XSD-XML

XML, в переводе с англ. *eXtensible Markup Language* — расширяемый язык разметки. Используется для хранения и передачи данных. Так что увидеть его можно не только в API, но и в коде.

Существует три важные характеристики XML, которые делают его полезным в различных системах и решениях −

- **XML является расширяемым** − XML позволяет создавать собственные самоописательные теги или язык, который подходит для вашего приложения.
- **XML переносит данные, но не представляет их** − XML позволяет хранить данные независимо от того, как они будут представлены.
- **XML является общедоступным стандартом**. XML был разработан организацией под названием World Wide Web Consortium (W3C) и доступен в качестве открытого стандарта.

Сам по себе XML — это язык разметки, чем-то похожий на HTML, который используется на веб-страницах. Но если последний применяется только для вывода информации и её правильной разметки, то XML позволяет её структурировать определённым образом, что делает этот язык чем-то похожим на аналог базы данных, который не требует наличия СУБД.

- **XML** — используется в SOAP *(всегда)* и REST-запросах *(реже)*;
- **JSON** — используется в REST-запросах.

Теги и элементы

XML-файл структурирован несколькими XML-элементами, также называемыми XML-узлами или XML-тегами. Имена XML-элементов заключены в треугольные скобки < >, как показано ниже −

```
<tag>
```

Текст внутри угловых скобок — название тега.

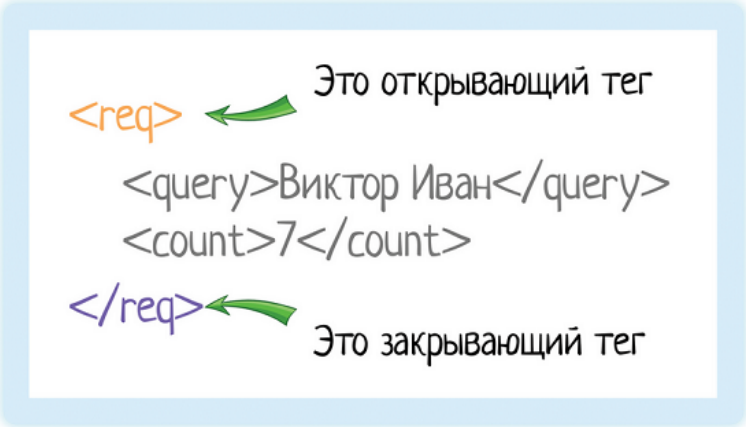
Тега всегда два:

- Открывающий — текст внутри угловых скобок

  ```
  <tag>
  ```

- Закрывающий — тот же текст (это важно!), но добавляется символ «/»

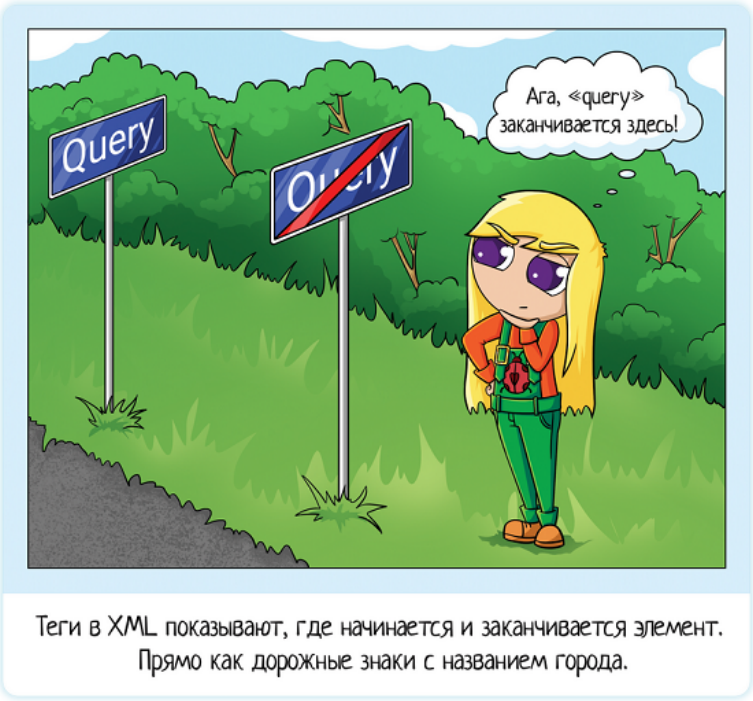
  ```
  </tag>
  ```



С помощью тегов мы показываем системе «вот тут начинается элемент, а вот тут заканчивается». Это как дорожные знаки:

— На въезде в город написано его название: Москва

— На выезде написано то же самое название, но перечеркнутое:~~Москва~~

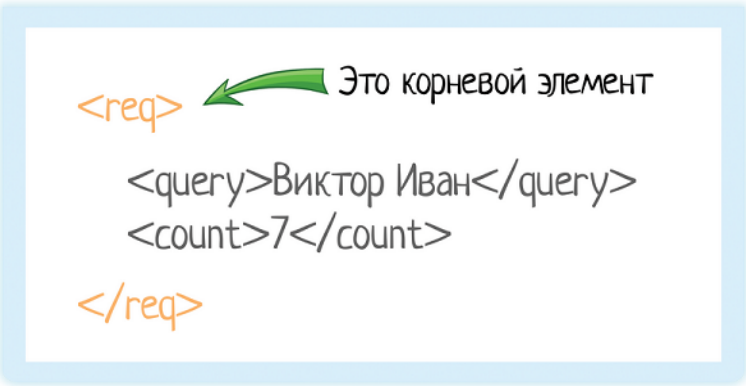


Теги в XML показывают, где начинается и заканчивается элемент. Прямо как дорожные знаки с названием города.

Корневой элемент

В любом XML-документе есть корневой элемент. Это тег, с которого документ начинается, и которым заканчивается. В случае REST API документ — это запрос, который отправляет система. Или ответ, который она получает.

Чтобы обозначить этот запрос, нам нужен корневой элемент. В подсказках корневой элемент — «req».



Он мог бы называться по другому:

```
<main>
```

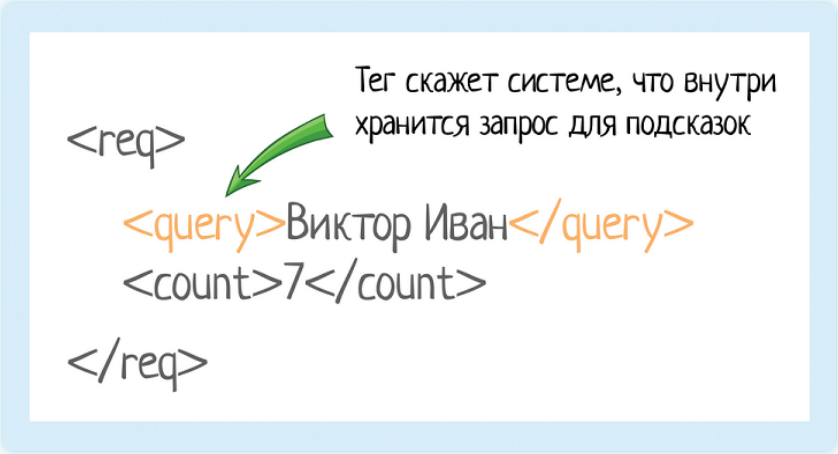
```
<sugg>
```

Да как угодно. Он показывает начало и конец нашего запроса, не более того. А вот внутри уже идет тело документа — сам запрос. Те параметры, которые мы передаем внешней системе. Разумеется, они тоже будут в тегах, но уже в обычных, а не корневых.

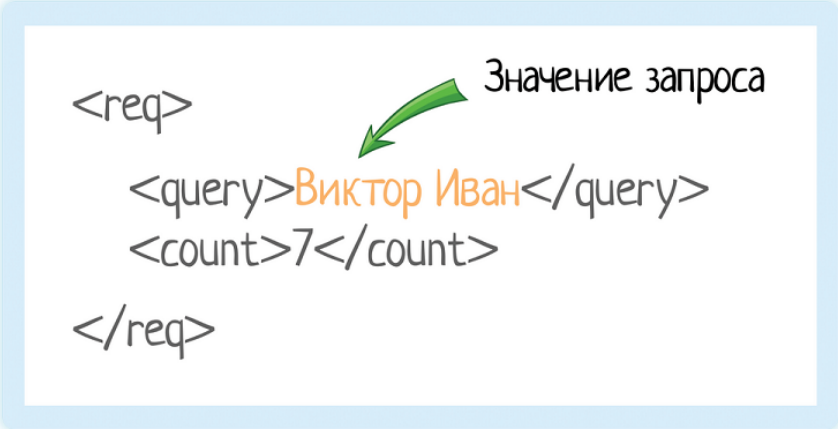
Значение элемента

Значение элемента хранится между открывающим и закрывающим тегами. Это может быть число, строка, или даже вложенные теги!

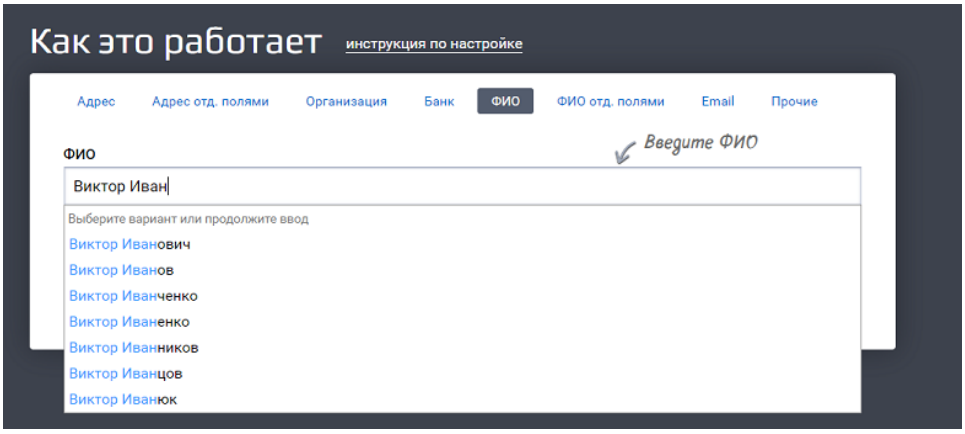
Вот у нас есть тег «query». Он обозначает запрос, который мы отправляем в подсказки.



Внутри — значение запроса.



Это как если бы мы вбили строку «Виктор Иван» в GUI (графическом интерфейсе пользователя):

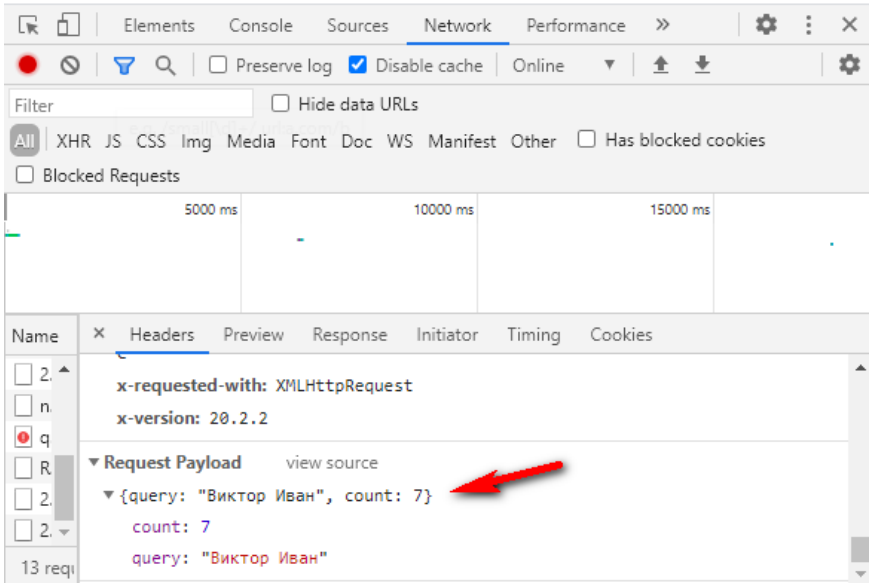


Пользователю лишняя обвязка не нужна, ему нужна красивая формочка. А вот системе надо как-то передать, что «пользователь ввел именно это». Как показать ей, где начинается и заканчивается переданное значение? Для этого и используются теги.

Система видит тег «query» и понимает, что внутри него «строка, по которой нужно вернуть подсказки».

Параметр *count = 7* обозначает, сколько подсказок вернуть в ответе. Если тыкать подсказки на демо-форме Дадаты, нам вернется 7 подсказок. Это потому, что туда вшито как раз значение *count = 7*. А вот если обратиться к документации метода, count можно выбрать от 1 до 20.

> Откройте консоль разработчика через *f12 Network count = 7*



Обратите внимание:

- Виктор Иван — строка
- 7 — число

Но оба значения идут без кавычек. В XML нам нет нужды брать строковое значение в кавычки (а вот в JSON это сделать придется).

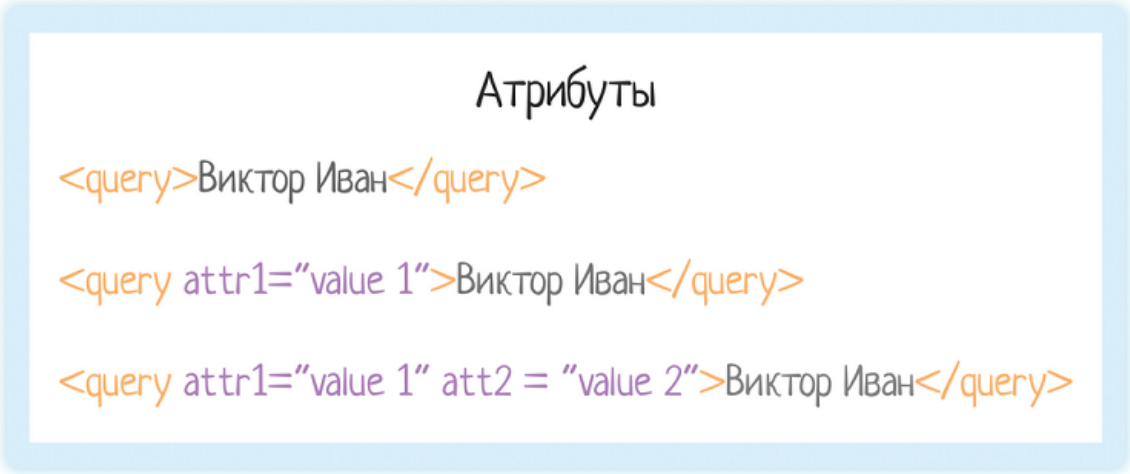
Атрибуты элемента

У элемента могут быть атрибуты — один или несколько. Их мы указываем внутри отрывающегося тега после названия тега через пробел в виде

```
название_атрибута = «значение атрибута»
```

Например:

```
<query attr1=“value 1”>Виктор Иван</query>
<query attr1=“value 1” attr2=“value 2”>Виктор Иван</query>
```



Зачем это нужно? Из атрибутов принимающая API-запрос система понимает, что такое ей вообще пришло.

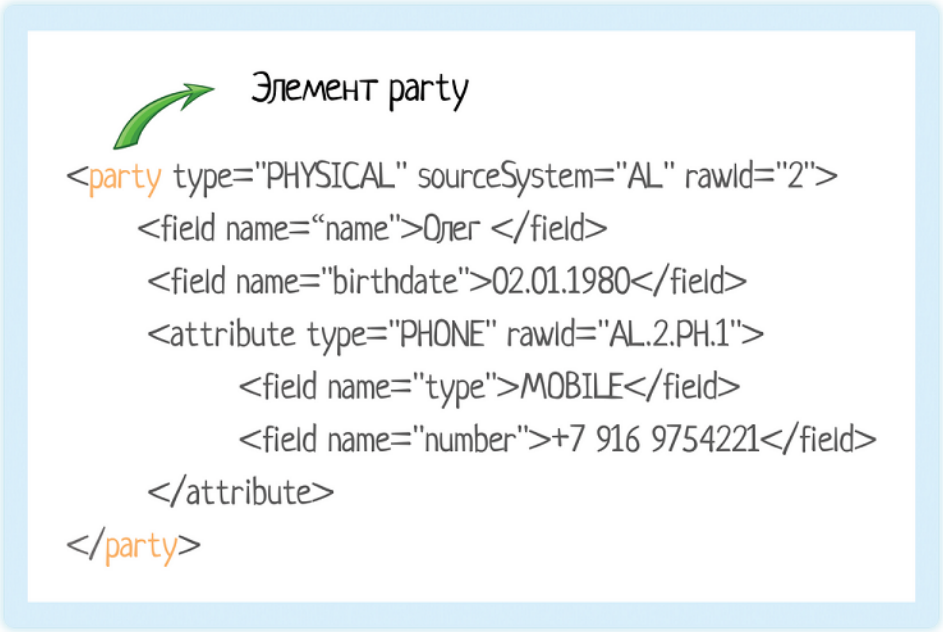
Например, мы делаем поиск по системе, ищем клиентов с именем Олег. Отправляем простой запрос:

```
<query>Олег</query>
```

А в ответ получаем целую пачку Олегов! С разными датами рождения, номерами телефонов и другими данными. Допустим, что один из результатов поиска выглядит так:

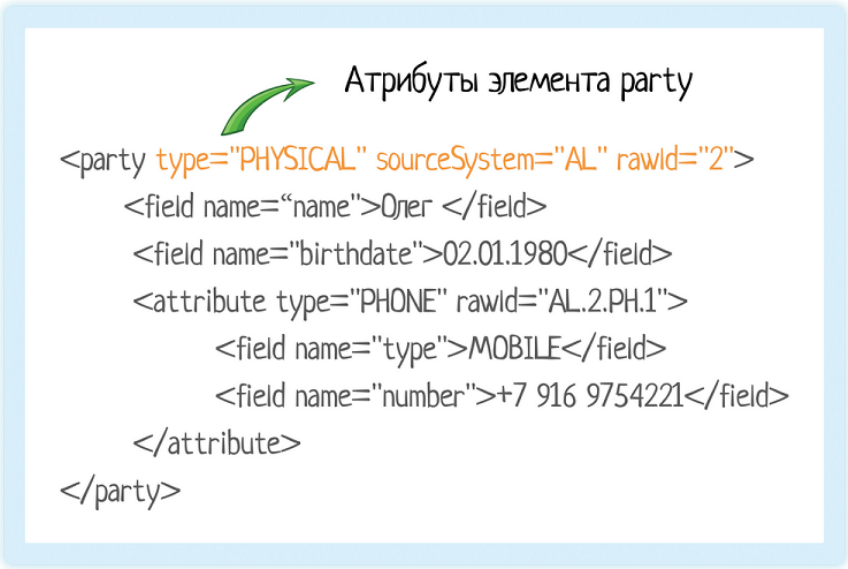
```
<party type="PHYSICAL" sourceSystem="AL" rawId="2">
    <field name="name">Олег </field>
    <field name="birthdate">02.01.1980</field>
    <attribute type="PHONE" rawId="AL.2.PH.1">
        <field name="type">MOBILE</field>
        <field name="number">+7 916 1234567</field>
    </attribute>
</party>
```

Давайте разберем эту запись. У нас есть основной элемент **party**.

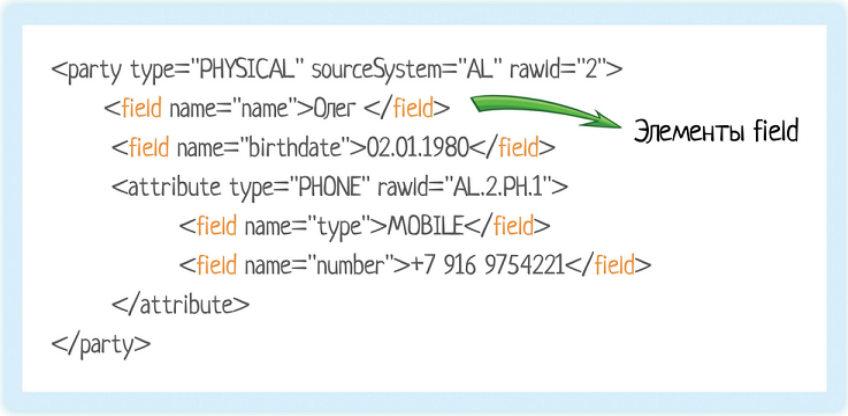


У него есть 3 атрибута:

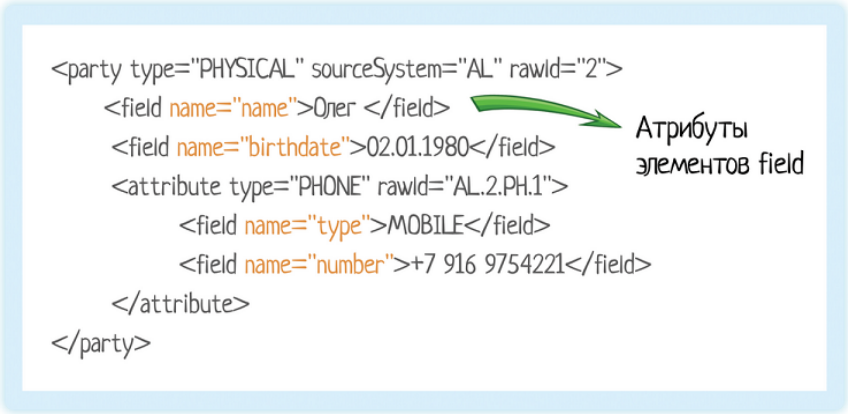
- **type = «PHYSICAL»** — тип возвращаемых данных. Нужен, если система умеет работать с разными типами: ФЛ, ЮЛ, ИП. Тогда благодаря этому атрибуту мы понимаем, с чем именно имеем дело и какие поля у нас будут внутри. А они будут отличаться! У физика это может быть ФИО, дата рождения ИНН, а у юр лица — название компании, ОГРН и КПП
- **sourceSystem = «AL»** — исходная система. Возможно, нас интересуют только физ лица из одной системы, будем делать отсев по этому атрибуту.
- **rawId = «2»** — идентификатор в исходной системе. Он нужен, если мы шлем запрос на обновление клиента, а не на поиск. Как понять, кого обновлять? По связке sourceSystem + rawId!



Внутри **party** есть элементы **field**.



У элементов **field** есть атрибут **name**. Значение атрибута — название поля: имя, дата рождения, тип или номер телефона. Так мы понимаем, что скрывается под конкретным **field**.



Это удобно с точки зрения поддержки, когда у вас коробочный продукт и 10+ заказчиков. У каждого заказчика будет свой набор полей: у кого-то в системе есть ИНН, у кого-то нету, одному важна дата рождения, другому нет, и т.д.

Но, несмотря на разницу моделей, у всех заказчиков будет одна XSD-схема (которая описывает запрос и ответ):

— есть элемент party;

— у него есть элементы field;

— у каждого элемента field есть атрибут name, в котором хранится название поля.

А вот конкретные названия полей уже можно не описывать в XSD. Их уже «смотрите в ТЗ». Конечно, когда заказчик один или вы делаете ПО для себя или «вообще для всех», удобнее использовать именованные поля — то есть «говорящие» теги. Какие плюшки у этого подхода:

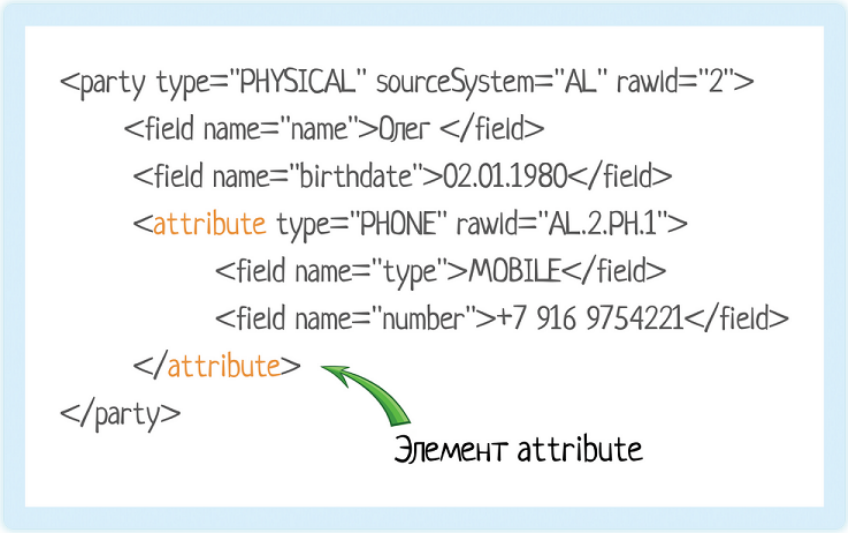
— При чтении XSD сразу видны реальные поля. ТЗ может устареть, а код будет актуален.

— Запрос легко дернуть вручную в SOAP Ui — он сразу создаст все нужные поля, нужно только значениями заполнить. Это удобно тестировщику + заказчик иногда так тестирует, ему тоже хорошо.

В общем, любой подход имеет право на существование. Надо смотреть по проекту, что будет удобнее именно вам. У меня в примере неговорящие названия элементов — все как один будут **field**. А вот по атрибутам уже можно понять, что это такое.

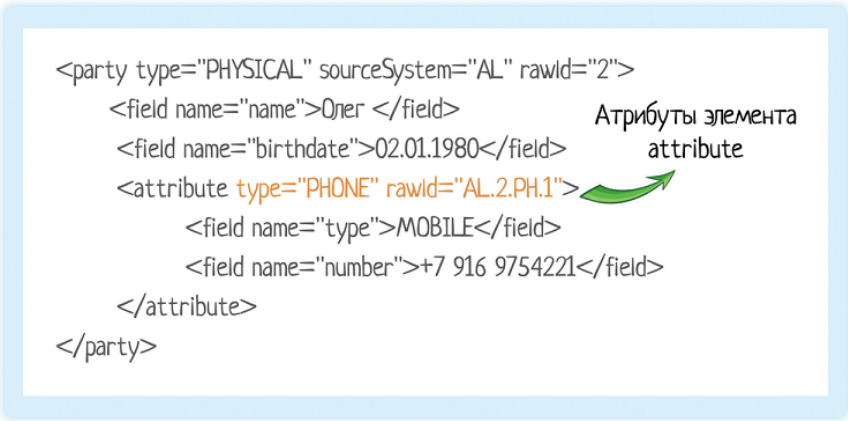
Помимо элементов **field** в party есть элемент **attribute**. Не путайте xml-нотацию и бизнес-прочтение:

- с точки зрения бизнеса это атрибут физ лица, отсюда и название элемента — *attribute*.
- с точки зрения xml — это элемент (не атрибут!), просто его назвали *attribute*. XML все равно (почти), как вы будете называть элементы, так что это допустимо.



У элемента **attribute** есть атрибуты:

- **type = «PHONE»** — тип атрибута. Они ведь разные могут быть: телефон, адрес, емейл...
- **rawId = «AL.2.PH.1»** — идентификатор в исходной системе. Он нужен для обновления. Ведь у одного клиента может быть несколько телефонов, как без ID понять, какой именно обновляется?



Такая вот XML-ка получилась. Причем упрощенная. В реальных системах, где хранятся физ лица, данных сильно больше: штук 20 полей самого физ лица, несколько адресов, телефонов, емейл-адресов…

Но прочитать даже огромную XML не составит труда, если вы знаете, что где. И если она отформатирована — вложенные элементы сдвинуты вправо, остальные на одном уровне. Без форматирования будет тяжеловато…

А так всё просто — у нас есть элементы, заключенные в теги. Внутри тегов — название элемента. Если после названия идет что-то через пробел: это атрибуты элемента.

Типы атрибутов

В следующей таблице перечислены типы атрибутов

Тип атрибута	Описание
StringType	Он принимает любую буквальную строку в качестве значения. CDATA - это StringType . CDATA - это символьные данные. Это означает, что любая строка символов без разметки является законной частью атрибута.
TokenizedType	Это более ограниченный тип. Ограничения допустимости, указанные в грамматике, применяются после нормализации значения атрибута. Атрибуты TokenizedType задаются как – ▪ **ID** – используется для указания элемента как уникального. ▪ **IDREF** – используется для ссылки на идентификатор, который был назван для другого элемента. ▪ **IDREFS** – используется для ссылки на все идентификаторы элемента. ▪ **СУЩНОСТЬ** – указывает, что атрибут будет представлять внешнюю сущность в документе. ▪ **СУЩНОСТИ** – указывает, что атрибут будет представлять внешние сущности в документе. ▪ **NMTOKEN** – он похож на CDATA с ограничениями на то, какие данные могут быть частью атрибута. ▪ **NMTOKENS** – это похоже на CDATA с ограничениями на то, какие данные могут быть частью атрибута.
EnumeratedType	В его объявлении содержится список предопределенных значений. из которых он должен присвоить одно значение. Существует два типа перечисляемых атрибутов – ▪ **NotationType** – объявляет, что элемент будет ссылаться на ОБОЗНАЧЕНИЕ, объявленное где-то еще в документе XML. ▪ **Перечисление** – Перечисление позволяет определить конкретный список значений, которым должно соответствовать значение атрибута.

Правила атрибутов элементов

Ниже приведены правила, которые необходимо соблюдать для атрибутов –

- Имя атрибута не должно появляться более одного раза в одном и том же начальном теге или теге пустого элемента.
- Атрибут должен быть объявлен в определении типа документа (DTD) с использованием объявления списка атрибутов.
- Значения атрибутов не должны содержать прямых или косвенных ссылок на внешние объекты.

- Текст замены любого объекта, на который прямо или косвенно ссылается значение атрибута, не должен содержать знака меньше (**<**)

XSD-схема

XSD (**X**ML **S**chema **D**efinition) — это описание вашего XML. Как он должен выглядеть, что в нем должно быть? Это ТЗ, написанное на языке машины — ведь схему мы пишем… Тоже в формате XML! Получается XML, который описывает другой XML.

Фишка в том, что проверку по схеме можно делегировать машине. И разработчику даже не надо расписывать каждую проверку. Достаточно сказать «вот схема, проверяй по ней».

Если мы создаем SOAP-метод, то указываем в схеме:

- какие поля будут в запросе;
- какие поля будут в ответе;
- какие типы данных у каждого поля;
- какие поля обязательны для заполнения, а какие нет;
- есть ли у поля значение по умолчанию, и какое оно;
- есть ли у поля ограничение по длине;
- есть ли у поля другие параметры;
- какая у запроса структура по вложенности элементов;

Теперь, когда к нам приходит какой-то запрос, он сперва проверяется на корректность по схеме. Если запрос правильный, запускаем метод, отрабатываем бизнес-логику. А она может быть сложной и ресурсоемкой! Например, сделать выборку из многомиллионной базы. Или провести с десяток проверок по разным таблицам базы данных…