**OTML спецификация**

OTML – язык модели дерева объектов.  
  
Это язык описания дерева объектов, который легко читается как человеком, так и машиной. Он абстрактен, гибок и прост, в нем нет привязки к структурам, типам данных и объектам, свойственным языкам программирования. Нет типизации данных и нет массивов. Еще OTML поддерживает комментарии и потоковую обработку.

Цели OTML:  
  
1. Быть легко понятным человеку

2. Быть минималистичным: иметь синтаксис, состоящий из минимального набора правил.

3. Быть максимально абстрактным и гибким, позволяя описывать максимально различные структуры данных

4. Поддерживать потоковую обработку данных

5. Развиваться и легко меняться, иметь систему версий

6. Забыть навсегда про это: { }, это: [ ] и это: ( )

**Синтаксис**

Список служебных символов, используемых в языке:

@ : , # ' "  
и еще tab character, пробел и newline character

OTML состоит из 3-х элементов:

Объект, свойство и значение.

Все объекты начинаются с символа @

Все свойства должны иметь с правой стороны символ **:** , после которого идет значение свойства

Все остальное будет интерпретировано как значение.

<example: 1>

**Иерархия**

За уровень вложенности отвечает символ Tab. Использование пробелов слева от объектов, свойств и значений не допускается.

Левая сторона отвечает исключительно за вложенность и допускает только использование символа Tab. У уровней имеется жесткий порядок, например:

если у родителя 2 отступа, то дочерний элемент должен иметь 3 отступа, если будет 4, то OTML сообщит от об ошибке

**Объект**

Объект может содержать как значения, так и свойства и всегда начинается с префикса @

Значения могут находиться на одной строке с объектом, в этом случае объект отделяется от значений символом :.

Значения отделяются друг от друга при помощи символа ,

<example: 2>

При описании значений в новых строках, символ : можно не использовать, но его использование, в данном случае, не будет считаться ошибкой

<example: 3>

Один tab символ означает, что значения принадлежат объекту "nodeWithValues"

Свойства объекта могут объявляться только с новой строки

<example: 4>

**Свойство**

Свойство всегда заканчивается на :, даже если свойство не содержит значений.

Может содержать одно или несколько значений

<example: 5>

Может содержать другие свойства и объекты.

Объекты и свойства описываются только с новой строки

<example: 6>

Пример описания словаря значений:

<example: 7>

**Значение**

Значение не может иметь дочерних элементов. Это атомарная единица OTML.

Значение не имеет типизации. Использование кавычек при указании значения не является обязательным.

Можно использовать как ' так и "

Когда необходимо использовать кавычки:

1. Необходимо использовать пробельные символы в начале/конце значения

"   value with spaces  " or '  value with spaces  '

2. Необходимо использовать символ , (разделить значений) в значении

"value, with, comma" or 'value, with, comma'

3. Необходимо использовать символ # (символ комментария) в значении

"value with # sharp char" or 'value with # sharp char'

4. Необходимо использовать символ + (многострочное значение) в тексте значения

"value with plus character +" or 'value with plus character +'

5. Необходимо использовать символ @ в начале текста значения

"@ value with at sign" or '@ value with at sign'

6. Вы просто не заморачиваетесь и всегда выделяете значения кавычками.

Экранирование:

Нужно экранировать только 2 символа: ", '

Экранирование происходит при помощи символа \

Сам символ \ не экранируется.

<example: 8>

Символы  " , ' можно не экранировать в следующих случаях:

Символ " находится внутри одинарных кавычек. Пример: '  "  '

Символ ' находится внутри двойных кавычек. Пример: "  '  "

Во всех остальных случаях для кавычек необходимо экранирование

<example: 9>

**Многострочные значения**

В XXXX есть способ представления многострочного текста.

Происходит это путем конкатенации значений со вставкой между ними символа или последовательности символов переноса строки.

Для добавления новой строки в конце значения указывается символ +

следующее значение будет добавлено как новая строка предыдущего значения.

Пример:

<example: 9>

<example: 10>

<example: 11>

<example: 12>

Следующий пример является коллекцией из трёх значений, где первое значение - многострочное,

второе и третье - однострочные

<example: 13>

**Комментарии**

Комментарием является все что находится от символа # до конца строки, если символ #

не находится внутри двойных или одинарных кавычек

<example: 14>

**OtmlParser**

OtmlParser считывает OTML из потока/строки и формирует объектную модель документа.

Для получения экземпляра парсера, необходимо использовать OtmlParserFactory.

Фабрика определяет версию документа OTML и возвращает экземпляр парсера, соответствующего этой версии.

Использование:

var otmlFactory = new OtmlParserFactory ();

//чтение из потока  
using (var stream = File.OpenRead("C:\test.otml"))  
{

var parser = otmlFactory.GetParser(stream);

var result = parser.Parse();

}

//чтение из строки

var text = File.ReadAllText(testDataFile);

var parser = otmlFactory.GetParser(text);

var result = parser.Parse();

**OtmlUnparser**

<nuget ref>

OtmlUnarser сериализует объектную модель документа в OTML текст/поток.  
Для получения экземпляра OtmlUnarser, необходимо использовать OtmlUnparserFactory.  
Фабрика вернет OtmlUnarser для OTML самой свежей версии, либо конкретной версии, которую   
вы укажите. Так указать кодировку, в которой вы хотите получить OTML. Кодировкой умолчанию является UTF-8.

Использование:

var otmlFactory = new OtmlUnparserFactory();

var otmlUnparser = otmlFactory.GetDefaultUnparser();

// or otmlFactory.GetUnparser(new Version(1, 0));

var otmlNodeFactory = otmlUnparser.GetNodeFactory();

var dom = new[]

{

otmlNodeFactory.CreateNode(NodeType.Object, "testObject", new []

{

otmlNodeFactory.CreateNode(NodeType.Property, "testProperty", new[]

{

otmlNodeFactory.CreateValue("test value")

})

})

};

//unparce to stream

using (var stream = File.OpenWrite(@"writeTest.otml"))

{

otmlUnparser.Unparse(dom, stream);

}

//unparce to string

var result = otmlUnparser.Unparse(dom);

**DOM**

Объектная модель документа представляет из себя коллекцию объектов OtmlNode

public interface OtmlNode

{

string Name { get; }

string Value { get; }

NodeType Type { get; }

bool IsMultiline { get; }

IEnumerable<INode> Children { get; }

}  
  
public enum NodeType

{

Object,

Property,

Value

}

Свойства INode:

|  |  |  |  |
| --- | --- | --- | --- |
| Type | NodeType.Object | NodeType.Property | NodeType.Value |
| Name | Имя объекта | Имя свойства | Всегда пустая строка |
| Value | Всегда пустая строка | Всегда пустая строка | Содержит значение |
| IsMultiline | Всегда false | Всегда false | True для многострочных значений, |
| Children | Коллекция дочерних элементов любого NodeType | Коллекция дочерних элементов любого NodeType | Всегда пустая коллекция |

Создавать элементы DOM можно при помощи OtmlNodeFactory, которую можно получить через метод GetNodeFactory() класса OtmlUnparser. Либо написать свои реализации для элементов не нарушая условия вышеуказанной таблицы.

**OtmlSerializer**

<nuget ref>

Дня конвертации .NET объекта в OTML и обратно можно использовать Serializer.

Использование:

<example: 15>