Данная система является аналогом системы XML2Spreadsheet с тем различием, что она оперирует не таблицами Excel, а документами Word.

Система предназначена для формирования сложноструктурированных отчётов в формате документов Word (DOCX), на основе данных, представленных в формате XML.

Система имеет один режим чтения исходных XML-файлов, а именно, DOM. Режим DOM допускает б**о**льшую гибкость в определении структуры отчёта и чтении XML-данных за счёт увеличенных требований к памяти и ресурсам.

Система для своей работы использует следующую (устанавливаемую вместе с системой) библиотеку с открытым исходным кодом:

Apache POI — для разбора и создания DOCX-файлов.

**Необходимые библиотеки**

На Jenkins собирается xml2document.jar (c указанием версии), составляющий весь функционал системы, а также 2 zip-архива:

* examples.zip (с указанием версии) содержит в себе примеры xml и docx – файлов, необходимых для работы системы;
* ExternalLibraries.zip содержит в себе библиотеки Apache POI, которые необходимо положить в classpath проекта для того, чтобы система заработала.

Также на Jenkins выкладывается настоящий Manual для того, чтобы быстро разобраться в работе системы.

**Общий принцип работы**

Общий принцип работы системы следующий:

На вход подаётся:

* XML-файл с исходными данными.
* Шаблон в формате .DOCX, в котором заданы визуальная раскладка и форматирование будущего документа.
* Написанный на языке XML дескриптор, описывающий способ обхода тэгов исходного XML-файла.

На выходе получается файл, соответствующий формату шаблона (также .DOCX).

Система XML2Document используется как Java-библиотека,

Для использования в качестве библиотеки необходимо проимпортировать класс ru.curs.xml2document.XML2Document и запустить статический метод process данного класса. Есть 2 перегруженных варианта этого метода: на строках и на потоках. Ниже предсталены оба.

**XML2Document.process(**String xmlData, String xmlDescriptor,

String temlpate, String output**)**

**XML2Document.process(**InputStream xmlData, InputStream xmlDescriptor,

InputStream temlpate, OutputStream output**)**

Здесь строковые параметры имеют смысл файлов с соответствующими расширениями (расширения указывать обязательно!):

**xmlData** — исходные данные (.xml);

**xmlDescriptor** — дескриптор (.xml), описывающий порядок итерации по исходным данным;

**template** — шаблон отчёта (.docx);

**output** — файл (.docx), в который записывается результирующий отчёт (его можно не создавать, просто указав его имя)

В случае потоков можно предавать в метод следующие параметры:

new FileInputStream(filename.ext) – для входных потоков,

new FileOuputStream(filename.ext) – для выходного потока,

где filename – имя соответствующего файла, как в случае строковых параметров, ext – расширение файла.

**Управляемый обход XML**

Вывод в документ осуществляется следующим образом:

* система определённым образом обходит элементы (тэги) XML-файла с данными (алгоритм обхода управляется настроечным XML-файлом, т.н. файлом-дескриптором),
* в заданные моменты осуществляется копирование фрагментов шаблона в документ результата, при этом поля данных шаблона заполняются информацией в контексте текущего элемента XML-файла с данными.

В процессе обхода XML-файла с данными система может находиться в одном из трёх режимов:

1. Состояние чтения элемента (element)
2. Состояние вывода (output)
3. Состояние итерации (iteration)

## Режим чтения элемента

В момент начала обработки система устанавливает в качестве текущего контекста корневой элемент документа с данными и в качестве режима — режим чтения элемента. В начале обработки система ожидает, что корневым элементом настроечного файла будет тэг вида <element name=”[имя\_корневого\_элемента]”>, т. е. значение атрибута name корневого тэга должно совпадать с именем корневого элемента в файле данных. В противном случае система не осуществит вывода. Т. е., если файл с данными имеет вид

<root>

...

</root>

то настроечный файл обязан иметь вид

<element name="root">

...

</element>

Атрибут name также обязаны иметь все прочие тэги <element>.

В режиме чтения элемента система читает субэлементы тэга <element> настроечного файла. Они могут быть двух типов: <output> и <iteration>. Соответственно, система переходит в режимы вывода или итерации. Субэлементов вида <output> и <iteration> в <element> может быть сколько угодно, и они могут идти в любой последовательности — система обрабатывает их последовательно один за другим.

Для атрибута **name** поддерживаются следующие варианты значений:

1. Прямое указание **имени тэга**. В этом случае интерпретатор переходит к обработке <element> лишь в том случае, если имя тэга в сканируемом файле данных совпадает с указанным в атрибуте.
2. Значение **\*** («звёздочка»). В этом случае для обработки подходит любой тэг в файле данных.
3. Значения **(before)** и **(after)**. Служат для вывода «пролога» и «эпилога» последовательности элементов.

Режим итерации

В режиме итерации система работает следующим образом:

* Запоминается значение контекста текущего элемента данных, чтобы восстановить его после завершения итерации.
* Далее, в зависимости от значения атрибута **index**:
  + Если тэг <iteration> не имеет атрибута index, читаются все субэлементы текущего элемента документа с данными и последовательно каждый из них выставляется в качестве текущего.
  + Если тэг <iteration> имеет атрибут index, читается и выставляется в качестве текущего конкретный субэлемент текущего элемента. В качестве значения атрибута index может быть указано целое число, начиная с нуля.
* После того, как прочитан и установлен очередной текущий элемент, система последовательно читает все субэлементы тэга <iteration>, которые могут быть только типа <element>.
* Если встречается тэг <element> с атрибутом name=”(before)”, то прежде всего обрабатывается родительский элемент данных (это даёт возможность вывести “header” последовательности элементов).
* Если получается так, что значение атрибута name тэга <element> совпадает с именем текущего элемента (или атрибут name установлен в значение '\*'), то система переходит в режим чтения элемента, описанный выше.
* Если встречается тэг <element> с атрибутом name=”(after)”, то после всего обрабатывается родительский элемент данных (это даёт возможность вывести “footer” последовательности элементов).
* После завершения итерации система восстанавливает значение контекста текущего элемента, по субэлементам которого началась итерация.

Т. к. может быть сколько угодно тэгов <iteration> внутри тэга <element> и тэгов <element> внутри тэга <iteration>, это позволяет гибко организовывать сложные обходы файла с данными. Например, если файл с данными имеет структуру

<root>

<a></a>

<a></a>

<b></b>

<a></a>

<a></a>

<b></b>

<b></b>

<a></a>

<b></b>

</root>

— т. е. внутри корневого элемента тэги <a> и <b> идут в произвольном порядке — то для того, чтобы обрабатывать тэги <a> и <b> в той последовательности, как они идут в файле данных, настроечный файл должен иметь вид:

<element name="root">

<iteration>

<element name="a">

</element>

<element name="b">

</element>

</iteration>

</element>

или же вид

<element name="root">

<iteration>

<element name="\*">

</element>

</iteration>

</element>

а для того, чтобы обработать сначала все тэги <a>, а затем все тэги <b> — вид

<element name="root">

<iteration>

<element name="a">

</element>

</iteration>

<iteration>

<element name="b">

</element>

</iteration>

</element>

Для того, чтобы обработать нулевой, а затем первый тэг, вне зависимости от имени этих тэгов, настроечный файл должен иметь вид:

<element name="root">

<iteration index="0">

<element name="\*">

</element>

</iteration>

<iteration index="1">

<element name="\*">

</element>

</iteration>

</element>

Режим вывода

Оказавшись в режиме вывода, система копирует фрагмент шаблона в определённое место результирующего файла и заполняет этот фрагмент данными на основе текущего элемента файла данных. Тэги <output> могут встречаться только внутри тэга <element>, однако их может быть сколько угодно и они могут идти в произвольном порядке вперемешку с тэгами <iteration>. Атрибутом тэга <output> являются

* **templateId** — атрибут, указывающий на фрагмент шаблона, с которого берётся диапазон вывода. Он обязателен, если нет других атрибутов типа **templates**
* **templates** — (новый!)необязательный атрибут, указывающий на ряд фрагментов шаблона, с которого берётся диапазон вывода, следующих по номерам **один за другим**. При этом надо следить, чтобы соответствующие темплейты в шаблоне были! Иначе будет ошибка. Пример использования: **<output** **templates=”1:10” />**

Нежелательно использовать оба атрибута одновременно в одном и том же теге <output> во избежании путаницы в выходном документе!!!

Чтобы смысл этих атрибутов стал более понятным, приведём пример шаблона документа.

Шаблон документа

<template id = “1” type=”paragraph”>

## Заголовок

А я ~{@value} иду, шагаю по Москве. А я **~{@value}** иду, шагаю по Москве. А я иду, шагаю по Москве. А я иду, шагаю по Москве. А я иду, шагаю по Москве. А я иду, шагаю по Москве. А я иду, шагаю по Москве. А я иду, шагаю по Москве. А я иду, шагаю по Москве.

j0090386

А я иду, шагаю по Москве. А я иду, шагаю по Москве. А я иду, шагаю по Москве. А я иду, шагаю по Москве. А я иду, шагаю по Москве.

</template>

<template id = “2” type=”letters”>ddd </template>

<template id = “3” type=”table”>

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Цвет | | Фигура | | |
| Синий | Желтый | Квадрат | Круг | Квадратокруглость |

</template>

<template id = “4” type=”table”>

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 45 | 34 | 345 | 3423 | 234234 |

</template>

<template id = “5” type=”table”>

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Цвет | | Фигура | | |
| Синий | Желтый | Квадрат | Круг | Квадратокруглость |

</template>

<template id = “6” type=”table”>

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | ~{@value} | 34 | 345 | 3423 | 234234 |

</template>

Пояснения

Здесь можно рассмотреть xml теги вида: <template id = “6” type=”table”> и </template>. Они открывают и закрывают секции шаблона, которые будут выводиться в соотвествии с тем, какой templateId указан в режиме вывода. В шаблоне этому значению отвечает арибут id тегов <template>. Особое внимание следует обратить на атрибут type тегов <template>. Он указывает нам, какого именно вида информация содержится в ограничивающих тегах <template>.

Так, при type=”table” там будет содержаться таблица (одна и только одна!), причём конфурация таблицы может быть какой угодно, включая подсветку ячеек и текста. При этом, при последовательном шествии шаблонов таблиц в xml-дескрипторах, таблицы будут сливаться (это будет видно на примере полученного документа).

При type=”paragraph” там будет содержаться несколько (или в крайнем случае один) параграф текста, который может содержать также картинку, а также включать в себя подстветку текста. Каждый параграф оканчивается символом перевода каретки!

При type=”letters” там будет содержаться только текст, не оканчивающийся символом перевода каретки (обратите на это особое внимание!). Если документ пустой, а вы захотите вывести текст с данным типом, то ничего не произойдёт, т.к. ещё раз повторюсь: данный текст не оканчивается символом перевода каретки, символизирующей наличие параграфа, а в данном случае параграфа не создаётся!

Также следует обратить внимание на последовательность символов ~{@value}. Это т.н. placeholder, являющийся X-Path выражением относительно xml-файла с данными. В конечном документе он заменится значением соответствующего атрибута.

Репрезентативный пример

При следующих xml файлах: данных

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<root>

<property value=*"Sergio"*/>

<property value=*"Dellavese"*/>

<property value=*"Fabio"*/>

</root>

и дескриптора

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<element name=*"root"*>

<output templateId=*"2"*/>

<output templateId=*"2"*/>

<iteration>

<element name=*"\*"*>

<output templateId=*"2"*/>

<output templateId=*"2"*/>

<output templateId=*"1"*/>

<output templateId=*"5"*/>

<output templateId=*"6"*/>

<output templateId=*"6"*/>

</element>

</iteration>

</element>

мы получим следующий конечный документ. Обратите внимание на последовательности букв ddd. Это получено благодаря описанному атрибуту letters тегов <template>.

## ddd ddd ddd ddd Заголовок

А я Sergio иду, шагаю по Москве. А я **Sergio** иду, шагаю по Москве. А я иду, шагаю по Москве. А я иду, шагаю по Москве. А я иду, шагаю по Москве. А я иду, шагаю по Москве. А я иду, шагаю по Москве. А я иду, шагаю по Москве. А я иду, шагаю по Москве.

Generated

А я иду, шагаю по Москве. А я иду, шагаю по Москве. А я иду, шагаю по Москве. А я иду, шагаю по Москве. А я иду, шагаю по Москве.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Цвет | | Фигура | | |
| Синий | Желтый | Квадрат | Круг | Квадратокруглость |
| 1 | Sergio | 34 | 345 | 3423 | 234234 |
| 1 | Sergio | 34 | 345 | 3423 | 234234 |

## ddd ddd Заголовок

А я Dellavese иду, шагаю по Москве. А я **Dellavese** иду, шагаю по Москве. А я иду, шагаю по Москве. А я иду, шагаю по Москве. А я иду, шагаю по Москве. А я иду, шагаю по Москве. А я иду, шагаю по Москве. А я иду, шагаю по Москве. А я иду, шагаю по Москве.

Generated

А я иду, шагаю по Москве. А я иду, шагаю по Москве. А я иду, шагаю по Москве. А я иду, шагаю по Москве. А я иду, шагаю по Москве.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Цвет | | Фигура | | |
| Синий | Желтый | Квадрат | Круг | Квадратокруглость |
| 1 | Dellavese | 34 | 345 | 3423 | 234234 |
| 1 | Dellavese | 34 | 345 | 3423 | 234234 |

## ddd ddd Заголовок

А я Fabio иду, шагаю по Москве. А я **Fabio** иду, шагаю по Москве. А я иду, шагаю по Москве. А я иду, шагаю по Москве. А я иду, шагаю по Москве. А я иду, шагаю по Москве. А я иду, шагаю по Москве. А я иду, шагаю по Москве. А я иду, шагаю по Москве.

Generated

А я иду, шагаю по Москве. А я иду, шагаю по Москве. А я иду, шагаю по Москве. А я иду, шагаю по Москве. А я иду, шагаю по Москве.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Цвет | | Фигура | | |
| Синий | Желтый | Квадрат | Круг | Квадратокруглость |
| 1 | Fabio | 34 | 345 | 3423 | 234234 |
| 1 | Fabio | 34 | 345 | 3423 | 234234 |

Итак, как видите, в использовании данной системы нет ничего сложного, тем более, если до этого вы уже пользовались системой XML2Spreadsheet.