_cardigan.jpg",

"#text": "Red"

},

{

"@image": "burgundy\_cardigan.jpg",

"#text": "Burgundy"

}]}]},

{

"@gender": "Women's",

"item\_number": "RRX9856",

"price": "42.50",

"size": [

{

"@description": "Small",

"color\_swatch": [

{

"@image": "red\_cardigan.jpg",

"#text": "Red"

},

{

"@image": "navy\_cardigan.jpg",

"#text": "Navy"

},

{

"@image": "burgundy\_cardigan.jpg",

"#text": "Burgundy"

}]},

{

"@description": "Medium",

"color\_swatch": [

{

"@image": "red\_cardigan.jpg",

"#text": "Red"

},

{

"@image": "navy\_cardigan.jpg",

"#text": "Navy"

},

{

"@image": "burgundy\_cardigan.jpg",

"#text": "Burgundy"

},

{

"@image": "black\_cardigan.jpg",

"#text": "Black"

}]},

{

"@description": "Large",

"color\_swatch": [

{

"@image": "navy\_cardigan.jpg",

"#text": "Navy"

},

{

"@image": "black\_cardigan.jpg",

"#text": "Black"

}]},

{

"@description": "Extra Large",

"color\_swatch": [

{

"@image": "burgundy\_cardigan.jpg",

"#text": "Burgundy"

},

{

"@image": "black\_cardigan.jpg",

"#text": "Black"

} ] } ] } ] } }}

**B. Niefunkcjonalne wymagania**

1. Dostarczone pliki XML poprawnie skonstruowane składniowo. W przypadku błędu, komunikat i koniec pracy programu
2. Aplikacja udostępnia prosty, intuicyjny interfejs użytkownika
3. Aplikacja prezentue możliwe konflikty w sposób łatwy do rozstrzygnięcia dla użytkownika. W przypadku pojawienia się problemów aplikacja nadaje możliwość rozstrzygnienia problemu uzytkownikiem.
4. Program nadaje niektóre pomocnicze funkce typu wypisz zbudowane drzewo XML lub podaj liste wszystkich nazw elementów(niepowtarzalną)

## 4. ZAŁOŻENIA PROJEKTOWE I SPOSÓB DZIAŁANIA

1. Uznajemy, że kolejność elementów jednego poziomu pliku XML nie ma znaczenia(wg. ogolnej konwencji przechowywania danych). Czyli dane:   
   <e>  
    <a>1</a>  
    <b>2</b>  
    <a>3</a>  
   </e>  
     
   Oraz:  
   <e>  
    <a>1</a>  
    <a>3</a>  
    <b>2</b>  
   </e>  
   Są równoważne.
2. Z tego wynika że wygenerowane pliki JSON nie będą odwracalne na plik XML semantycznie(te same dane w tym samym porządku), ale nie składniowo(porządek przychowywania nie ma znaczenia). Jest to zgodne z wszystkimi koncepjami przechowywania danych(np. bazodanowe).
3. Problem rozdzielonych danych XML. Rozdzielone dane tekstowe również przenosimy wedlug 7 schematów, jak w przykładzie poniżej:

<e>

some

<a>textual</a>

content

</e>

"e": {

"#text": ["some", "content"],

"a": "textual"

}

4. Program nie sprawdza wszystkich aspektów formalnych specyfikacji XML. Naprzykład program działa poprawnie przy tych samych nazwach atrybutow dla jednego elementu, czy też nie sprawdza obecności potrzebnej deklaracji przestrzeni nazw. Przy takich przypadkach program bezbłędnie będzie generował poprawny plik JSON. Takie rozumowanie wynika z założeń projektowych mianowicie z poprawności pliku wejściowego XML.

5. Program przebiegać będzie w następujący sposób:

* Analiza leksykalna i rozpoznawanie tokenów
* Tworzenie drzewa rozbioru pliku XML na podstawie napotkanych tokenów
* Generacja kodu wynikowego JSON na podstawie drzewa rozbioru przechodząc wgłąb drzewa.

## 5. SPOSÓB URUCHOMIENIA, WE/WY

Wybrany interfejs użytkownika to linia poleceń cmd. Program można uruchomić wpisując polecenie xml2json(trzeba znajdować się w folderze z plikiem xml2json, lub dodać dany folder do zmiennej PATH) w konsoli lub klikając dwukrotnie odpowiedni plik .bat.

Następnie program się pyta o podanie nazwy pliku XML (z pełną ścieżką do niego), który chemy przerobić na plik JSON.

Przy napotkaniu probłemów decyzyjnych wg rostrzygnienia niektórych niejednoznacznych schematów składniowych, program powiadomi o tym użytkownika i da możliwość na wybranie jednej z możliwości przez użytkownika(interfejs tekstowy).

Wynikiem programu jest plik JSON o takiej samej nazwie. Program sygnalizuje użytkownika o swoim pomyślnym zakończeniu jak również o wszyskich możliwych problemach.

## 6. PROJEKT REALIZACJI: OPIS MODUŁÓW I INTERFEJSÓW

## A. Analiza leksykalna

Pierwszym zadaniem programu jest analiza leksykalna tekstu podanego na wejściu plików XML. Tekst jest przeglądany znak po znaku w celu wyróżnienia w nim dpowiednich tokenów.

**1. Tokeny**

Klasa token jest klasą o strukturze która będzie przechowywać nazwe oraz wartość tokenu

Rozróżniane tokeny(terminale gramatyki):

|  |  |
| --- | --- |
| Tekst | Nazwa tokenu |
| <? value ?> | proInsTag |
| <!- - value - - > | CommentTag |
| < | startOpenTag |
| </ | endOpenTag |
| > | closeTag |
| /> | closeEmptyTag |
| <![CDATA[ value ]]> | cdataTag |
| = | equalTag |
| “value” | ‘value’ | attributeValueTag |

Oprócz tego, rozróżniamy tokeny:

nameTag – nazwa elementa lub jego argumena, ENDofFILE, simpleTEXT

# B. Analiza składniowa, semantyczna i Gramatyka w postaci EBNF

**GXML = <Tokeny, Pomocnicze, Produkcje, Plik\_XML>**

Plik\_XML = ***optionals***\*, **element, optionals\***;

**optionals** = proInsTag | commentTag | CdataTag;

**element** = **otwarcie\_elem**, (closeEmptyTag | closeTag, content\*, **zamkniecie\_elem**)

**otwarcie\_elem** = startOpenTag, nazwa, atrybut\*;

**zamkniecie\_elem** = closeOpenTag, nazwa, closeTag;

content = **optionals | simpletext | element;**

**simpletext** = (letters | Digits | symbols)\*;

**nazwa** = [napespace:](letters | '\_' ) (nazwaChar)\*;

nazwaChar = letters , (digits | '.' | '-' | '\_'| letters)\*;

**atrybut** = **nameTag, equalTag, attributeValueTag;**

namespace = letters, (letters | digits)\*;

letters = “A”|”a”|”B”|”b”|”C”|”c” ... “Z”|”z”;

digits = 0|1|2|3|4|5|6|7|8|9;

symbols = “?” | “<” | “>” | “#” | “$” | “%” | … | “\*” | “(“ | “)”;

AnyChar = letters | digits | symbols;

Analiza semantyczna programu głównie połega na kontroli zgodnośći nazw etykiet otwierających i zamykających(za pomocą stosu), kontroli struktury pliku oraz kontroli dopuszczalności nazw(znaki specjalne, cyfry etc.)

|  |
| --- |
|  |

W celu lepszego zrozumienia, powyższa gramatyka jest tylko uproszczeniem całościowej gramatyki XML. Pełna notacja EBNF ze strony W3.org podana w dodatku na końcu dokumentacji.

## C. Działanie programu

Odpowiednio budowane drzewo jest zgodne z zasadą struktury drzewa DOM z tą róznicą, że rozrózniamy atrybuty , teksty i Elementy tylko za ich nazwą.

Przykładowo, poniższy prik XML

<root>

<element> value of text1 </element>

<element> value of text2 </element>

<branch1 atr=”atrvalue”>

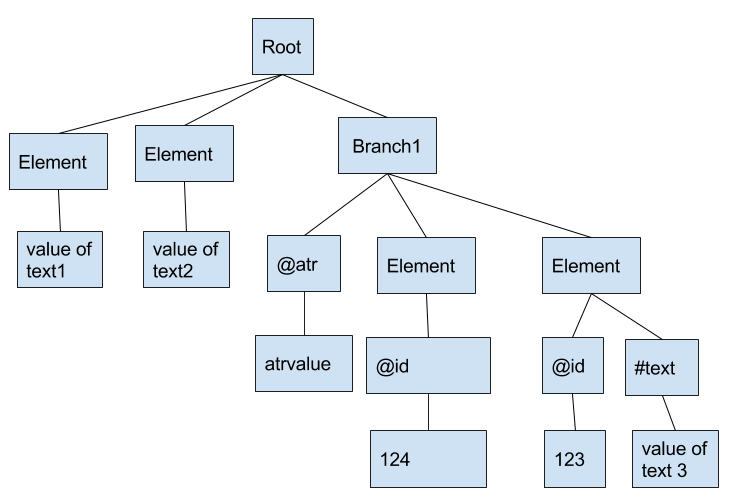
<element id=”123”> value of text3 </element>

<element id=”124” />

</branch1>

</root>

Będzie rozwinięty w następujące drzewo:



Odpowiednio zbudowany plik JSON z drzewa musi wygłądać jak poniżej:

{

"root": {

"element": [

" value of text1" ,

" value of text2"

],

"branch1": {

"@atr": "atrvalue",

"element": [

{

"@id": "123",

"#text": " value of text3 "

},

{ "@id": "124" }

]

}

}

}

Założenia o sposóbie działania nad dodatkowymi danymi:

- Ponieważ format JSON nie pozwala na przechowanie żadnych komentarzy, na dany moment zakładany że komentarze oraz komendy specjalne XML są ignorowane przez nasz translator(na poziomie analizatora leksykalnego).

- Komendy CDATA są wpisywane w całośći do treści value(text)(na poziomie analizatora leksykalnego)

Pod czas analizy leksykalnej oraz rozpoznawania tokenów drzewo dokumentu jest budowane zgodnie z opisanym schematem gramatyki XML.

## D. Podstawowe klasy programu

Wybrane klasy programu:

*//klasa tworzona jest dla jednej instancji pliku XML*

*Class XMLdoc{*

*Private String filename;*

*XMLnode DOMroot;*

*Friend XMLnode;*

*Public:*

*void readFilename();*

*void buildXMLtree();*

*void xml2json ()*

*{*

*DOMroot.writeMe2JSON;*

*}*

*void xmlTreeHelp() //pokazuje uzytkowniku zbudowane drzewo xml*

*{*

*If(DOMroot == NULL) print(„Najpierw zbuduj drzewo XML za pomocą komendy buildXMLtree”)*

*Else ...*

*}*

*void showMenu();*

*}*

Węzel drzewa będzie miał następującą strukture:

Class XMLnode {

String Nazwa;

Bool is\_text;

List <pair< bool processed, \*XMLnode achild>> childElements;

void writeMe2JSON(bool no\_name?);

void next\_child\_with(string name);

void more\_children\_have\_theSameName(string name);

void takeNextChild();

void not\_the\_last\_twin();

void not\_the\_last\_child();

void printJS(String);

}

Prototyp funkcji budowania skladni JSON z drzewa XML(pseudokod):

private void WriteMe2JSON(bool no\_name = false){

//jest wartością lub tekstem

if(childElements == NULL && is\_text) printJS(„+Nazwa+”)

else{

printJS(“{“);

If(!no\_name) printJS(Nazwa+”:”);

if(childElements == NULL) printJS(“null”);

else

While(XMLnode Child = takeNextChild()) {

If(more\_children\_have\_theSameName(Child.nazwa))

{

Print(“{“); Print(“Child.Nazwa”:);

Print([) ;

For(twin = next\_child\_with(Child.Name)

{

Twin.WriteMyJson(no\_name)  
 }

print(“]”); if(not\_the\_last\_twin(Name) print(“,”);

}

Else Child.WriteMe2JSON(); if(not\_the\_last\_child) print(“,”);

} //wszytkie dzieci zostaly obsluzone

Print(“}”)

} //I’m not a leaf

return;

}

Funkcje pomocnicze:

Private void next\_child\_with(string name); - wyszukuje w liscie childElements danego węzła drzewa jeszcze nie uzyte dziecko o nazwie name

Private void more\_children\_have\_theSameName(string name); - zwraca true, jezeli istnieja dwa dziecka o takim samym imieniu.

Private void TakeNextChild(); - zwraca kolejne dziecko o wartosci processed rownej 0.

W celu zapewnienia zgodności nazw elementów, na bieżąco będzie tworzony stos nazw elementów. Ostatnio dodana nazwa za pomocą tagu startOpenTag będzie musiala być zgodna z pierwszą zamykaną nazwa, inaczej – bląd:

List <String> elementNameStack;

## 7. PRZYKLADY TESTOWE

## Program xml2js pozwala na uruchomienie w dwóch trybach:

## Tryb programowy: głównie używany dla szybkiego przetważania i wykorzystywania przez inne programy. Uruchamia się wtedy, gdy programie jako argument zostanie przekazany 1 argument z nazwą pliku XML. Program po wywolaniu wygeneruje odpowiadający plik JSON.

## Tryb interfejsu użytkownika, za który odpowiada klasa UI. Zostanie uruchomiony, gdy programie pod czas wywołania nie zostanie przekazano żadnych argumentów w linii poleceń. Ten tryb pozwala na wielokrotne wczytywanie i anlize plikow XML, a mianowicie pozwala: - wylistować rozpoznane tokeny w kolejności rozpoznania - narysować drzewo rozbioru xml - słedzenie błędów składniowych w razie ich wystąpienia

## Dla automatyzacji testowania programu napisałem skrypt w języku Python, który dla wszystkich plików XML umieszczonych w podanym podkatałogu generuję za pomocą programu xml2js.exe odpowiadające im pliki JSON, po czym sprawdza, czy składnia wygenerowanych plików jest zgodna ze standartem JSON. W razie napotkania błędu program informuję użytkownika w którym miejscu to się stało.

## Pliki XML użyte do testowania zostały podzielone na kilka podzbiorów:

## Wszystkie pliki umieszczone są w katałogu xml, natomiast ich nazwa wskazuje na rodzaj klasy testowej:

## Pliki zaczynające sie od „s\_” (skrót od simple lub small) sprawdzają poprawność przetwarzania podstawowych konstrukcji:

|  |  |  |
| --- | --- | --- |
| Plik | Kod XML | Kod JSON |
| s\_1 | <root/> | { "root": null } |
| s\_2 | <root></root> | { "root": null } |
| s\_3 | <root art="val1"></root> | {"root": { "@art": "val1" }} |
| s\_4 | <root><atr></atr></root> | { "root": { "atr": null }} |
| s\_5 | <root><child atr="val">text</child></root> | {"root":{"child":{"@atr":"val","#text":"text"}}} |
| s\_6 | <root><atr>text1</atr> <atr>text2</atr></root> | {"root": {"atr": [ "text1","text2"] }} |
| s\_7 | <root><atr>text1</atr> <atr>text2</atr> <else>text3</else></root> | {"root": { "atr": [ "text1", "text2" ], "else": "text3" }} |
| s\_8 | <root ><c a="1" b="2"/>text1<c >text2</c> <else a="1">text3</else></root> | {"root":{"c":[{"@a":"1","@b":"2"},"text2"], "#text":"text1","else":{"@a":"1","#text":"text3"}}} |

## Pliki zaczynające się od u\_ (skrót od units) mieszczą w sobie domieszane elementy typu komentarz lub instrukcja przetwarzania w dowolnych miejscach. Także obecne są pliki w których użyte są przestrzeni nazw, symboły typu &amp i inne elementy dodatkowe, nie przenoszone do notacji JSON.

## Pliki zaczynające się od x\_(skrót od xlarge) są to rzeczywiste przykłady plików XML wzięte z internetu. Rozmiar tych plików jest zazwyczaj większy od plików poprzednich klas.

## Pliki zaczynające się od e\_(skrót od error) są plikamy mieszczącymi różne drobne błędy w plikach xml(chociaż to niezgodne z założeniami, jednak dla pokazania odporności programu)

## Użytkownik może dodać dowolną ilość swoich plików testowych do katalogu.

Konwerter XML na JSON online: <http://www.utilities-online.info/xmltojson/>

Dla otrzymania niektórych dużych plików XML skorzystano ze srony <http://www.xmldatasets.net/>

<ol class="xoxo">

<li>Subject 1

<ol>

<li>subpoint a</li>

<li>subpoint b</li>

</ol>

</li>

<li><span>Subject 2</span>

<ol compact="compact">

<li>subpoint c</li>

<li>subpoint d</li>

</ol>

</li>

</ol>

Odpowiednio wygenerowany JSON:

{"ol": {

"li": [

{

"#text": "Subject 1",

"ol": {

"li": ["subpoint a", "subpoint b"]

}

},

{

"span": "Subject 2",

"ol": {

"@compact": "compact",

"li": ["subpoint c", "subpoint d"] } } ]}}

<span class="vevent">

<a class="url" href="http://www.web2con.com/">

<span class="summary">Web 2.0 Conference</span>

<abbr class="dtstart" title="2005-10-05">October 5</abbr>

<abbr class="dtend" title="2005-10-08">7</abbr>

<span class="location">Argent Hotel, San Francisco, CA</span>

</a>

</span>

{

"span": {

"a": {

"@class": "url",

"@href": "http://www.web2con.com/",

"span": [{

"@class": "summery",

"#text": "Web 2.0 Conference"

}, {

"@class": "location",

"#text": "Argent Hotel, San Francisco, CA"

}]

},

"abbr": [{

"@class": "dtstart",

"title": "2005-10-05",

"#text": "October 5"

}, {

"@class": "dtend",

"title": "2005-10-08",

"#text": "7"

}] }}

## Pełna notacja EBNF dla XML 1.1.

[1] document ::= prolog element Misc\* - Char\* RestrictedChar Char\*  
[2] Char ::= [#x1-#xD7FF] | [#xE000-#xFFFD] | [#x10000-#x10FFFF] /\* any Unicode character, excluding the surrogate blocks, FFFE, and FFFF. \*/  
[2a] RestrictedChar ::= [#x1-#x8] | [#xB-#xC] | [#xE-#x1F] | [#x7F-#x84] | [#x86-#x9F]  
[3] S ::= (#x20 | #x9 | #xD | #xA)+  
[4] NameStartChar ::= ":" | [A-Z] | "\_" | [a-z] | [#xC0-#xD6] | [#xD8-#xF6] | [#xF8-#x2FF] | [#x370-#x37D] | [#x37F-#x1FFF] | [#x200C-#x200D] | [#x2070-#x218F] | [#x2C00-#x2FEF] | [#x3001-#xD7FF] | [#xF900-#xFDCF] | [#xFDF0-#xFFFD] | [#x10000-#xEFFFF]  
[4a] NameChar ::= NameStartChar | "-" | "." | [0-9] | #xB7 | [#x0300-#x036F] | [#x203F-#x2040]  
[5] Name ::= NameStartChar (NameChar)\*  
[6] Names ::= Name (#x20 Name)\*  
[7] Nmtoken ::= (NameChar)+  
[8] Nmtokens ::= Nmtoken (#x20 Nmtoken)\*  
[9] EntityValue ::= '"' ([^%&"] | PEReference | Reference)\* '"' | "'" ([^%&'] | PEReference | Reference)\* "'"  
[10] AttValue ::= '"' ([^<&"] | Reference)\* '"' | "'" ([^<&'] | Reference)\* "'"  
[11] SystemLiteral ::= ('"' [^"]\* '"') | ("'" [^']\* "'")  
[12] PubidLiteral ::= '"' PubidChar\* '"' | "'" (PubidChar - "'")\* "'"  
[13] PubidChar ::= #x20 | #xD | #xA | [a-zA-Z0-9] | [-'()+,./:=?;!\*#@$\_%]  
[14] CharData ::= [^<&]\* - ([^<&]\* ']]>' [^<&]\*)  
[15] Comment ::= '<!--' ((Char - '-') | ('-' (Char - '-')))\* '-->'  
[16] PI ::= '<?' PITarget (S (Char\* - (Char\* '?>' Char\*)))? '?>'  
[17] PITarget ::= Name - (('X' | 'x') ('M' | 'm') ('L' | 'l'))  
[18] CDSect ::= CDStart CData CDEnd  
[19] CDStart ::= '<![CDATA['  
[20] CData ::= (Char\* - (Char\* ']]>' Char\*))  
[21] CDEnd ::= ']]>'  
[22] prolog ::= XMLDecl Misc\* (doctypedecl Misc\*)?  
[23] XMLDecl ::= '<?xml' VersionInfo EncodingDecl? SDDecl? S?'?>'  
[24] VersionInfo ::= S 'version' Eq ("'" VersionNum "'" | '"' VersionNum '"')  
[25] Eq ::= S? '=' S?  
[26] VersionNum ::= '1.1'  
[27] Misc ::= Comment | PI | S  
[28] doctypedecl ::= '<!DOCTYPE' S Name (S ExternalID)? S? ('[' intSubset ']' S?)? '>'  
[28a] DeclSep ::= PEReference | S  
[28b] intSubset ::= (markupdecl | DeclSep)\*  
[29] markupdecl ::= elementdecl | AttlistDecl | EntityDecl | NotationDecl | PI | Comment  
[30] extSubset ::= TextDecl? extSubsetDecl  
[31] extSubsetDecl ::= ( markupdecl | conditionalSect | DeclSep)\*  
[32] SDDecl ::= #x20+ 'standalone' Eq (("'" ('yes' | 'no') "'") | ('"' ('yes' | 'no') '"'))  
(Productions 33 through 38 have been removed.)  
[39] element ::= EmptyElemTag  
[40] STag ::= '<' Name (S Attribute)\* S? '>'  
[41] Attribute ::= Name Eq AttValue  
[42] ETag ::= '</' Name S? '>'  
[43] content ::= CharData? ((element | Reference | CDSect | PI | Comment) CharData?)\*  
[44] EmptyElemTag ::= '<' Name (S Attribute)\* S? '/>'  
[45] elementdecl ::= '<!ELEMENT' S Name S contentspec S? '>'  
[46] contentspec ::= 'EMPTY' | 'ANY' | Mixed | children  
[47] children ::= (choice | seq) ('?' | '\*' | '+')?  
[48] cp ::= (Name | choice | seq) ('?' | '\*' | '+')?  
[49] choice ::= '(' S? cp ( S? '|' S? cp )+ S? ')'  
[50] seq ::= '(' S? cp ( S? ',' S? cp )\* S? ')'  
[51] Mixed ::= '(' S? '#PCDATA' (S? '|' S? Name)\* S? ')\*' | '(' S? '#PCDATA' S? ')'  
[52] AttlistDecl ::= '<!ATTLIST' S Name AttDef\* S? '>'  
[53] AttDef ::= S Name S AttType S DefaultDecl  
[54] AttType ::= StringType | TokenizedType | EnumeratedType  
[55] StringType ::= 'CDATA'  
[56] TokenizedType ::= 'ID' | 'IDREF' | 'IDREFS' | 'ENTITY' | 'ENTITIES' | 'NMTOKEN' | 'NMTOKENS'  
[57] EnumeratedType ::= NotationType | Enumeration  
[58] NotationType ::= 'NOTATION' S '(' S? Name (S? '|' S? Name)\* S? ')'  
[59] Enumeration ::= '(' S? Nmtoken (S? '|' S? Nmtoken)\* S? ')'  
[60] DefaultDecl ::= '#REQUIRED' | '#IMPLIED' | (('#FIXED' S)? AttValue)  
[61] conditionalSect ::= includeSect | ignoreSect  
[62] includeSect ::= '<![' S? 'INCLUDE' S? '[' extSubsetDecl ']]>'  
[63] ignoreSect ::= '<![' S? 'IGNORE' S? '[' ignoreSectContents\* ']]>'  
[64] ignoreSectContents::= Ignore ('<![' ignoreSectContents ']]>' Ignore)\*  
[65] Ignore ::= Char\* - (Char\* ('<![' | ']]>') Char\*)  
[66] CharRef ::= '&#' [0-9]+ ';' | '&#x' [0-9a-fA-F]+ ';'  
[67] Reference ::= EntityRef | CharRef  
[68] EntityRef ::= '&' Name ';'  
[69] PEReference ::= '%' Name ';'  
[70] EntityDecl ::= GEDecl | PEDecl  
[71] GEDecl ::= '<!ENTITY' S Name S EntityDef S? '>'  
[72] PEDecl ::= '<!ENTITY' S '%' S Name S PEDef S? '>'  
[73] EntityDef ::= EntityValue| (ExternalID NDataDecl?)  
[74] PEDef ::= EntityValue | ExternalID  
[75] ExternalID ::= 'SYSTEM' S SystemLiteral | 'PUBLIC' S PubidLiteral S SystemLiteral  
[76] NDataDecl ::= S 'NDATA' S Name  
[77] TextDecl ::= '<?xml' VersionInfo? EncodingDecl S? '?>'  
[78] extParsedEnt ::= TextDecl? content - Char\* RestrictedChar Char\*  
[80] EncodingDecl ::= S 'encoding' Eq ('"' EncName '"' | "'" EncName "'" )  
[81] EncName ::= [A-Za-z] ([A-Za-z0-9.\_] | '-')\*  
[82] NotationDecl ::= '<!NOTATION' S Name S (ExternalID | PublicID) S? '>'  
[83] PublicID ::= 'PUBLIC' S PubidLiteral

Żródło: <http://www.liquid-technologies.com/XML/EBNF1.1.aspx>