Автоматическая генерация AsciidocАвтоматическая генерация Asciidoc

Содержание

[1. Общая схема автоматической генерации документации 1](#__RefHeading___Toc318_3984646613)

[2. Преобразование исходного кода в структурированный формат 3](#__RefHeading___Toc320_3984646613)

[2.1. Информация для документации извлекается из структуры исходного кода 3](#__RefHeading___Toc322_3984646613)

[2.2. Структурированный формат получается как один из результатов исполнения исходного кода 3](#__RefHeading___Toc324_3984646613)

[3. Исходный код сразу представляет собой структурированном формате 4](#__RefHeading___Toc326_3984646613)

[3.1. Пример — генерация документации по структуре базы данных 4](#__RefHeading___Toc328_3984646613)

[4. Шаблонизаторы — превращение структурированного файла в документ 6](#__RefHeading___Toc330_3984646613)

[4.1. Простой пример 7](#__RefHeading___Toc332_3984646613)

[5. Ключевые техники, используемые при генерации документации 9](#__RefHeading___Toc334_3984646613)

[5.1. Борьба с пробельными символами 11](#__RefHeading___Toc336_3984646613)

[5.2. Рекурсия 12](#__RefHeading___Toc338_3984646613)

[5.3. Экранирование и другие операции вставляемыми данными 12](#__RefHeading___Toc340_3984646613)

[6. Выводы 13](#__RefHeading___Toc342_3984646613)

Продолжая начатую тему о различных аспектах использования Asciidoc (на самом деле и других аналогичных форматов) для организации процессов непрерывного документирования, хочу рассмотреть тему автоматический генерации документации.

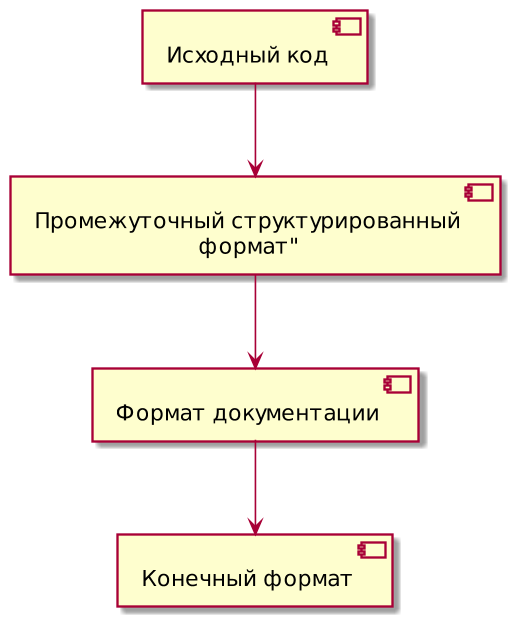
Автоматическая генерация документации — распространенный сегодня термин, но очень расплывчатый. Я понимаю под этим термином извлечение информации, которая уже содержится в исходном коде или настройках программы для представления ее в удобном виде.

# 1. Общая схема автоматической генерации документации

Если рассматривать процесс автоматической генерации как чёрный ящик, то на входе имеем исходный код, а на выходе — собственно документацию или её фрагмент. Однако в реальности при автоматической генерации документации есть еще два промежуточных звена.

* преобразование исходного кода в структурированный формат. Данный шаг обусловлен тем, что для получения документов используется паттерн так называемых шаблонизаторов. В программирование обычно нет единственно возможных подходов, однако это исключение из правил. Все современные технологии, связанные с генерацией пользовательских интерфейсов или человеко-читаемых документов (в том числе, кстати, программного кода), основаны на шаблонизаторах, вопрос только в выборе наилучшего. Вход для шаблонизатора — структура данных, получаемая чаще всего на основе форматов json или xml;
* преобразование структурированного формата в один из форматов документации (обычно Asciidoc, DITA, Docbook, Markdown, reStructureText). За исключением самых простых случаев документация готовится в различных выходных форматах (html, docx, odt, pdf и т.п.) и собирается из разных источников (в том числе не автоматически генерируемых) поэтому целесообразно использовать специальные форматы, для подготовки документации. Предположим, необходимо подготовить документацию по стандартам ЕСКД? Это сама по себе сложная проблема, описана в [предыдущей статье](https://habr.com/ru/post/558940/). Очевидно, при решении проблем автоматической генерации хватает своих проблем, чтобы ещё думать о требованиях ГОСТ.

Таким образом, общая схема генерации документации выглядит следующим образом.



В данной статье мы рассмотрим практические приёмы, которые можно использовать при реализации ИТ-проектов. Для примеров мы будем использовать Asciidoc, однако эти приёмы применимы к любым текстовым маркапам (reStructuredText, Markdown), а также текстовым маркапам для построения диаграмм (рекомендую проект [kroki](https://kroki.io/), который позволяет быстро ознакомиться и даже внедрить наиболее популярные средства построения диаграмм).

# 2. Преобразование исходного кода в структурированный формат

Единых подходов, как исходный код превратить в структурированный формат, не существует, поэтому рассмотрим наиболее частые варианты

## 2.1. Информация для документации извлекается из структуры исходного кода

Как правило, при этом используется дополнительные средства языка, обычно комментарии в специальном формате (комментарии javadoc, ReST и т.п.) и аннотации.

Средств, обеспечивающих преобразование исходного кода в документацию, причём очень зрелых, довольно много. Если есть подходящие для конкретного проекта, можно смела брать и использовать. Однако, разработка собственных средств затратна. Мы пошли указанным путём только один раз, когда разрабатывали [проект для миграции структуры базы данных](https://github.com/CourseOrchestra/2bass). Целесообразность в данном случае определялось тем, что это средство используется во всех наших проектах, ну и желанием попробовать свои силы, конечно, куда без этого.

Следующие описанные подходы обладают большей гибкостью с точки зрения настройки автогенерации документации в конкретных реализуемых проектах.

## 2.2. Структурированный формат получается как один из результатов исполнения исходного кода

Здесь можно использовать следующие подходы:

* создание документации на основе логов работы приложения;
* создание документации на основе считывания состояния объектов (например, структуры базы данных, конфигурации развернутой среды информационной системы и т.п.), которые создало приложение в результате своей работы.

Типовой пример создания документации на основе логов — тесты. Например, есть более менее типовой формат отчётов о выполненных тестах, junit xml report. В этом формате выдают результаты тестирования большинство имеющихся инструментов для тестирования.

В [этой статье](https://habr.com/ru/company/alfa/blog/454720/) показано, как используют json-файлы, которые генерирует при работе [Cucumber](https://cucumber.io/). Также там показано, как документация строится на основе логов, создаваемых в результате работы тестов.

Типовой пример создания документации на основе считывания состояния объектов, создаваемых в результате работы приложения, — документирование структуры БД. В конце данного раздела приведен пример, иллюстрирующий данный подход.

# 3. Исходный код сразу представляет собой структурированном формате

Многие языки уже реализованы в структурированном формате (например, xsd-схемы, open api, различные dsl, создаваемые для описания предметной области, файлы настроек).

Иногда проводят предварительную обработку этих форматов, например, объединение спецификации в единую иерархическую структуру (так называемая операция «flatten»).

Частным (и частым) случаем является ситуация, когда настройки содержатся в базе данных.

## 3.1. Пример — генерация документации по структуре базы данных

Данные пример иллюстрирует достаточно частую ситуацию, когда информация для документации хранится в таблицах СУБД.

Для простоты создадим скрипт, описывающий структуру БД. Этот скрипт не выглядит как исходник для поддержания структуры БД, однако, как это не парадоксально, таковым является, [подробности в документации к уже упомянутому проекту](https://github.com/CourseOrchestra/2bass). На самом деле это может быть миграционный скрипт в любой системе контроля версии базы данных.

create table geo.Strana (  
 id int  
 , naimenovaniye varchar(255)  
 , primary key (id)  
);  
create table geo.Gorod (  
 id int  
 , naimenovaniye varchar(255)  
 , strana\_id int  
 , constraint strana\_gorod foreign key (strana\_id)  
 references geo.Strana(id)  
);

Применим его к базе данных и воспользуемся двумя инструментам СУБД (дальнейший пример приведен для [PostgreSQL](https://www.postgresql.org/)): динамические представления для извлечения сведений о структуре и умение создавать json-файлы на основе результатов сохранения запросов.

drop table if exists fk;  
select x.table\_schema as table\_schema  
 , x.table\_name  
 , y.table\_schema as foreign\_table\_schema  
 , y.table\_name as foreign\_table\_name  
into temp fk  
from information\_schema.referential\_constraints rc  
join information\_schema.key\_column\_usage x  
 on x.constraint\_name = rc.constraint\_name  
join information\_schema.key\_column\_usage y  
 on y.ordinal\_position = x.position\_in\_unique\_constraint  
 and y.constraint\_name = rc.unique\_constraint\_name;  
  
select  
 json\_agg(json\_build\_object(  
 'name', t.table\_schema || '.' || t.table\_name  
 , 'columns'  
 , (select  
 json\_agg(json\_build\_object (  
 'name', column\_name  
 ,'type', data\_type  
 ))  
 from information\_schema.columns as c  
 where c.table\_name = t.table\_name and c.table\_schema = t.table\_schema  
 )  
 , 'fk'  
 , (select  
 json\_agg(json\_build\_object (  
 'fk\_table'  
 , fk.foreign\_table\_schema || '.' || fk.foreign\_table\_name  
 ))  
 from fk  
 where fk.table\_name = t.table\_name and fk.table\_schema = t.table\_schema  
 )  
 ))  
from information\_schema.tables as t  
where table\_schema = 'geo';

В результате получим json-файл следующего содержания:

[{  
 "name": "geo.Strana",  
 "columns": [{  
 "name": "id",  
 "type": "integer"  
 }, {  
 "name": "naimenovaniye",  
 "type": "character varying"  
 }  
 ],  
 "fk": null  
 }, {  
 "name": "geo.Gorod",  
 "columns": [{  
 "name": "id",  
 "type": "integer"  
 }, {  
 "name": "naimenovaniye",  
 "type": "character varying"  
 }, {  
 "name": "strana\_id",  
 "type": "integer"  
 }  
 ],  
 "fk": [{  
 "fk\_table": "geo.Strana"  
 }  
 ]  
 }  
]

В следующей главе будет показано, как данный файл превратить в документ.

# 4. Шаблонизаторы — превращение структурированного файла в документ

Для превращение структурированного файла в документ используется специальный тип языков «Шаблонизаторы». Шаблонизатор позволяет задать правила обхода иерархической структуры данных и правила, по которым элементы иерархии исходного документа преобразуются выходной документ.

Формат этих правил достаточно простой, они безопасны с точки зрения исполнения, поэтому часто шаблонизаторы используются для настройки различных аспектов работы приложений непосредственно пользователями.

Самым известным языком для написания шаблонов (но далеко не самым простым) является [XSLT](https://ru.wikipedia.org/wiki/XSLT). Самым минималистичным — [Mustache](https://mustache.github.io/).

Свой язык написания шаблонов и шаблонизатор также создать довольно просто. Не знаю, по каким причинам, но для создания отчётной системы генерации отчётов в форматах Excel и ods мы пошли [этим путём](https://github.com/CourseOrchestra/xylophone).

Можно вообще обойтись без шаблонизтора, просто структурировать код определенным образом, [в этой старой статье 2003 года](https://martinfowler.com/bliki/MovingAwayFromXslt.html) Мартин Фаулер признается в нелюбви к XSLT и заодно объясняет, как его заменить кодом, написанным на языке Ruby. За 18 лет оказалось, что статические языки также можно прекрасно использовать для этих целей, да и XSLT прекрасно себя чувствует, но и предложенный в статье подход оказался очень хорошо.

Я для примеров будут использовать [Liquid](https://github.com/Shopify/liquid) для работы с JSON и XSLT для работы с XML. В обоих случаях будет использоваться реализация в Ruby, потому что (1) Asciidoc написан на Ruby (2) Ruby-скрипты отлично работают в java и javascript, что в некоторых задачах позволяет не плодить цирк технологий.

## 4.1. Простой пример

Рассмотрим простой пример по генерации документа на основе полученного выше JSON-файла.

Генерация диаграммы в формате PlantUML:

{% assign bl = "\n" %}  
{%- for table in data -%}  
 class {{ table.name }}{{ bl }}  
 {%- for fk in table.fk -%}  
 {{ table.name }} "\*" -- "1" {{ fk.fk\_table }}{{ bl }}  
 {%- endfor -%}  
{%- endfor -%}

В данном пример мы обходим все таблицы. Для каждой таблицы создаем строку PlantUML для отрисовки классов class [Наименование класса]. Далее внутри каждой таблицы проверяем наличие внешних ключей и создаем соединительную линию с соответствующими классами.

На выходе получается следующий текст диаграммы:

class geo.Strana  
class geo.Gorod  
geo.Gorod "\*" -- "1" geo.Strana

Аналогичным образом мы можем сгенерировать документ в Формате Asciidoc:

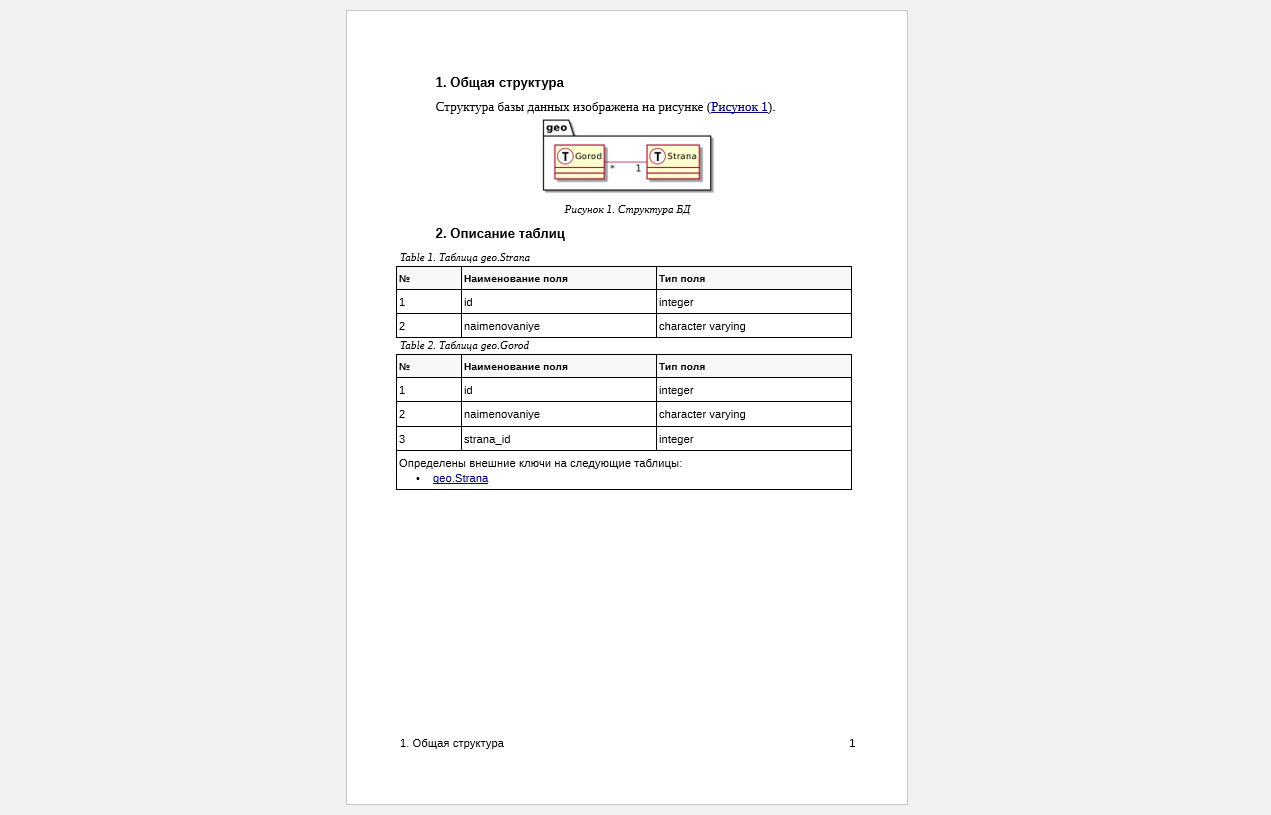
{% assign bl = "\n" %}{% assign bbl = "\n\n" %}  
{%- for table in data -%}  
 [[{{ table.name }}]]{{- bl -}}  
 .Таблица {{ table.name }}{{- bl -}}  
 [cols="1,3,3", options="header"]{{- bl -}}  
 |==={{- bl -}}  
 |№ |Наименование поля |Тип поля{{ bl }}  
 {%- for column in table.columns -%}  
 |{counter:{{ table.name }}} |{{ column.name }} |{{ column.type }}{{- bl -}}  
 {%- endfor -%}  
 {%- if table.fk -%}  
 3+a|Определены внешние ключи на следующие таблицы:{{- bbl -}}  
 {%- for fk in table.fk -%}  
 \* <<{{fk.fk\_table}}, {{fk.fk\_table}}>>{{- bl -}}  
 {%- endfor -%}  
 {%- endif -%}  
 |==={{- bbl -}}  
{%- endfor -%}

Для объединения обоих кусков в один документ воспользуемся директивой include:

= Структура БД  
:lang: ru  
:figure-caption: Рисунок  
:xrefstyle: short  
:sectnums:  
  
== Общая структура  
  
Структура базы данных изображена на рисунке (<<struktura>>).  
  
[[struktura]]  
.Структура БД  
[plantuml, struktura, png, fitrect="170x240mm", srcdpi=300, width="50%"]  
....  
skinparam dpi 300  
left to right direction  
include::pu\_sql.pu[]  
....  
  
== Описание таблиц  
  
include::adoc\_sql.adoc[]

Синтаксис Asciidoc рассмотрен в статье [Asciidoc для ЕСКД](https://habr.com/ru/post/558940/). Подробнее структурирование документации в Asciidoc хотелось бы также рассмотреть в отдельной статье. Здесь лишь хотелось бы отметить, что при вставке диаграммы мы указываем параметры ее отображения. Таким образом, в разных документах одну и ту же диаграмму мы можем отобразить по-разному (цвета, разрешения, ориентация и т.п.).

Результаты я превращаю в файл в формате Microsoft Word с помощью проекта, о котором я рассказывал в [предыдущей](https://habr.com/ru/post/556624/) статье.



# 5. Ключевые техники, используемые при генерации документации

Для рассмотрения ключевых техник приведем другой пример, на основе преобразования теперь уже XML-файла.

Для примера возьмем выписку из ЕГРЮЛ, которую генерирует ФНС. Пример не совсем связан с документацией, однако он удобен для демонстрации основных приёмов преобразования структурированных данных в документацию.

Данные для примера (схема xsd и пример сообщения) взяты на сайте СМЭВ 3 — <https://smev3.gosuslugi.ru/portal/inquirytype_one.jsp?id=41108&zone=fed>.

Исходными данными для преобразования является выписка из ЕГРЮЛ, для примера приведен небольшой её кусок:

<ns1:СвНаимЮЛ НаимЮЛПолн="НАИМЕНОВАНИЕ 5087746429843"  
 НаимЮЛСокр="СОКРАЩЕННОЕ НАИМЕНОВАНИЕ 5087746429843">  
 <ns1:ГРНДата ГРН="5087746429843" ДатаЗаписи="2008-11-18"/>  
</ns1:СвНаимЮЛ>  
<ns1:СвАдресЮЛ>  
 <ns1:АдресРФ КодРегион="77" КодАдрКладр="770000000002990" Дом="7"  
 Корпус="6">  
 <fnst:Регион ТипРегион="ГОРОД" НаимРегион="МОСКВА"/>  
 <fnst:Улица ТипУлица="УЛИЦА" НаимУлица="ФИЛЕВСКАЯ 2-Я"/>  
 <fnst:ГРНДата ГРН="5087746429843" ДатаЗаписи="2008-11-18"/>  
 </ns1:АдресРФ>  
</ns1:СвАдресЮЛ>

Кроме того, для извлечения названий атрибутов используется схема xsd.

Преобразование в формат Asciidoc выглядит следующим образом:

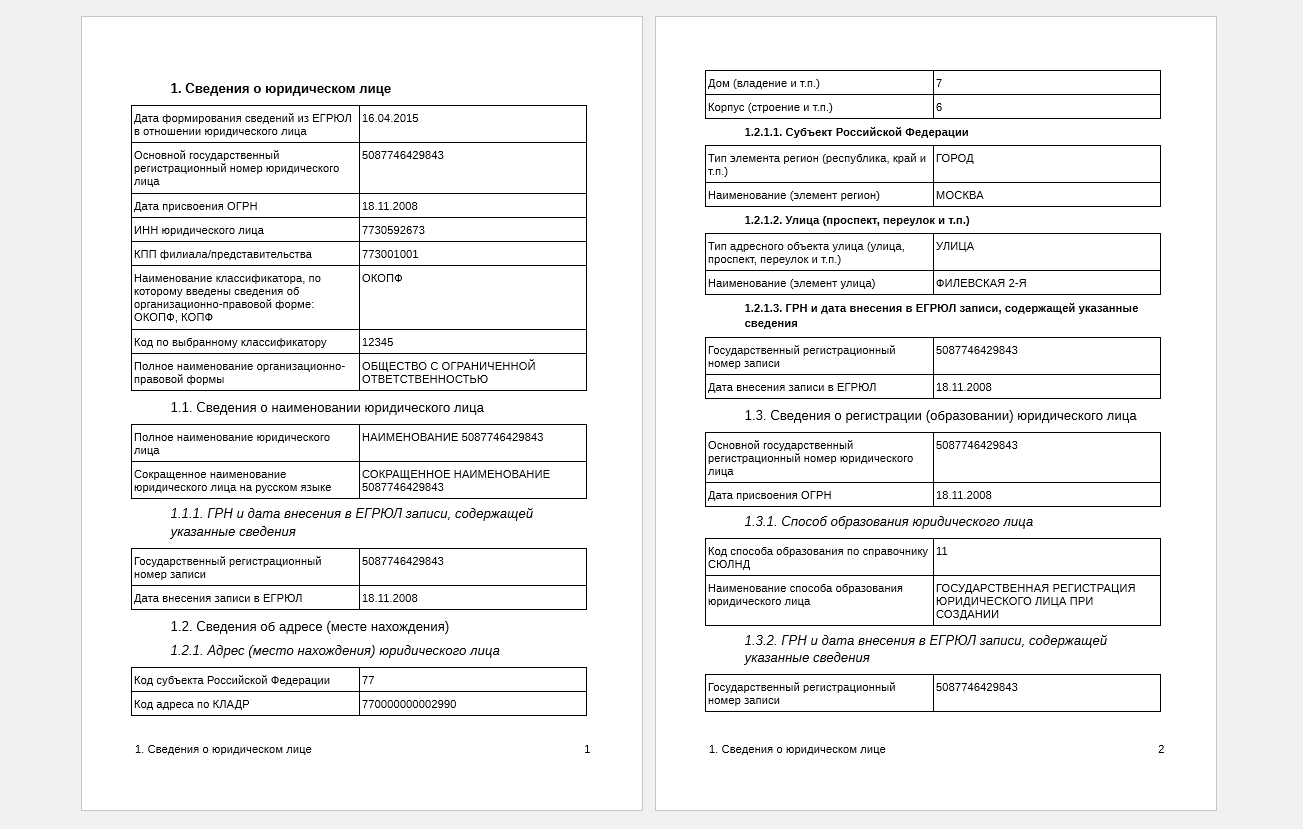
<?xml version="1.0" encoding="UTF-8"?>  
<stylesheet version="1.0" xmlns="http://www.w3.org/1999/XSL/Transform"  
 xmlns:ep="uri:asciidoc:doc:automation"  
 extension-element-prefixes="ep">  
 <output method="text" />  
 <strip-space elements="\*"/>  
 <template match="/">  
 <apply-templates/>  
 </template>  
 <template match="\*[count(@\*|\*) > 0 and count(ancestor::\*) > 0]">  
 <value-of select="'\n='"/>  
 <for-each select="ancestor::\*"><value-of select="'='"/></for-each>  
 <value-of select="' '"/>  
 <value-of select="concat('{',local-name(),'}')"/><text>\n\n</text>  
 <text>|===\n</text>  
 <for-each select="(@\*)|(\*[./text()])">  
 <text>|</text><value-of select="concat('{',local-name(),'}')"/>  
 <text>|</text><value-of select="ep:iformat(current())"/>  
 <text>\n</text>  
 </for-each>  
 <text>|===\n</text>  
 <apply-templates/>  
 </template>  
 <template match="text()"/>  
</stylesheet>

В данном примере обеспечивается простой обход файла с данными ЕГРЮЛ. Тэги, в которых имеются атрибуты или дополнительные тэги трансформируются в заголовок с нужным уровнем иерархии. Атрибуты и текстовые тэги — в строки таблицы.

Наименование тэгов представляются в виде атрибутов Asciidoc, чтобы потом извлечь их значения из xsd-схемы с помощью следующего преобразования:

<?xml version="1.0" encoding="UTF-8"?>  
<stylesheet version="1.0" xmlns="http://www.w3.org/1999/XSL/Transform"  
 xmlns:xs="http://www.w3.org/2001/XMLSchema">  
 <output method="text" />  
 <strip-space elements="\*"/>  
 <template match="\*[@name]">  
 <value-of select="concat(':', @name, ': ')"/>  
 <value-of select="normalize-space(xs:annotation/xs:documentation)"/>  
 <text>\n</text>  
 <apply-templates/>  
 </template>  
 <template match="\*[not(@name)]">  
 <apply-templates/>  
 </template>  
 <template match="text()"></template>  
</stylesheet>

На выходе Microsoft Word дает нам следующую картинку:



## 5.1. Борьба с пробельными символами

Поскольку конечным форматом преобразования является текстовый маркап, то вопрос пробелов является крайне важным. Например, текст смещенный на несколько пробелов воспринимается как блок с моноширинным текстом.

Здесь есть также вопрос эстетики выходного документ. Например, после каждого абзаца в Asciidoc должно быть два переноса строки. Их может быть и три, но читается файл гораздо хуже. Многие автосгенерированные документы напоминают пьяную икоту из «Москвы — Петушки»: количество переносов строк абсолютно не предсказуемы. Особенно это неудобно при сравнении двух версий файла. Если на выходе xml или json, то есть специальный команды, которые делают красивый выходной файл. Для текстовых маркапов, насколько я знаю, такого не существует.

С другой стороны, крайне важно, чтобы сам шаблон был красивым и удобным для редактирования. Т.е., чтобы в нём, как минимум, были отступы в циклах.

Поработав со многими шаблонизаторами, пришёл к выводу, что единственный практически универсальный вариант — это указание шаблонизатору, чтобы он вырезал все пробелы и переносы, а переносы указывать вручную в шаблоне. Например, в приведенном примере указана соответствующая опция <strip-space elements="\*"/> и после каждой выводимой строчки помещена команда <text>\n</text>. Некоторые шаблонизаторы корректно воспринимают \n как символ переноса, однако иногда приходится проводить пост-обработку выходного файла и самостоятельно заменять данную комбинацию на перенос строки.

В примере для Liquid применен аналогичный подход, только для наглядности символ переноса присвоен переменной bl.

## 5.2. Рекурсия

Рекурсия поддерживается большинством шаблонизаторов. XSLT — вообще очень хорошо поддерживают рекурсию через директиву apply-templates. В приведенном примере основной обработчик завершается указанием шаблонизатору пройтись по всем вложенным тэгом и применить соответствующий тэгу шаблон (в данном случае тот же самый).

## 5.3. Экранирование и другие операции вставляемыми данными

Данные, которые вставляются в Asciidoc файл могут вступить в конфликт с разметкой Asciidoc. Например, вы хотите взять текст из Open API спецификации и добавить символ ;. Однако разработчик мог при описании сам поставить тот же символ. В результате в выходной файл попадёт два символа ;; и Asciidoc будет воспринимать получившийся текст как терминологический список, и хорошо ещё, если мы поймём, почему на выходе текст получается отформатирован страрным образом.

Чтобы этого избежать, можно оборачивать вставляемый текст собственными функциями, которые экранируют значения и производят с ними требуемые преобразования. В приведенном примере — это функция iformat. Она добавляет в начале и в конце значения нулевой пробел и, если значения дата, переводит его в формат DD.MM.YYYY.

AsciidocDocAutomation = Class.new do  
 def iformat(node)  
 value = node.to\_s  
 re = /^([0-9]{4})-([0-9]{2})-([0-9]{2})$/  
 vm = value.match(re)  
 value = "#{vm[3]}.#{vm[2]}.#{vm[1]}" if !!(value =~ re)  
 "&#8203;#{value}&#8203;"  
 end  
end

Если необходимо полностью отключить синтаксис Asciidoc во вставляемых значениях, достаточно их просто экранировать.

# 6. Выводы

Технологии автоматическая генерации документации достаточно отработаны и их можно эффективно использовать в ИТ-проектах любого уровня сложности.

И анонс на следующую статью. Она будет посвящена вопросам обеспечения качества документации в формате Asciidoc.