#### Язык схем XSD.

#### Статус данного документа

Этот раздел описывает статус документа "XML-схема. Часть 0: Пример" на момент его публикации. Другие документы могут заменить этот документ. Текущий статус этого документа можно уточнить в W3C.

Данный документ был рассмотрен членами W3C, другими заинтересованными сторонами и утвержден Директором в качестве Рекомендации W3C. Документ является окончательным и может использоваться как материал для ссылки и цитирования в других документах. Участие W3C в продвижении представленной Рекомендации заключается в привлечении к ней внимания и способствовании ее широкому распространению. Тем самым наращиваются функциональные возможности, и повышается степень универсальности Сети.

Документ разработан W3C рабочей группой по XML-схеме как часть XML-направления. Назначение языка XML-схемы рассмотрено в документе "Требования к XML-схеме" (<http://www.w3.org/TR/NOTE-xml-schema-req>). Авторы этого документа - члены рабочей группы по XML-схеме. Различные части документа редактировались разными людьми.

Версия данной редакции документа включает в себя изменения, относившиеся к более ранним редакциям.

Пожалуйста, сообщите об ошибках обнаруженных вами в этом документе по email [www-xml-schema-comments@w3.org](mailto:www-xml-schema-comments@w3.org) (архив сообщений находится по адресу <http://lists.w3.org/Archives/Public/www-xml-schema-comments/>). Список известных ошибок доступен по адресу <http://www.w3.org/2001/05/xmlschema-errata>.

Только английская версия этой спецификации является нормативной. Информация о переводах этого документа на другие языки доступна по адресу http:// www.w3.org/2001/05/xmlschema-translations.

Список W3C рекомендаций и другие технические документы можно найти по адресу <http://www.w3.org/TR/>.

#### Введение

Документ, "XML-схема. Часть 0: Пример", являясь учебником для начинающих, обеспечивает легко доступное описание языка XML-схемы, и должен использоваться наряду с формальным описанием языка, содержащегося в документах "XML-схема. Часть 1: Структуры" и "XML-схема. Часть 2: Типы данных". Документ рассчитан на разработчиков программного обеспечения, создающих программы для работы со схемами документов, а также на разработчиков схем документов, желающих ознакомиться с преимуществами языка XML-схем перед языком DTD. Текст документа предполагает, что Вы имеете основные понятия о XML 1.0 (<http://www.w3.org/TR/2000/REC-xml-20001006>) и XML-Namespaces (<http://www.w3.org/TR/1999/REC-xml-names-19990114/>).

Каждый раздел документа вводит новые особенности языка, и описывает их в контексте конкретных примеров:

* раздел 2 раскрывает основные механизмы XML-схемы. Здесь описывается, как объявить элементы и атрибуты, использованные в XML-документах, рассматриваются различия между простыми и комплексными типами, дается определение комплексных типов, рассматривается использование простых типов для описания значений элементов и атрибутов, приводится пример аннотации XML-схемы, рассматривается механизм повторного использования определений элементов и атрибутов. Дается понятие пустого (нуль-) значения;
* раздел 3 содержит описание дополнительных понятий. Здесь рассматривается использование именного пространства (namespaces) в XML-схемах;
* раздел 4 также содержит описание дополнительных понятий. Здесь рассматривается способ создания производных типов из существующих типов. Описывается механизм слияния фрагментов схемы из нескольких источников, и механизм замещения элемента;
* раздел 5 содержит описание дополнительных особенностей, включая описание механизмов для определения уникальности атрибутов и элементов, использования типов в различных именных пространствах, расширения типов, базирующихся на именном пространстве. Рассматривается, каким образом документ может быть проверен на соответствие его схеме.

В дополнение к только что описанным разделам, документ имеет приложения, которые содержат описание простых типов и языка регулярных выражений.

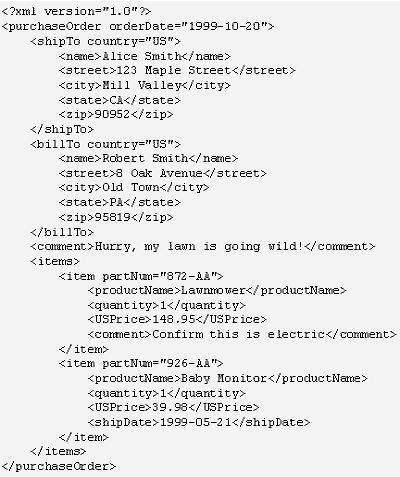
Представленный учебник не является нормативным документом. Это означает, что в нем не содержится четкого описания (с точки зрения W3C) спецификации языка XML-схемы. Примеры и другой пояснительный материал в этом документе предназначены для того, чтобы помочь Вам понимать XML-схему, но они не всегда могут дать полный ответ на то, как использовать те или иные операторы языка. Для удобства, в документе имеется предметный указатель элементов и атрибутов, упомянутых в примерах, и обобщающая таблица с описанием типов данных.

#### 2 Основные понятия: документ "Заказ на покупку"

Назначение XML-схемы состоит в том, чтобы определить класс XML-документов. В связи с этим термин "документ" часто используется для того, чтобы обозначить XML-документ, который соответствует определенной схеме. В действительности схемы документов не существуют сами по себе. Обычно они представляют собой потоки данных, которыми обмениваются между собой приложения, набор тегов в XML-файлах или поля в записях баз данных. Но для упрощения мы будем рассматривать примеры документов и их схемы как просто документы и файлы.

Для начала рассмотрим документ "Заказ на покупку", представленный в файле po.xml. Этот XML-документ описывает заказ, сгенерированный с помощью программ, обеспечивающих формирование заказа на покупку.

Заказ на покупку. Файл po.xml:



Заказ состоит из основного элемента, purchaseOrder, и подэлементов shipTo, billTo, comment и items. Эти подэлементы (кроме comment) в свою очередь содержат другие подэлементы, и так далее, пока не встретится подэлемент типа USPrice. Элемент USPrice содержит число, а не другие подэлементы. Элементы, которые содержат подэлементы или имеют атрибуты, называют элементами комплексного типа, тогда как элементы, которые содержат числа (строки, даты, и т.д.), но не содержат подэлементов или атрибутов, называются элементами простого типа. Как видно из примера, некоторые элементы имеют атрибуты. Атрибуты всегда представляют собой элементы простого типа.

В приведенном примере документа, комплексные типы, и некоторые простые типы, определены в его XML-схеме. Вместе с тем в документе имеются простые типы, которые представляют собой типы, встроенные в язык XML-схем.

Прежде чем продолжить исследование заказ на покупку, сделаем небольшое отступление, чтобы рассмотреть связь между ним и его схемой. Как вы можете заметить, в приведенном выше тексте заказа нет упоминания о его схеме. Это связано с тем, документ не обязан иметь ссылку на свою схему, хотя очень часто такая ссылка может присутствовать. Для начала мы упростим задачу, и не будем рассматривать то, каким образом обработчик документа получает ссылку на его схему. Позднее мы рассмотрим, каким образом устанавливается связь документа с его схемой.

##### 2.1 Схема заказа на покупку

Схема документа "Заказ на покупку" содержится в файле po.xsd.

Схема заказа на покупку. Файл po.xsd:

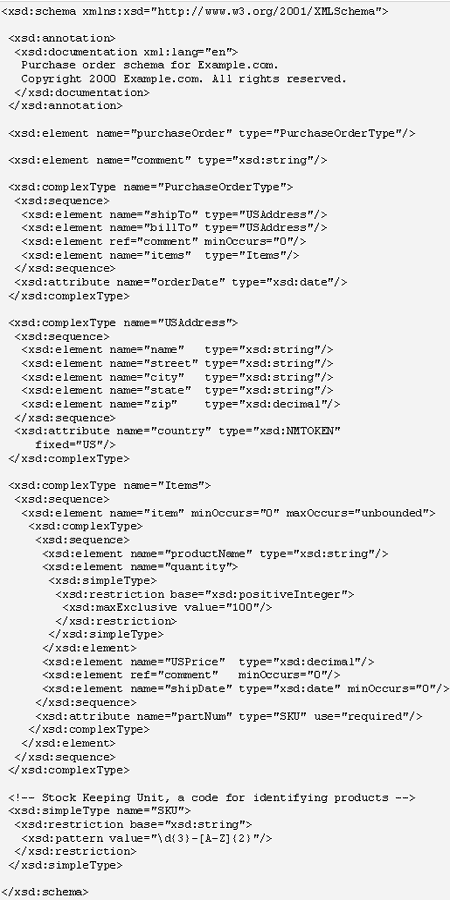


Схема заказа на покупку состоит из элемента schema и множества подэлементов, среди которых наиболее часто упоминаются element, complexType и simpleType. Элементы схемы определяют порядок следования элементов и их содержание в документах типа "Заказ на покупку".

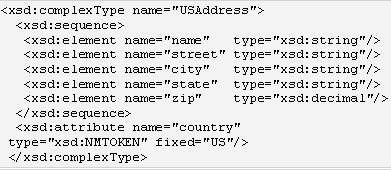
Каждый из элементов в схеме имеет префикс xsd:. Этот префикс связан с именным пространством XML-схемы через объявление xmlns:xsd=http://www.w3.org/2001/XMLSchema, которое задано в элементе schema. Префикс xsd: используется в соответствии с соглашением об использовании этого именного пространства для обозначения элементов XML-схемы, хотя можно использовать любой префикс. Тот же самый префикс, и следовательно, та же самая ассоциация с именным пространством, используется и в названиях встроенных простых типов. Например, xsd:string. Цель ассоциации состоит в том, чтобы идентифицировать принадлежность элементов и простых типов словарю языка XML-схем, а не словарю автора схемы. Для упрощения, мы будем упоминать только названия элементов и простых типов, опуская префикс. Например, simpleType вместо xsd:simpleType.

##### 2.2 Определение комплексных типов, объявление элементов и атрибутов

В XML-схеме есть различие между комплексными типами элементов, которые могут иметь вложенные элементы и атрибуты, и простыми типами, которые не могут иметь вложенных элементов или атрибутов. Также имеется различие между определениями и объявлениями. Определения создают новые типы элементов (простые и комплексные). Объявления задают имена и содержимое элементов и атрибутов (простых и комплексных), которые могут использоваться в документах, соответствующих данной схеме. В этом разделе, мы сосредоточимся на определении комплексных типов и объявлении элементов и атрибутов, которые могут появиться внутри них.

Новые комплексные типы определяются с помощью оператора complexType. Такие определения обычно содержат набор из объявлений элементов, ссылок на элементы, и объявлений атрибутов. Объявления не задают самостоятельно типы. Скорее они создают ассоциации между именем элемента и ограничениями, которые управляют появлением этого имени в документах, соответствующих данной схеме. Элементы объявляются, с помощью оператора element. Атрибуты объявляются, с помощью оператора attribute. В качестве примера рассмотрим определение комплексного типа USAddress. Внутри определения USAddress мы видим пять объявлений элемента и одно объявление атрибута.

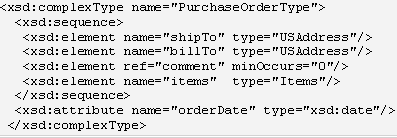
Определение типа USAddress:



В результате этого определения любой элемент типа USAddress, появляющийся в документе (например, элемент shipTo в файле po.xml), должен состоять из пяти элементов и одного атрибута. Имена этих пяти элементов (name, street, city, state и zip) объявляются с помощью атрибута name оператора element, причем элементы должны появиться в той же самой последовательности, в которой они объявлены. Первые четыре из этих элементов будут содержать строковое значение, а пятый - десятичное число. Элемент, тип которого объявляют как USAddress, может появиться с атрибутом country, который должен содержать строковую константу US.

Определение USAddress содержит объявления, включающие только простые типы: string, decimal и NMTOKEN. Напротив, определение PurchaseOrderType содержит объявления элементов, имеющих комплексные типы. Например, USAddress. Хотя оба вида объявлений (простые и комплексные) используют тот же самый атрибут type.

Определение типа PurchaseOrderType:



В определении PurchaseOrderType, объявления элементов shipTo и billTo, связывают различные имена элементов с одним и тем же комплексным типом, а именно с USAddress. Вследствие этого определения любой элемент типа PurchaseOrderType, появляющийся в документе (например, в po.xml), должен состоять из элементов shipTo и billTo. Каждый из этих элементов должен содержать пять подэлементов (name, street, city, state и zip), которые были объявлены в определении USAddress. Элементы shipTo и billTo могут иметь атрибут country, который был объявлен как часть определения USAddress.

Определение PurchaseOrderType содержит объявление атрибута orderDate, который, подобно объявлению атрибута country, задается с помощью простого типа. Фактически, все объявления атрибутов должны выполняться с помощью простых типов, потому что, в отличие от элементов, атрибуты не могут содержать другие элементы или другие атрибуты.

Объявления элементов, которые мы ранее описывали, представляют собой имя, связанное с заданным нами типом элемента. Иногда предпочтительно использовать ссылку на существующий тип элемента, а не объявлять новый, например:

< xsd:element ref="comment" minOccurs="0"/>

В этом объявлении приводится ссылка на существующий элемент comment, который объявлен где-то в другом месте схемы заказа на закупку. Значение атрибута ref должно рассматриваться, как ссылаться на глобальный элемент, который был объявлен в элементе schema, а не как часть определения комплексного типа. Вследствие этого элемент comment может появиться в документе внутри элемента PurchaseOrderType, причем его содержание должно быть совместимо с типом string.

###### 2.2.1 Ограничение вхождений

Значение параметра minOccurs равное 0 у элемента comment говорит о том, что он не обязательно будет присутствовать в составе элемента PurchaseOrderType. Вообще, элемент является обязательным, если значение minOccurs больше или равно 1. Максимальное число появлений элемента определяется значением, задаваемым параметром maxOccurs. Это значение может быть положительным целым числом типа 41, или термом unbounded, что означает отсутствие ограничения максимального числа появлений. Значение по умолчанию для minOccurs и для maxOccurs равно 1. Таким образом, когда элемент типа comment объявлен без maxOccurs , то это означает, что элемент может появиться не более одного раза. Если Вы определяете значение только для minOccurs, то убедитесь что это значение меньше или равно значению по умолчанию maxOccurs. То есть это значение должно быть 0 или 1. Точно так же, если Вы определяете значение только для maxOccurs, то это значение должно быть больше или равно значению по умолчанию minOccurs, то есть 1 или больше. Если оба значения опущены, элемент должен появиться в документе точно один раз.

Атрибуты, в отличие от элементов, могут появиться только однажды или ни разу. Поэтому и синтаксис для определения появления атрибутов отличается от синтаксиса для определения числа появлений элементов. В частности атрибуты могут быть объявлены с параметром use. В зависимости от значения этого параметра атрибут обязателен (use="required"), необязателен (use="optional"), или запрещен (use="prohibited"). Например, объявление атрибута partNum в po.xsd.

Значения по умолчанию и атрибутов и элементов могут быть объявлены с использованием параметра default, хотя этот параметр в том или ином случае работает по разному. Атрибут со значением, определенным по умолчанию, может появляться или не появляться в документе. Если атрибут не появляется в документе, то обработчик схемы обеспечивает, атрибут со значением равным значению default. Обратите внимание, что значения по умолчанию для атрибутов имеют смысл, только если сами атрибуты являются необязательными, поэтому будет ошибкой определить значение по умолчанию вместе с параметром use отличным от use="optional".

Значение по умолчанию для элементов обрабатывается немного по-другому. Если элемент появляется в документе, но не содержит какого либо значения, то в качестве его значения подставляется значение по умолчанию. Однако если элемент не появляется в документе, то обработчик схемы не обеспечивает его значения вообще. В общем, различия между значениями по умолчанию элемента и атрибута в следующем: заданное по умолчанию значение атрибута применяется тогда, когда атрибут отсутствует, а заданное по умолчанию значение элемента применяются тогда, когда элемент присутствует в документе, но не имеет значения (пуст).

Атрибут fixed используется в объявлениях и атрибутов и элементов. Он используется, чтобы указать, что атрибут или элемент могут принимать фиксированные значения. Например, в po.xsd объявлен атрибут country, со значением равным US. Приведенное в примере объявление означает, что атрибут country является необязательным (по умолчанию значение параметра use равно optional), хотя, если атрибут появляется в документе, то его значением должно быть US. Если атрибут country, не появляется в документе, обработчик схемы обеспечит атрибут country со значением US. Обратите внимание, что понятия фиксированного значения и значения по умолчанию являются взаимоисключающими, поэтому объявление не может одновременно содержать атрибуты fixed и default.

Примеры использования указанных выше параметров в объявлениях элементов и атрибутов приведены в таблице 1.

Таблица 1 . Ограничения на появление элементов и атрибутов в документе

|  |  |  |
| --- | --- | --- |
| **Элементы (minOccurs, maxOccurs) fixed, default** | **Атрибуты use, fixed, default** | **Примечание** |
| (1, 1) -, - | required, -, - | элемент/атрибут должен появиться однажды, может иметь любое значение |
| (1, 1) 37, - | required, 37, - | элемент/атрибут должен появиться однажды, его значение должно быть 37 |
| (2, unbounded) 37, - | n/a | элемент должен появиться не менее 2 раз, его значение должно быть 37; minOccurs и maxOccurs могут быть положительными целыми числами. maxOccurs равный unbounded задает неограниченное появление элемента |
| (0, 1) -, - | optional, -, - | элемент/атрибут может появиться однажды, и может иметь любое значение |
| (0, 1) 37, - | optional, 37, - | элемент/атрибут может появиться однажды. Если он появится, то его значение должно быть 37. Если он не появится, то его значение будет 37. |
| (0, 1) -, 37 | optional, -, 37 | элемент/атрибут может появиться однажды. Если элемент не задан, то его значение по умолчанию равно 37, иначе его допустимое значение равно заданному |
| (0, 2) -, 37 | n/a | элемент может появиться однажды, или дважды, или ни разу. Если элемент не появится, то его значение не обеспечено; если появится, и его значение не задано, то по умолчанию его значение будет равно 37; иначе его допустимое значение равно заданному в документе. Вообще значениями, minOccurs и maxOccurs могут быть положительные целые числа. При этом значение maxOccurs может также принимать значение unbounded (не ограничено) |
| (0, 0) -, - | prohibited, -, - | элемент/атрибут не должен появиться в документе |
| Обратите внимание, что ни minOccurs, maxOccurs, ни use не может появиться в объявлениях глобальных элементов и атрибутов. | | |

###### 2.2.2 Глобальные элементы и атрибуты

Глобальные элементы, и глобальные атрибуты, создаются с помощью объявлений, которые являются дочерними элементами элемента schema. Как было указано выше, на глобальный элемент или глобальный атрибут можно сослаться с помощью параметра ref в одном или более объявлениях.

Объявление, которое использует ссылку на глобальный элемент, позволяет элементу, на который указывает ссылка, появиться в документе в качестве содержимого данного объявления. Так, например, элемент comment появляется в po.xml на том же самом уровне, что и shipTo, billTo и элементы items, что связано с тем, что объявление comment находится в комплексном определении типа на том же самом уровне, что и объявления других трех элементов.

Объявление глобального элемента дает возможность элементу появиться в документе в качестве элемента верхнего уровня. Следовательно, purchaseOrder, который объявлен как глобальный элемент в po.xsd, может появиться как элемент верхнего уровня в po.xml. Обратите внимание, что объявление элемента comment как глобального позволяет ему появиться в документе po.xml как элементу верхнего уровня, а не только в качестве подэлемента элемента purchaseOrder.

Есть множество особенностей относительно использования глобальных элементов и атрибутов. Одна из них состоит в том, что глобальные объявления, не могут содержать ссылки: такие объявления должны задавать простые или сложные типы непосредственно. Говоря конкретно, глобальные объявления не могут содержать атрибута ref, они должны использовать атрибут type. Вторая особенность состоит в том, что в глобальных объявлениях нельзя использовать ограничение на количество вхождений элементов, хотя оно может быть помещено в локальные объявления, которые на них ссылаются. Другими словами, глобальные объявления не могут содержать атрибуты minOccurs, maxOccurs, или use.

###### 2.2.3 Конфликты имен

Теперь разберем, как определить новые комплексные типы (см., PurchaseOrderType), объявить элементы (см., purchaseOrder) и объявить атрибуты (см., orderDate). Все эти действия манипулируют с именами, поэтому обычно возникает вопрос, что будет, если объявить два объекта с одинаковыми именами?

Рассмотрим несколько примеров. Если объявить два объекта с одинаковыми именами, но разными типами, то такое объявление создаст конфликтную ситуацию. Например, конфликт имен вызовут комплексный тип с именем USStates и простой тип с именем USStates. Однако если мы определяем комплексный тип с именем USAddress, и объявляем элемент или атрибут с именем USAddress, то конфликт не возникает. Конфликт также не возникает, если элементы с одинаковыми именами объявлены внутри определения различных типов. Например, если мы объявим один элемент как часть типа USAddress, а второй элемент с тем же именем как часть типа Item, то конфликт имен не возникнет. Такие объявления называют локальными. Наконец, если имеется два типа, один из которых определен вами (например, decimal), а второй встроен в язык XML-схемы, то конфликт имен также не возникает. Отсутствие конфликта связано с тем, что эти два типа принадлежат различным именным пространствам. Более подробно использование именных пространств в XML-схемах будет рассмотрено позже.

##### 2.3 . Простые типы

XML-схема заказа на покупку содержит объявления нескольких элементов и атрибутов простого типа. Некоторые из этих простых типов, например string и decimal встроены в язык XML-схемы, в то время как другие определены специально для данного типа документа. Например, атрибут partNum имеет тип по имени SKU (Stock Keeping Unit), который получен из встроенного простого типа string. Встроенные простые типы, и их вариации могут использоваться в объявлениях элементов и атрибутов. В таблице 2 приводится список простых встроенных в язык XML-схемы типов.

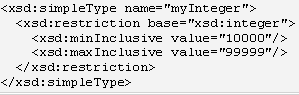
Таблица 2. Список простых типов

|  |  |  |
| --- | --- | --- |
| **Простой тип** | **Примеры (разграничены запятыми)** | **Примечания** |
| string | Confirm this is electric |  |
| normalizedString | Confirm this is electric | см., 3) |
| token | Confirm this is electric | см., 4) |
| byte | -1, 126 | см., 2) |
| unsignedByte | 0, 126 | см., 2) |
| base64Binary | GpM7 |  |
| hexBinary | 0FB7 |  |
| integer | -126789, -1, 0, 1, 126789 | см., 2) |
| positiveInteger | 1, 126789 | см., 2) |
| negativeInteger | -126789, -1 | см., 2) |
| nonNegativeInteger | 0, 1, 126789 | см., 2) |
| nonPositiveInteger | -126789, -1, 0 | см., 2) |
| int | -1, 126789675 | см., 2) |
| unsignedInt | 0, 1267896754 | см., 2) |
| long | -1, 12678967543233 | см., 2) |
| unsignedLong | 0, 12678967543233 | см., 2) |
| short | -1, 12678 | см., 2) |
| unsignedShort | 0, 12678 | см., 2) |
| decimal | -1.23, 0, 123.4, 1000.00 | см., 2) |
| float | -INF, -1E4, -0, 0, 12.78E-2, 12, INF, NaN | 32-х битовое число однократной точности с плавающей запятой. NaN - число не задано, прим. (2) |
| double | -INF, -1E4, -0, 0, 12.78E-2, 12, INF, NaN | 64-х битовое число двойной точности с плавающей запятой, см., 2) |
| boolean | true, false 1, 0 |  |
| time | 13:20:00.000, 13:20:00.000-05:00 | см., 2) |
| dateTime | 1999-05-31T13:20:00.000-05:00 | 31 мая 1999 года 13 часов 20 минут Восточноевропейского стандартного времени, которое на 5 часов отстает от Универсального времени. См.,2) |
| duration | P1Y2M3DT10H30M12.3S | 1 год, 2 месяца, 3 дня, 10 часов, 30 минут, и 12.3 секунды |
| date | 1999-05-31 | см., 2) |
| gMonth | --05-- | Май, см., 2) и 5) |
| gYear | 1999 | 1999, см., 2) и 5) |
| gYearMonth | 1999-02 | Февраль 1999 года, не зависимо от номера дня, см., 2) и 5) |
| gDay | ---31 | 31 день, см., 2) и 5) |
| gMonthDay | --05-31 | 31 мая, см., 2) и 5) |
| Name | shipTo | XML 1.0 тип Name |
| QName | po:USAddress | XML Namespace QName |
| NCName | USAddress | XML Namespace NCName, то есть QName без префикса и двоеточия |
| anyURI | <http://www.example.com/>, [http://www.example.com/ doc.html#ID5](http://www.example.com/doc.html#ID5) |  |
| language | en-GB, en-US, fr | Значения допустимые для xml:lang как определено в XML 1.0 |
| ID |  | XML 1.0 атрибут типа ID, см., 1) |
| IDREF |  | XML 1.0 атрибут типа IDREF, см., 1) |
| IDREFS |  | XML 1.0 атрибут типа IDREFS, см., 1) |
| ENTITY |  | XML 1.0 атрибут типа ENTITY, см., 1) |
| ENTITIES |  | XML 1.0 атрибут типа ENTITIES, см., 1) |
| NOTATION |  | XML 1.0 атрибут типа NOTATION, см., 1) |
| NMTOKEN | US, Bresil | XML 1.0 атрибут типа NMTOKEN, см., 1) |
| NMTOKENS | US UK, Bresil Canada Mexique | XML 1.0 атрибут типа NMTOKENS, то есть список NMTOKEN разделенных пробелами, см., 1) |
| Примечания: 1) Для обеспечения совместимости XML-схем и XML 1.0 DTD, простые идентификаторы типов ( IDREF, IDREFS, ENTITY, ENTITIES, NOTATION, NMTOKEN, NMTOKENS), должны использоваться только в атрибутах. 2) Значение этого типа может быть представлено больше чем одним лексическим форматом. Например, 100 и 1.0E2 - оба значения представлены в формате с плавающей точкой, и значением равным "сто". Однако, для этого типа были установлены правила, которые определяют канонический лексический формат (см., "XML-схема. Часть 2: Типы данных"). 3) Символы перевода строки, табуляции, и перевода каретки в типе normalizedString перед обработкой схемы преобразуются в пробелы. 4) Как и в normalizedString, смежные пробелы сокращены до единственного пробела, предшествующие и завершающие пробелы удаляются. 5) Префикс "g", задает время по Григорианскому календарю. | | |

Новые простые типы можно определить, получая их от существующих простых типов (встроенных или ранее определенных). В частности мы можем получить новый простой тип, ограничивая существующий простой тип. Другими словами, для нового типа мы можем установить собственный диапазон значений как подмножество диапазона значений существующего типа. Для определения имени и типа собственного простого типа используют оператор simpleType. При этом c помощью оператора restriction указывают прототип, и идентифицируют фасеты (параметры), которые ограничивают диапазон значений базового типа. Список фасетов приводится в приложении B.

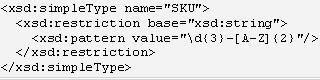
Предположим, что мы хотим создать новый тип целого числа, названного myInteger, чей диапазон значений должен находиться между 10000 и 99999 (включительно). Мы базируем наше определение на встроенном простом типе integer, чей диапазон значений включает так же и целые числа как меньше чем 10000, так и больше чем 99999. Чтобы определить тип myInteger, мы ограничиваем диапазон базового типа integer, используя два фасета, названные minInclusive и maxInclusive.

Определение myInteger. Диапазон 10000-99999:



Приведенный пример показывает использование одной из возможных комбинаций базового типа, и двух фасетов, которые были применены для определения myInteger. Но возможно использование и других комбинаций встроенных простых типов и фасетов (см., приложение B). Рассмотрим более сложный пример определения простого типа. Тип по имени SKU получен из простого типа string. Мы ограничиваем значения SKU путем использования фасета pattern, который содержит регулярное выражение, определяющее допустимый формат строки "\d{3}-[A-Z]{2}". Это выражение читается следующим образом: "строка начинается с трех цифр, за которыми следует дефис, за дефисом следует два ASCII-символами верхнего регистра".

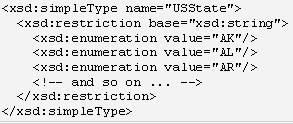
Определение простого типа SKU:



Более полно язык регулярных выражений описан в приложении D.

Язык XML-схем определяет пятнадцать фасетов, которые перечислены в приложении B. Среди них особенно полезен фасет enumeration. Его можно использовать для ограничения значения почти каждого простого типа, кроме boolean. Фасет enumeration ограничивает простой тип набором явных значений. Например, мы можем использовать enumeration, чтобы определить новый простой тип по имени USState, полученный из типа string, значение которого должно быть одним из стандартных сокращений наименования штатов США.

Использование фасета enumeration:



Использование типа USState является хорошей заменой типа string, при использовании в элементе state (содержит наименование штата). За счет такой замены мы можем ограничить значения элемента state в подэлементах billTo и shipTo. При этом эти значения ограничены следующим перечнем: AK, АL, AR, и т.д. Обратите внимание, что перечисляемые значения, указанные для данного типа должны быть уникальны.

###### 2.3.1 Тип List

В дополнение к так называемым атомарным типам, которые составляют большинство, XML-схема имеет понятие списка. Перечень атомарных типов перечислен в таблице 2. Атомарные типы, списочные типы, и типы объединения, описанные в следующем разделе, все вместе называются простыми типами. Использование атомарных типов индивидуализирует используемые значения. Например, NMTOKEN индивидуализирует значение US, делая его неделимым в том смысле, что никакая часть US, типа символа "S", не имеет значения отдельно от целого. Списочные типы состоят из последовательностей атомарных типов, и, следовательно, допустимыми значениями могут быть только "атомы" из этой последовательности. Например, списочный тип NMTOKENS состоит из значений типа NMTOKEN, разделенных пробелами. Например, "US UK FR". Язык XML-схем имеет три встроенных списочных типа: NMTOKENS, IDREFS и ENTITIES.

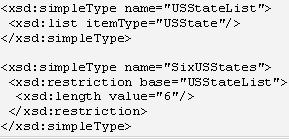
В дополнение к встроенным списочным типам Вы можете создать новые списочные типы из существующих атомарных типов. Невозможно создать списочные типы из существующих списочных типов или из комплексных типов. Например, рассмотрим списочный тип listOfMyIntType, состоящий из значений myInteger:

http://citforum.ru/internet/xml/scheme/8.gif

Элемент в документе, содержимое которого соответствует типу listOfMyIntType, может выглядеть следующим образом:

http://citforum.ru/internet/xml/scheme/9.gif

Для создания списочного типа могут быть применены следующие фасеты: length, minLength, maxLength, и enumeration. Например, чтобы определить список точно из шести штатов США (SixUSStates), мы сначала определяем новый списочный тип (полученный из типа USState) с именем USStateList, а затем создаем тип SixUSStates, ограничивая USStateList только шестью элементами. Списочный тип SixUSStates:



Элементы, тип которых - SixUSStates, должны содержать шесть элементов, и каждый из этих шести элементов должен быть одним из атомарных значений перечислимого типа USState, например:

http://citforum.ru/internet/xml/scheme/11.gif

Обратите внимание, что мы получили списочный тип из элементов атомарного типа string. Однако, тип string может содержать пробелы, а пробелы разграничивают элементы в списочном типе. Поэтому Вы должны быть внимательным, используя списочные типы, исходный тип которых - string. Например, мы определили новый списочный тип на основе базового типа string и фасета length равным 3. Тогда следующие три элемента списка являются законными:

Asie Europe Afrique.

Но следующие три элемента списка незаконны:

Asie Europe Amerique Latine.

Даже учитывая что "Amerique Latine" может существовать вне списка как отдельная строка, когда это значение включено в список, то пробел между Amerique и Latine фактически создает четвертый элемент списка. Поэтому последний пример не будет соответствовать списочному типу с тремя элементами.

###### 2.3.2 Тип Union

Атомарные типы и списочные типы дают возможность элементу или атрибуту принимать значение (одно или более) экземпляра одного атомарного типа. Тип Union дает возможность элементу или атрибуту принимать значение (одно или более) одного типа, образованного путем объединения множества атомарных и списочных типов. Например, создадим union-тип для идентификации штатов США как односимвольного сокращения названия или списка числовых кодов. Рассмотрим тип zipUnion. Он сформирован из одного атомарного типа, и одного списка:

http://citforum.ru/internet/xml/scheme/12.gif

Когда мы определяем union-тип, то атрибут memberTypes оператора union задает список всех типов в объединении. Предположим, что мы объявили элемент с названием zips типа zipUnion, тогда он может принимать следующие значения:

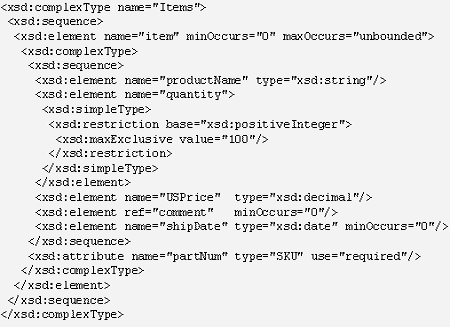
http://citforum.ru/internet/xml/scheme/13.gif

К типу union могут быть применены два фасета: pattern и enumeration.

##### 2.4 Определение анонимных типов

При создании схем применяется два стиля. Схемы могут создаваться путем определения поименованных типов (например, PurchaseOrderType) с последующим объявлением элементов этого типа (например, purchaseOrder). При этом объявленные элементы ссылаются на поименованный тип с помощью конструкции type= . Этот стиль является достаточно простым, но может стать неуправляемым, особенно если Вы определяете много типов, на которые ссылаетесь только один раз, и которые содержат немного ограничений. В этих случаях, тип может быть более кратко определен как анонимный. Анонимный тип нет необходимости именовать и, следовательно, задавать на него ссылки. Определение типа Items в po.xsd содержит два объявления item и quantity, использующие анонимный тип. Наличие в схеме анонимного типа можно идентифицировать за счет отсутствия в объявлении элементов или атрибутов параметра type=, и присутствия непоименованного определения простого или комплексного типа.

Объявление двух анонимных типов:

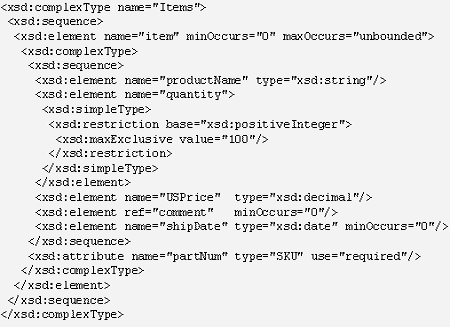


Элемент item представляет собой анонимный комплексный тип, состоящий из элементов productName, quantity, USPrice, comment, и shipDate, и имеющий атрибут partNum. Элемент quantity представляет собой анонимный простой тип, полученный из integer. При этом элемент типа quantity может принимать значение между 1 и 99.

**2.4 Определение анонимных типов**

При создании схем применяется два стиля. Схемы могут создаваться путем определения поименованных типов (например, PurchaseOrderType) с последующим объявлением элементов этого типа (например, purchaseOrder). При этом объявленные элементы ссылаются на поименованный тип с помощью конструкции type= . Этот стиль является достаточно простым, но может стать неуправляемым, особенно если Вы определяете много типов, на которые ссылаетесь только один раз, и которые содержат немного ограничений. В этих случаях, тип может быть более кратко определен как анонимный. Анонимный тип нет необходимости именовать и, следовательно, задавать на него ссылки. Определение типа Items в po.xsd содержит два объявления item и quantity, использующие анонимный тип. Наличие в схеме анонимного типа можно идентифицировать за счет отсутствия в объявлении элементов или атрибутов параметра type=, и присутствия непоименованного определения простого или комплексного типа.

Объявление двух анонимных типов:



Элемент item представляет собой анонимный комплексный тип, состоящий из элементов productName, quantity, USPrice, comment, и shipDate, и имеющий атрибут partNum. Элемент quantity представляет собой анонимный простой тип, полученный из integer. При этом элемент типа quantity может принимать значение между 1 и 99.

**2.5 Содержимое элемента**

Схема заказа имеет много примеров элементов, содержащих другие элементы (например, items); элементов, имеющих атрибуты и содержащих другие элементы (например, shipTo); и элементов, содержащих только значения простого типа (например, USPrice). Однако мы еще не рассматривали элементы, которые имеют атрибуты, но содержат значения простого типа; элементы, которые содержат другие элементы, смешанные с символьными выражениями; элементы, которые вообще не имеют никакого содержания. В этом разделе мы исследуем эти разновидности элементов.

**2.5.1 Комплексные типы из простых типов**

Вначале рассмотрим, как объявить элемент, который имеет атрибут и содержит значение простого типа. В документе такой элемент мог бы выглядеть как:

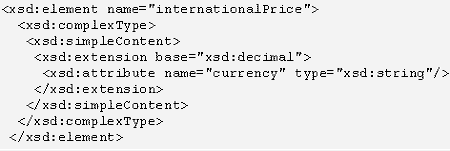
http://citforum.ru/internet/xml/scheme/15.gif

XML-схема заказа содержит объявление элемента USPrice, которое является отправной точкой:

http://citforum.ru/internet/xml/scheme/16.gif

Добавим к элементу атрибут. Поскольку, как мы прежде выяснили, простые типы не могут иметь атрибут, а decimal - простой тип, то чтобы добавить объявление атрибута, мы должны определить комплексный тип. Поскольку мы хотим, чтобы содержание элемента было простым числом типа decimal, то требуется ответить на вопрос: как задать определение комплексного типа, который основан на простом типе decimal? Ответ - мы должны получить новый комплексный тип из простого типа decimal.

Получение комплексного типа из простого типа:

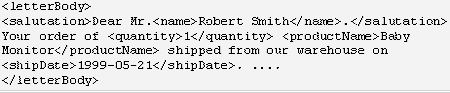


Для того чтобы начать описание нового анонимного типа, мы используем элемент complexType. Чтобы указать, что новый тип содержит только символьные данные и не содержит подэлементов, мы используем элемент simpleContent. Наконец, мы получаем новый тип, расширяя простой тип decimal. Расширение типа decimal заключается в добавлении (путем использования стандартного объявления) атрибута currency. Более детально образование типов будет рассмотрено в разделе 4. Пример использования в документе элемента internationalPrice, описанного в приведенном выше примере XML-схемы, располагается в начале данного пункта.

**2.5.2 Смешанное содержимое**

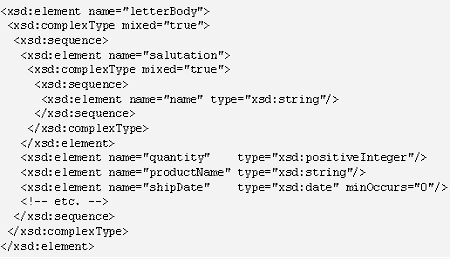
Схема заказа может быть охарактеризована как конструкция из элементов, содержащих подэлементы. При этом наиболее глубоко вложенные подэлементы содержат символьные данные. XML-схема также предусматривает конструкцию, где символьные данные могут появиться вместе с подэлементами. Это значит, что символьные данные могут располагаться в любых элементах.

Для иллюстрации рассмотрим следующий отрывок документа "письмо клиенту", который содержит теже элементы, что и заказ на покупку. Отрывок письма клиенту:



Обратите внимание на текст, появляющийся между элементами и их дочерними подэлементами. Текст появляется между элементами salutation, quantity, productName и shipDate, которые являются дочерними элементами letterBody. Ниже приводится отрывок схемы содержащий объявление letterBody.

Отрывок XML-схемы документа "письмо клиенту" c объявлением letterBody:



Элементы, появляющиеся в письме клиенту объявлены, и их типы определены, с помощью операторов element и complexType которые мы уже рассматривали. Чтобы разрешить символьным данным появиться между дочерними элементами letterBody, атрибут mixed в операторе определения типа равен true.

Обратите внимание, что смешанная модель, задаваемая XML-схемой, существенно отличается от смешанной модели принятой XML 1.0 (<http://www.w3.org/TR/2000/REC-xml-20001006>). В смешанной модели задаваемой XML-схемой порядок и число дочерних элементов, появляющихся в документе-образце должны согласовываться с порядком и номером дочерних элементов, указанных в XML-схеме. Напротив, в смешанной модели XML 1.0, порядок и число дочерних элементов, появляющихся в документе не ограничивается. В общем, применение XML-схемы, позволяет полностью контролировать применение смешанной модели содержимого элемента в отличие от смешанной модели XML 1.0, которая обеспечивает частичную проверку содержимого элемента.

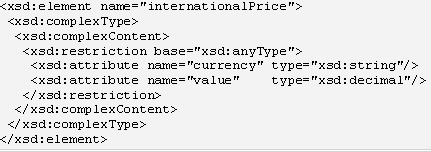
**2.5.3 Пустое содержимое**

Теперь предположим, что элемент internationalPrice будет задавать наименование валюты и цену как значения атрибутов, а не как значение атрибута и содержимого элемента. Например:

http://citforum.ru/internet/xml/scheme/20.gif

Такой элемент вообще не имеет никакого содержания. Чтобы определить тип, содержание которого пусто, мы по существу, определяем тип, который позволяет включать в его состав только подэлементы, но при этом не объявляем никаких элементов.

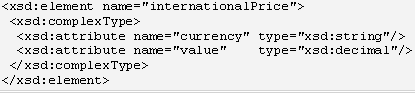
Пустой комплексный тип:



В этом примере, мы определяем анонимный тип с помощью оператора complexContent, то есть предполагается, что он будет содержать только элементы. Оператор complexContent говорит о том, что мы намереваемся ограничивать или расширять модель комплексного типа, а элемент restriction с параметром anyType объявляет два атрибута, но не задает никакого содержания элемента (более подробно ограничения рассматриваются в подразделе 4.4). В вышеприведенном примере показано как элемент internationalPrice, объявленный этим способом, может появиться в документе.

Предыдущий синтаксис для объявления пустого элемента является относительно подробным. Элемент internationalPrice можно объявить короче.

Упрощенное объявление пустого комплексного типа:



Этот компактный синтаксис работает потому, что комплексный тип, определенный без simpleContent или complexContent интерпретируется как упрощенное описание комплексного типа, который по умолчанию ограничивается параметром anyType.

**2.5.4 anyType**

anyType представляет абстракцию, называемую оur-typ (<http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/#Type_Derivation>), которая является базовым типом прародителем всех простых и комплексных типов. Тип anyType не ограничивает как-либо свое содержимое. Тип anyType используется подобно другим типам, например:

http://citforum.ru/internet/xml/scheme/23.gif

Содержание элемента, объявленного этим способом никак не ограничивается. Элемент может принимать значение 423.46, но это может быть и любая другая последовательность символов, или смесь символов и элементов. Фактически, anyType - это тип, задаваемый по умолчанию. Так что вышеуказанное объявление может выглядеть следующим образом:

http://citforum.ru/internet/xml/scheme/24.gif

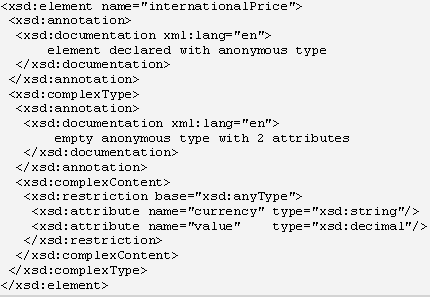
Элементы с произвольным содержанием могут применяться, например, в случае элементов содержащих литературный текст с символами национальной разметки. В этом случае заданное по умолчанию объявление или немного ограниченный вид этого объявления могут быть вполне подходящими. Тип text, описанный в подразделе 5.5 - пример такого типа.

**2.6 Аннотации**

Язык XML-схемы обеспечивает три элемента предназначенные для аннотации схемы. Содержимое этих элементов предназначено как для чтения человеком, так и для чтения приложением. В схеме заказа на покупку, мы размещаем описание схемы и информацию об авторском праве в элементе documentation, в котором рекомендуется размещать информацию для чтения документа человеком. Мы рекомендуем для указания языка комментариев использовать атрибут xml:lang со всеми элементами documentation. Вместо этого можно указать язык всей схемы, помещая атрибут xml:lang в элементе schema.

Элемент appInfo, который мы не применяли в схеме заказа закупки, может использоваться, чтобы предоставить информацию для инструментальных средств, таблиц стилей и других приложений. Интересный пример, использования appInfo приводится в схеме (http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/#schema), которая в документе "XML-схема. Часть 2: Типы данных" описывает простые типы. Например, внутри элемента appInfo представлена информация о том, какие фасеты могут быть применены к каждому из простых типов. Впоследствии эта информация была использована специальным приложением для автоматической генерации текста документа "XML-схема. Часть 2: Типы данных".

И documentation, и appInfo являются подэлементами annotation. Элемент annotation обычно размещают в начале большинства схем. В приведенном ниже примере элементы annotation располагаются вначале объявления элемента и вначале определения комплексного типа. Элементы annotation в объявлении элемента и определении типа:



Элемент annotation также может располагаться в начале других операторов языка XML-схемы, например, таких как schema, simpleType, и attribute.

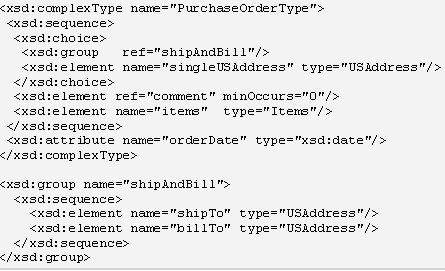
**2.7 Создание моделей содержимого**

Все определения комплексных типов в схеме заказа на закупку представляют собой последовательность объявлений элементов, которые должны появиться в документе-образце. Вхождение в документ каждого элемента, объявленного в так называемой модели содержимого данного типа, является необязательным, если атрибут minOccurs равен нулю или может быть ограничено в зависимости от значений атрибутов minOccurs и maxOccurs. XML-схема также может обеспечить ограничения вхождения группы элементов в данную модель содержимого. Эти ограничения отражают правила, применяемые в XML 1.0 и плюс некоторые дополнительные ограничения. Заметим, что ограничения не применимы к атрибутам.

XML-схема позволяет определить поименованную группу элементов, которые могут использоваться в моделях содержимого комплексного типа. Также может быть определена непоименованная группа элементов, которые вместе с элементами из поименованной группы будут появляться в документе в той же самой последовательности, в которой были объявлены. Вместе с тем, группы также могут быть спроектированы таким образом, что только один из элементов группы может появиться в документе-образце.

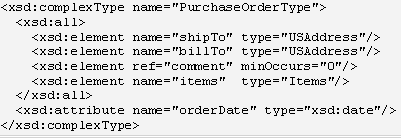
Для иллюстрации вышесказанного в определение PurchaseOrderType из схемы заказа на покупку введем две группы, так что заказ сможет содержать либо специальные элементы для указания адреса отправителя и продавца, либо адрес и отправителя и продавца будут задаваться одним и тем же элементом.

Группы выбора и последовательности:



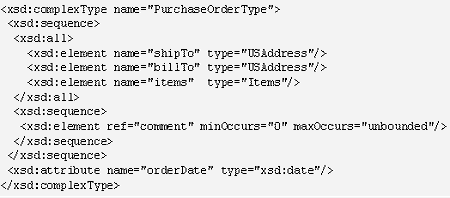
Элемент выбора в группе choice обеспечивает правило, по которому в документе-образце может появиться только один из его дочерних элементов. Элемент choice имеет двух потомков. Один из его потомков - элемент group, который ссылается на поименованную группу shipAndBill, и состоит из последовательности элементов shipTo, billTo. Второй потомок - singleUSAddress. Следовательно, в документе, элемент purchaseOrder должен содержать или элемент shipTo, за которым следует элемент billTo, или элемент singleUSAddress. За элементом группового выбора choice следуют объявления элементов comment и items. В свою очередь и элемент группового выбора и объявления элементов являются дочерними элементами групповой последовательности sequence. Применение двух последовательностей позволяет описать правило, по которому за адресом должны следовать комментарии, а за ними спецификация товаров.

Для ограничения появления элементов в группе существует еще одна возможность. Все элементы группы должны появиться один раз или не должны появиться ни разу, причем появляться они могут в произвольном порядке. Групповой элемент all ограничивает модель содержимого сверху. Кроме того, все дочерние элементы группы должны быть индивидуальными элементами (не группами), и все элементы должны появиться не более одного раза. То есть это соответствует значениям minOccurs = 0 и maxOccurs = 1. Например, чтобы позволить дочерним элементам purchaseOrder, появиться в любом порядке, переопределим PurchaseOrderType указанным ниже образом. Группа 'All':



В соответствии с этим определением элемент comment может появиться в любом месте purchaseOrder, причем как до, так и после элементов shipTo, billTo или Items. Но при этом он может появиться только однажды. Кроме того соглашения группы all не позволяют нам объявлять элементы вроде comment вне группы, что ограничивает возможность его использования для многократного появления. Язык XML-схемы предполагает, что группа all будет использоваться как единственный потомок в начале модели содержимого. Другими словами следующее объявление неверно.

Пример неверного использования группы 'All':

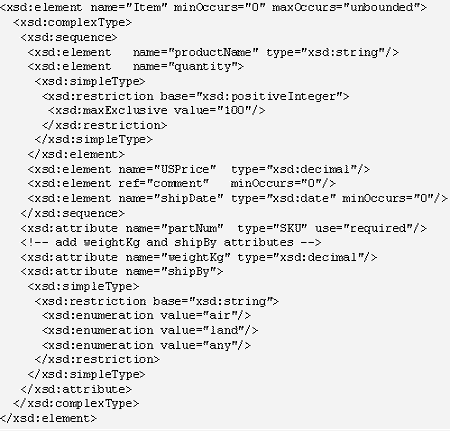


Поименованные и непоименованные группы, которые используются в моделях содержимого (group, choice, sequence, all) могут иметь атрибуты minOccurs и maxOccurs. Комбинируя и вкладывая различные группы, обеспечиваемые языком XML-схемы, и устанавливая значения minOccurs и maxOccurs, возможно представить любую модель содержимого, которая может быть выражена с помощью XML 1.0 DTD. Кроме того, группа all обеспечивает дополнительную выразительность языка XML-схемы по сравнению с XML 1.0 DTD.

**2.8 Группы атрибутов**

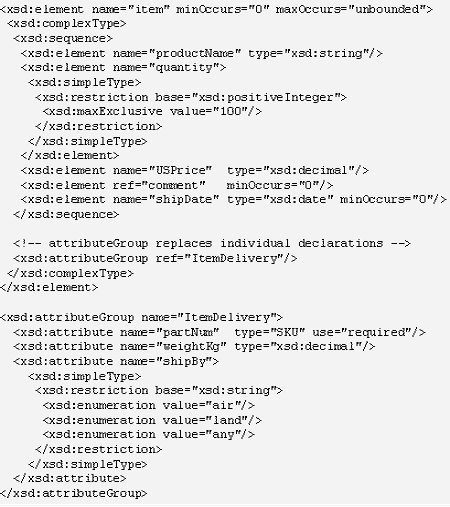
Предположим, что мы хотим обеспечить подробную информацию о каждом продукте в заказе на закупку. Например, вес каждого продукта и предпочтительный вариант отгрузки. Мы можем достигнуть этого, добавив к определению типа item (анонимному) объявления атрибутов weightKg и shipBy .

Включение атрибутов непосредственно в определение типа:



Вместо этого, мы можем создать поименованную группу атрибутов, содержащую все желательные атрибуты элемента item, и в объявлении item сделать ссылку на эту группу.

Добавление атрибутов с использованием группы атрибутов:



Применение группы атрибутов улучшает читаемость и облегчает модификацию схемы, так как группа атрибутов может быть определена и отредактирована в одном месте, а ссылки на нее могут использоваться в нескольких определениях и объявлениях. Эта характеристика групп атрибутов делает их похожими на сущность параметра в XML 1.0. Обратите внимание, что группа атрибутов может содержать другие группы атрибутов. Обратите также внимание, что и объявления атрибутов и ссылка на группу атрибутов, должны находиться в конце определений комплексных типов.

**2.9 Значения Nil**

Один из объектов в заказе на закупку, перечисленных в po.xml, Lawnmower, не имеет элемента shipDate. Автор схемы, возможно, предусмотрел такую возможность, чтобы указать еще не отгруженные изделия. Но вообще, отсутствие элемента не дает какой-либо определенной информации. Это может указывать на то, что информация отсутствует, или не соответствует действительности, или элемент может отсутствовать по другой причине. Иногда желательно представить не отгруженное изделие, неизвестную или неподходящую информацию явно с помощью элемента, а не отсутствующим элементом. Например, это может быть полезным при работе с пустыми значениями ("Null") реляционной базы данных. Для этих целей в языке XML-схемы имеется Nil-механизм. Этот механизм позволяет элементу появляться c или без нулевого значения.

Для индикации возможности пустого значения элемента Nil-механизм XML-схемы использует специальный признак. Другими словами, возможное пустое значение элемента обозначается не с помощью какого-либо специального Nil-значения содержимого, а с помощью специального атрибута, индицирующего возможность пустого значение элемента. Для иллюстрации вышесказанного, объявление элемента shipDate изменим так, чтобы можно было идентифицировать его пустое значение:

http://citforum.ru/internet/xml/scheme/31.gif

Для того чтобы явно указать в документе, что shipDate имеет пустое значение, устанавливаем атрибут nil равным true. Для идентификации того, что атрибут xsi:nil принадлежит языку XML-схемы, а не схеме данного типа документа в имени атрибута используем именное пространство языка XML-схем:

http://citforum.ru/internet/xml/scheme/32.gif

Атрибут nil определен в именном пространстве языка XML-схемы, http://www.w3.org/2001/XMLSchema-instance, и поэтому в документе-образце используется с префиксом (таким как xsi:), связанным с этим именным пространством. Как и xsd:, префикс xsi: используется в соответствии со стандартным соглашением. Заметим, что nil-механизм применим только к значениям элементов, а не к значениям атрибутов. Элемент с xsi:nil="true" не может иметь никакого содержания, но может иметь атрибуты.

**3 Дополнительные понятия I: именные пространства, схемы & квалификация**

Схема может рассматриваться как коллекция (словарь) определений типов и объявлений элементов, имена которых принадлежат определенному именному пространству, называемому целевым именным пространством. Целевые именные пространства дают возможность различать определения и объявления, принадлежащие различным словарям. Например, целевое именное пространство позволяет отличить элемент объявленный в словаре языка XML-схемы от элемента, объявленного в гипотетическом словаре языка описания химических элементов. Первый элемент принадлежит целевому именному пространству <http://www.w3.org/2001/XMLSchema>, а последний - часть некого другого целевого именного пространства.

Когда мы хотим убедиться, что документ соответствует одной или более схемам (с помощью процесса, называемого верификацией схемы), то мы должны идентифицировать объявления элементов и атрибутов, а также определения типов в схемах которые будут использоваться для проверки соответствующих элементов и атрибутов в документе. Целевое именное пространство играет важную роль в процессе идентификации. Роль целевого именного пространства будет рассмотрена в следующем разделе.

Автору схемы предоставляется ряд опций, которые влияют на то, как будут отождествляться элементы и атрибуты, представленные в документе. Автор может решить, действительно ли появление локально объявленных элементов и атрибутов в документе должно быть квалифицировано явным или неявным (заданным по умолчанию) префиксом именного пространства. Выбор автора схемы относительно квалифицирования локальных элементов и атрибутов может зависеть от множества факторов касающихся структуры схемы и документов. Некоторые из этих факторов мы рассмотрим в следующих разделах.

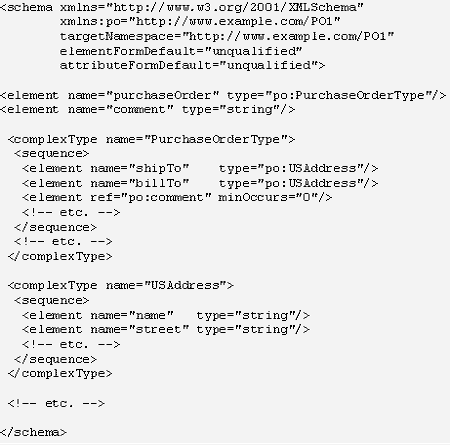
**3.1 Целевые именные пространства и не квалифицируемые локальные объекты**

В новом варианте схемы заказа на закупку (po1.xsd) мы явно объявляем целевое именное пространство, и определяем, что локально определенные элементы и локально определенные атрибуты должны быть не квалифицированы. Целевое именное пространство в po1.xsd задается значением атрибута targetNamespace и равно "http://www.example.com/PO1".

Квалификация локальных элементов и атрибутов может быть глобально определена парой атрибутов, elementFormDefault и attributeFormDefault, элемента schema, или может быть определена отдельно для каждого локального объявления с помощью атрибута form. Значение каждого из этих атрибутов может быть установлено в unqualified или в qualified, что указывает на признак квалифицируемости локально объявленных элементов и атрибутов.

В po1.xsd глобально определен параметр квалифицированности элементов и атрибутов путем присвоения атрибутам elementFormDefault и attributeFormDefault значения равного unqualified. Строго говоря, эти атрибуты с таким значением можно не задавать, потому что это значение является для обоих атрибутов значением по умолчанию. Мы задаем их здесь в явном виде для того, чтобы показать разницу между этим вариантом и другими вариантами, которые мы опишем позже.

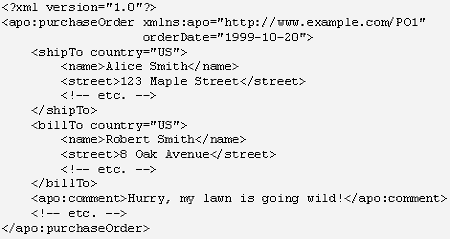
Схема заказа на закупку с целевым именным пространством, po1.xsd:



Чтобы увидеть, как пополняется целевое именное пространство приведенной выше схемы, рассмотрим каждое из объявлений элементов и определений типов. Начнем с конца схемы. Здесь мы определяем тип с названием USAddress, который состоит из элементов name, street, и т.д. В результате определения, тип USAddress включается в состав именного пространства схемы. Затем мы определяем тип с названием PurchaseOrderType, который состоит из элементов shipTo, billTo, comment, и т.д. PurchaseOrderType также включается в состав именного пространства схемы. Обратите внимание, что ссылки на тип в объявлениях трех элементов имеют префикс: po:USAddress, po:USAddress и po:comment, и этот префикс связан с именным пространством http://www.example.com/PO1. Это то же самое именное пространство, что и целевое именное пространство схемы. Поэтому обработчик этой схемы будет знать, что определение типа USAddress и объявление элемента comment находятся в пределах данной схемы. Вы также можете сослаться на определение типов находящееся в другой схеме с другим целевым именным пространством. Следовательно, допускается повторное переопределение типов и переобъявление элементов от схемы к схеме.

В начале схемы po1.xsd мы объявляем элементы purchaseOrder и comment. Они включаются в целевое именное пространство схемы. Тип элемента purchaseOrder имеет префикс также как и тип элемента USAddress. Напротив, префикс типа у элемента comment имеющего тип string не установлен.

Ниже приводится пример того, как целевое именное пространство схемы затрагивает соответствующий документ. Заказ на закупку с неквалифицированными локальными объектами, po1.xml:



В документе объявлено одно именное пространство http://www.example.com/PO1, которое связано с префиксом apo:. Этот префикс используется для того, чтобы квалифицировать два элемента в документе: purchaseOrder и comment. Именное пространство документа то же, что и целевое именное пространство схемы в файле po1.xsd. Поэтому при анализе документа обработчик будет знать, в какой схеме искать объявления purchaseOrder и comment. Целевым именное пространство названо в том смысле, что оно является именным пространством, в котором находятся элементы purchaseOrder и comment. Поэтому целевые именные пространства схемы управляют верификацией соответствующих именных пространств документа.

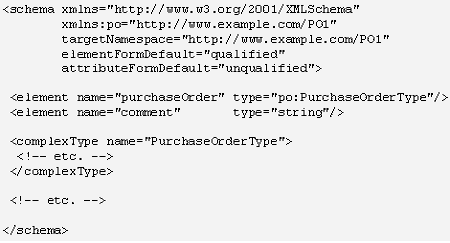
Префикс apo: применен к глобальным элементам purchaseOrder и comment. Кроме того, elementFormDefault и attributeFormDefault требуют, чтобы префикс не применялся ни к одному из локальных элементов вроде shipTo, billTo, name и street, и ни к одному из атрибутов (которые все объявлены локально). Элементы purchaseOrder и comment являются глобальными, так как объявлены в контексте схемы в целом, а не в пределах контекста какого-либо типа. Например, объявление элемента purchaseOrder в po1.xsd является дочерним по отношению к элементу schema, тогда как объявление shipTo является дочерним по отношению к элементу complexType, который определяет тип PurchaseOrderType.

Если локальные элементы и атрибуты не квалифицированы, то автору для создания корректных документов могут в той или иной степени потребоваться знания о подробностях схем этих документов. Ситуация несколько проясняется если автор знает что только корневой элемент (типа purchaseOrder) является глобальным. Тогда квалифицировать нужно только корневой элемент. Если автор знает, что все элементы объявлены глобально, то все элементы в документе могут иметь префикс, который возможно связан с именным пространством, объявленным по умолчанию (мы исследуем этот подход в подразделе 3.3). С другой стороны, если нет какого-либо однородного правила для глобальных и локальных объявлений, то автор документов будет нуждаться в детальном знании схемы для правильного применения префиксов к глобальным элементам и атрибутам.

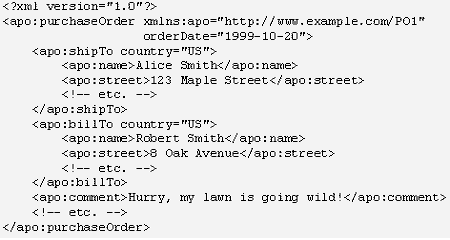
**3.2 Квалифицируемые локальные объекты**

Квалификация элементов и квалификация атрибутов может быть произведена независимо. Начнем с рассмотрения квалифицирования локальных элементов. Для указания того, что все локально объявленные элементы в схеме должны быть квалифицированы, мы устанавливаем значение elementFormDefault равным qualified.

Модификация po1.xsd с квалифицируемыми локальными объектами:

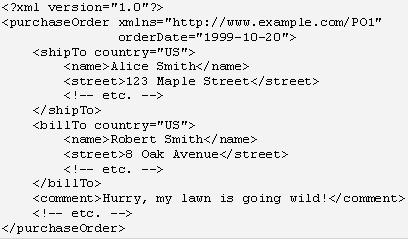


Ниже приводится приспособленный к измененной схеме документ, в котором все элементы квалифицированы явно. Заказ на закупку с явно квалифицированными локальными элементами:



Попробуем заменить явную квалификацию каждого элемента неявной. Неявная квалификация обеспечивается именным пространством, назначенным по умолчанию (пример в po2.xml).

Заказ на покупку с квалификаций локальных объектов по умолчанию. po2.xml:



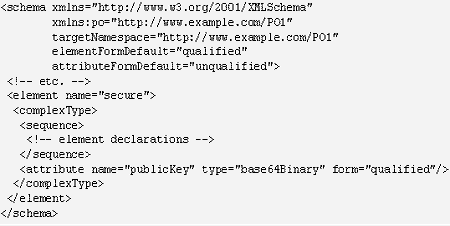
В po2.xml все элементы в документе принадлежат тому же именному пространству, которое объявлено по умолчанию с помощью атрибута xmlns. Следовательно, явное добавление префикса к элементам становится ненужным.

Другой пример использования квалифицированных элементов - схемы в разделе 5. Все указанные схемы требуют квалифицированных элементов.

Квалификация атрибутов очень похожа на квалификацию элементов. Атрибуты, которые должны быть квалифицированы (либо потому что объявлены глобально, либо потому что признак attributeFormDefault, установлен в qualified), в документах появляются с префиксом. Один из примеров квалифицированного атрибута - атрибут xsi:nil, который был рассмотрен в подразделе 2.9. Фактически, атрибуты, которые должны быть квалифицированы, должны иметь явно заданные префиксы. Это связано с тем, что спецификация XML-Namespaces (<http://www.w3.org/TR/1999/REC-xml-names-19990114/>) не обеспечивает механизм задания именного пространства для атрибутов по умолчанию. Атрибуты, которые не обязаны быть квалифицированны, появляются в документах без префиксов, что является типичным случаем.

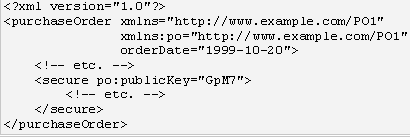
Механизм квалификации, который мы до сих пор описывали, управлял всеми локальными объявлениями элементов и атрибутов в пределах конкретного целевого именного пространства. С помощью атрибута form, возможно управлять квалификацией отдельного объявления. Например, если требуется, чтобы локально объявленный атрибут publicKey в документах был квалифицирован, мы объявляем его нижеследующим способом.

Квалификация одного атрибута:



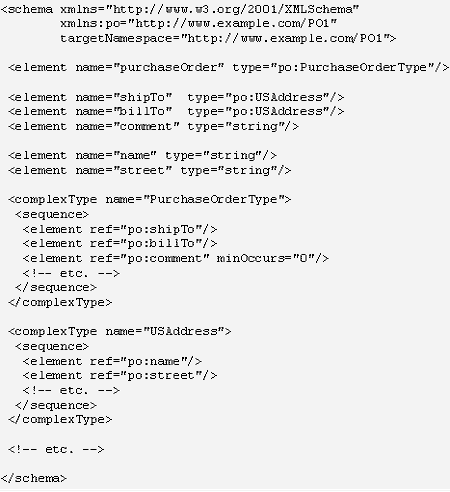
Заметьте, что для publicKey, значение атрибута form отменяет значение атрибута attributeFormDefault. Таким же образом атрибут form может быть применен к объявлению элемента. Ниже приводится документ, который соответствует приведенной схеме.

Документ с квалифицированным атрибутом:



**3.3 Глобальные объявления в сравнении с локальными объявлениями**

Другой стиль применяется тогда, когда наименования всех элементов в пределах именного пространства уникальны. При этом автор создает схемы, в которых все элементы являются глобальными. Это напоминает использование оператора < !ELEMENT> в DTD. В нижеприведенном примере, мы изменили оригинал po1.xsd так, что все элементы объявлены глобально. Обратите внимание, что в этом примере мы опустили атрибуты elementFormDefault и attributeFormDefault. Это сделано для того, чтобы подчеркнуть, что когда объявлены только глобальные элементы и атрибуты, значения этих атрибутов являются несущественными. Модифицированная версия po1.xsd, использующая только глобальные объявления элементов:



С помощью этого "глобального" варианта po1.xsd можно проверить правильность ранее рассмотренного документа po2.xml, который также соответствует "квалифицированному" варианту схемы po1.xsd. Другими словами, оба варианта схемы могут проверить правильность одного и того же документа, чье именное пространство задано по умолчанию. Таким образом, с одной стороны оба варианта схемы подобны, хотя с другой стороны различны. Когда все элементы объявлены глобально, преимущества локального объявления теряются. Например, Вы можете объявить только один глобальный элемент с именем "title". В противоположность к глобальному объявлению, Вы можете объявить несколько локальных элементов с одинаковым именем. Например, можно объявить локальный элемент с именем "title", который будет подэлементом элемента "book", и иметь строковый тип. В пределах той же самой схемы (целевого именного пространства) Вы можете объявить второй элемент с именем "title", который является перечислением значений "Mr Mrs Ms".

**3.4 Необъявленные целевые именные пространства**

В разделе 2 мы объяснили основы языка XML-схем, используя схему, в которой не объявлялось целевое именное пространство, и документ, в котором не объявлялось именное пространство. Возникает вопрос: каково целевое именное пространство в этих примерах, и как на него ссылаются?

В схеме заказа на закупку po.xsd мы не объявляли ни ее целевое именное пространство, ни префикса (наподобие po:) связанного целевым именным пространством схемы в котором содержатся определения типов и объявления элементов и атрибутов. Вследствие того, что в схеме отсутствует объявление целевого именного пространства, на определения и объявления находящиеся внутри схемы, типа USAddress и purchaseOrder, ссылаются без квалификационного префикса. Другими словами в схеме нет ни явного префикса, используемого в ссылках ни определения именного пространства используемого в ссылках по умолчанию. Пример - элемент purchaseOrder объявленный с использованием ссылки на тип PurchaseOrderType. Напротив, все элементы и типы, принадлежащие словарю языка XML-схемы, используемые в po.xsd явно квалифицированы префиксом xsd:, который связан с именным пространством языка XML-схемы.

В случаях, когда схема разработана без целевого именного пространства, строго рекомендуется, чтобы все элементы, принадлежащие словарю языка XML-схемы, были явно квалифицированы префиксом xsd:, который связан с именным пространством языка XML-схемы (как в po.xsd). Такая рекомендация объясняется тем, что если элементы и типы языка XML-схем связываются с именным пространством языка XML-схем по умолчанию, то есть без префиксов, то ссылки на типы определенные в языке XML-схемы будет невозможно отличить от ссылок на типы определенные пользователем.

Объявления элементов в схеме без целевого именного пространства используется для верификации неквалифицированных элементов в документе. То есть они определяют правильность элементов, для которых не заданы никакие префиксы именного пространства: ни явные, ни заданные по умолчанию (xmlns:).

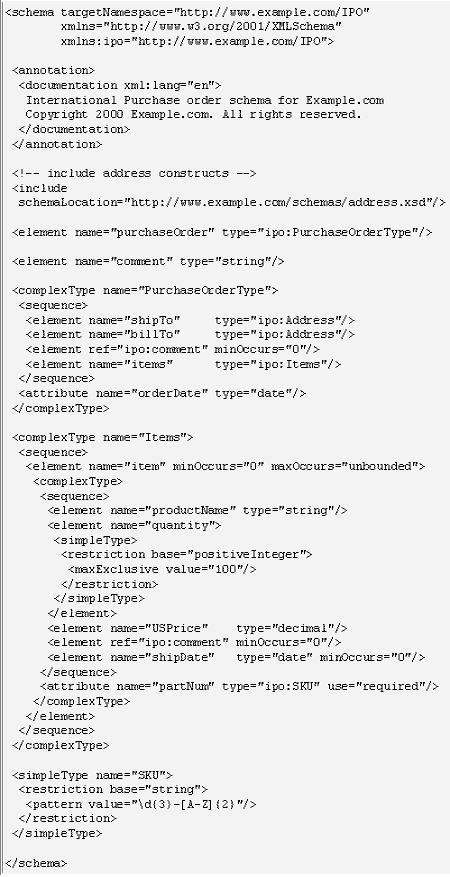
**4 . Дополнительные понятия II: международный заказ на покупку**

Схема заказа на покупку, описанная в разделе 2 размещалась в одном файле, и большинство конструкций схемы, типа объявлений элементов и определений типов, были созданы вновь. В действительности, у авторов схем возникает потребность составлять схемы из конструкций, расположенных в нескольких файлах, и создавать новые типы, основанные на существующих типах. В этом разделе, мы исследуем механизмы, которые допускают такие композиции схем и методы создания типов.

**4.1 Схема в нескольких файлах**

Поскольку схемы становятся большими, то в целях упрощения сопровождения, управления доступом, и читабельности, желательно делить их содержимое на части. По этим причинам, мы взяли конструкции связанные с определением адресов из схемы po.xsd, и поместили их в новый файл с именем address.xsd. Измененную схему заказа сохраним в файле с именем ipo.xsd.

Международная схема заказа на покупку, ipo.xsd:



Ниже приводится файл с конструкциями описания адресов.

Адреса для схемы международного заказа на покупку, address.xsd:

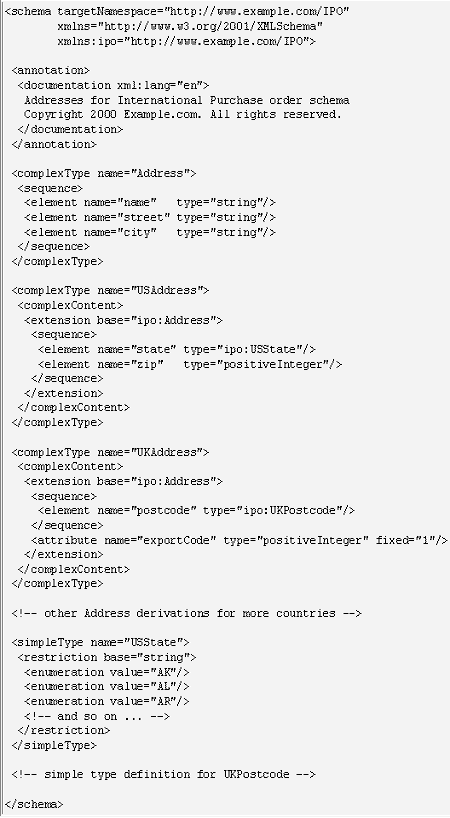


Схема заказа на покупку и адресные конструкции теперь содержатся в двух различных файлах: ipo.xsd и address.xsd. Чтобы включать описание адресов в схему международного заказа на покупку, другими словами, чтобы включить их в именное пространство схемы международного заказа на покупку, в схеме ipo.xsd используется элемент include:

< include schemaLocation="http://www.example.com/schemas/address.xsd"/>

Элемент include добавляет определения и объявления, содержащиеся в address.xsd, и делать их доступными как часть целевого именного пространства схемы международного заказа на покупку. Использование include имеет одно важное ограничение: целевое именное пространство включенных элементов должно быть то же самое что и целевое именное пространство схемы, в которую производится включение. В приведенном выше примере это http://www.example.com/IPO. Включение определений и объявлений с использованием механизма include добавляет элементы к существующему целевому именному пространству. В подразделе 4.5, мы опишем механизм, который дает возможность переопределить некоторые включенные элементы после того, когда они добавлены.

В нашем примере, мы показали только одну включающую схему и одну включенную подсхему. Практически возможно включить больше чем одну подсхему, используя несколько элементов include, при этом включаемые подсхемы могут включать другие подсхемы. Однако, вложение подсхем этим способом допустимо, только если все включенные части схемы объявлены с одним и тем же целевым именным пространством.

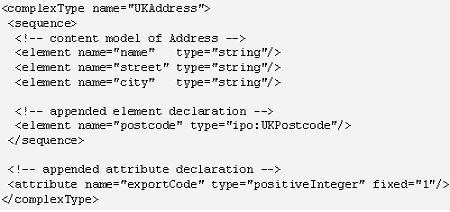
Документы, которые соответствуют схеме сформированной из нескольких частей (подсхем), нуждаются в ссылке только на "самую верхнюю" схему и общее для всех подсхем именное пространство. Ответственность за сборку схемы из подсхем ложится на программу обработки схемы. В нашем примере, документ ipo.xml (см. подраздел 4.3) ссылается только на одно общее целевое именное пространство, <http://www.example.com/IPO>, и (косвенно) на один файл со схемой, http://www.example.com/schemas/ipo.xsd. Подключение файла address.xsd обеспечивает программа обработки схемы. В подразделе 5.4 мы опишем, как схемы могут использоваться для проверки правильности содержимого документа, зависящего от нескольких именных пространств.

**4.2 Получение производных типов расширением**

Создание адресных конструкций начинаем с определения комплексного типа Address (см. address.xsd). Тип Address содержит основные элементы адреса: имя, улицу и город (такое определение адреса используется не во всех странах, но вполне подойдет для нашего примера). От этой отправной точки мы получаем два новых комплексных типа, которые содержат все элементы первоначального типа плюс дополнительные элементы, необходимые для указания адресов принятых в США и Великобритании. Методика, которую мы здесь используем, чтобы получить новые (комплексные) адресные типы, расширяя существующий тип - та же самая методика, которую мы использовали в пункте 2.5.1. Отличие заключается в том, что наш исходный тип, это комплексный тип, тогда как исходным типом в предыдущем разделе был простой тип.

С помощью элемента complexType определяем два новых комплексных типа, USAddress и UKAddress. Используя элемент complexContent, мы указываем, что модели содержимого новых типов комплексны, то есть содержат элементы. Кроме того, с помощью атрибута base элемента extension указываем, что новый тип создается путем расширения базового типа Address. Когда комплексный тип получен расширением, его модель содержимого включает модель содержимого исходного типа плюс модель содержимого заданная в описании создаваемого типа. Кроме того, обе модели содержимого оформлены как две дочерних записи групповой последовательности. В случае UKAddress, модель содержимого UKAddress состоит из модели содержимого Address плюс объявления элемента postcode и атрибута exportCode. Такое объявление аналогично объявлению типа UKAddress с нуля, пример которого приводится ниже.

Пример альтернативного объявления UKAddress:

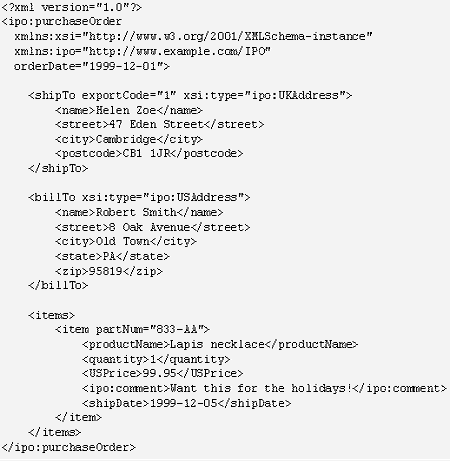


**4.3 Использование производных типов в документах**

В сценарии нашего примера заказы на покупку генерируются в ответ на запросы клиентов, которые могут находиться в различных странах. Значит, реквизиты отгрузки товаров и счетов могут зависеть от национальных особенностей. Ниже приводится пример международного заказа на покупку, ipo.xml, в котором товар отгружается в Великобританию, а счет отсылается в США. Очевидно, что нецелесообразно определять в схеме все возможные комбинации международных адресов, целесообразнее определить новые типы адресов, порождая их из типа Address.

В XML-схеме определены billTo и shipTo как элементы типа Address (см. ipo.xsd) которые использованы в документе для указания международного адреса в том месте, где должны находится элементы типа Address. Другими словами документ, содержащий данные типа UKAddress будет синтаксически правильным, если это содержимое находится в документе на том месте, где ожидается элемент типа Address (подразумевается, что содержимое UKAddress синтаксически правильно). Для того чтобы эта функция XML-схемы работала, и для указания того, какой именно производный тип применяется, производный тип должен быть идентифицирован в документе-образце. Тип идентифицируется с помощью атрибута xsi:type, который принадлежит именному пространству языка XML-схемы. В примере ipo.xml использование производных типов UKAddress и USAddress идентифицируется с помощью значения заданного атрибутом xsi:type.

Международный заказ на покупку, ipo.xml:



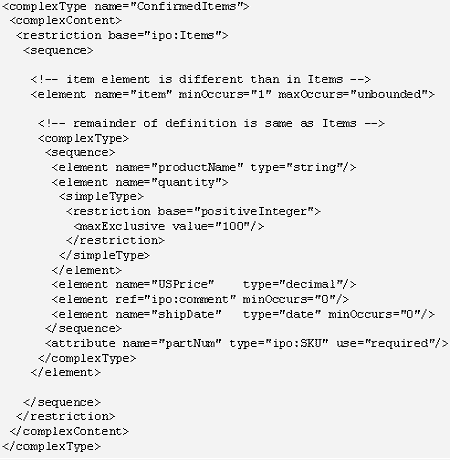
Как управлять использованием производных типов описывается в подразделе 4.8.

**4.4 Производные сложные типы, полученные путем ограничений**

Существует два метода создания новых комплексных типов. Первый метод основан на расширении модели содержимого базового типа, а второй - на ограничении модели содержимого базового типа. Ограничение сложных типов концептуально аналогично ограничению простых типов, за исключением того, что ограничение сложных типов затрагивает объявления типа, а не диапазон допустимых значений как в случае простого типа. Сложный тип, полученный ограничением очень похож на свой базовый тип, за исключением того, что его объявления более ограничены, чем соответствующие объявления в базовом типе. Фактически, значения, представленные новым типом являются подмножеством значений, представленных в базовом типе (как это имеет место с ограничением простых типов). Другими словами, приложение, подготовленное к работе со значениям базового типа будет готово к работе со значениями ограниченного типа.

Например, предположим, что мы хотим модифицировать определение списка продукции в международном заказе на покупку так, чтобы он содержал не менее одного товара в заказе. Схема, приведенная в ipo.xsd, позволяет элементу items появляться без каких-либо дочерних элементов. Чтобы создавать новый тип ConfirmedItems, мы обычным путем определяем новый тип и указываем, что он получен ограничением базового типа Items. При этом задаем новое (более ограниченное) значение минимального числа элементов item. Обратите внимание, что типы, полученные ограничением должны повторить все компоненты определения базового типа, которые должны быть включены в производный тип.

Тип ConfirmedItems полученный ограничением типа Items:



В соответствии с приведенными выше изменениями, требуется, чтобы список состоял не менее чем из одного дочернего элемента вместо того, чтобы как раньше разрешать нуль или больше дочерних элементов. Это сужает диапазон допустимого числа дочерних элементов от минимума равного 0 к минимуму равному 1. Заметим, что все элементы типа ConfirmedItems также будут допустимы как элементы типа Item.

Для иллюстрации ограничений в таблице 3 приводится несколько примеров того, как в пределах определений типов могут быть ограничены объявления элементов и атрибутов.

Таблица 3 Примеры ограничений

|  |  |  |
| --- | --- | --- |
| **Базовые значения** | **Ограничения** | **Примечания** |
|  | default="1" | Установка значения по умолчанию |
|  | fixed="100" | Установка фиксированного значения |
|  | type="string" | Спецификация типа |
| (minOccurs, maxOccurs) | (minOccurs, maxOccurs) |  |
| (0, 1) | (0, 0) | Исключение необязательного компонента; этого также можно достигнуть, опуская объявление компонента в определении производного типа |
| (0, unbounded) | (0, 0) (0, 37) |  |
| (1, 9) | (1, 8) (2, 9) (4, 7) (3, 3) |  |
| (1, unbounded) | (1, 12) (3, unbounded) (6, 6) |  |
| (1, 1) | - | Ограничение с помощью minOccurs или maxOccurs невозможно |

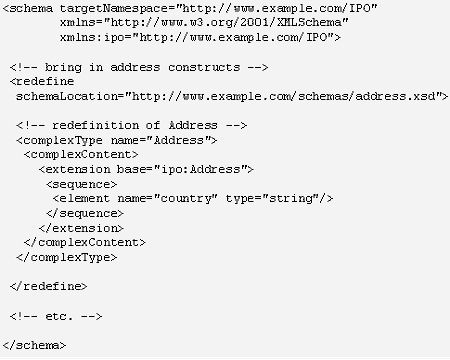
**4.5 Переопределение типов и групп**

В подразделе 4.1 мы описали, как включить определения и объявления, полученные из внешних подсхем, имеющих одинаковое целевое именное пространство. Механизм include дает возможность использовать созданные вовне компоненты схемы "как есть", без какой либо модификации.

Мы описали, как получить новые типы расширением и ограничением. Здесь мы рассмотрим механизм переопределения (redefine), который дает возможность переопределить простые и комплексные типы, группы, и группы атрибутов, полученные из внешних файлов схемы. Подобно механизму включения, механизм переопределения, требует, чтобы внешние компоненты находились в том же самом целевом именном пространстве, что и переопределяемая схема, хотя внешние компоненты схем, которые не имеют никакого именного пространства, также могут быть переопределены. В последнем случае, переопределенные компоненты становятся частью адресного пространства переопределяемой схемы.

Чтобы проиллюстрировать механизм переопределения, мы используем его вместо механизма включения в схеме международного заказа на покупку, ipo.xsd. Используем механизм переопределения для изменения комплексного типа Address, находящегося в address.xsd.

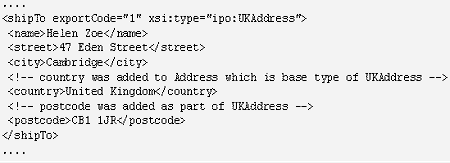
Использование переопределения в международном заказе на покупку:



Элемент redefine действует подобно элементу include, поскольку включает все объявления и определения из файла address.xsd. Определение комплексного типа Address использует знакомый синтаксис расширения для того, чтобы добавить элемент country к определению Address. Обратите внимание, что исходный тип - также Address.

Теперь, когда тип Address переопределен, расширение применяется ко всем элементам схемы, которые используют Address. Например, address.xsd содержит определения типов международных адресов, которые образованы из Address. Эти образования отражают переопределенный тип Address, как показано в приведенном ниже фрагменте.

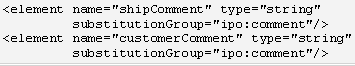
Фрагмент ipo.xml использующий переопределенный тип Address:



Приведенный пример был тщательно продуман на предмет того, чтобы переопределенный тип Address не конфликтовал с производными типами, полученными из первоначального определения типа Address. Но заметим, что такой конфликт создать очень просто. Например, если тип международных адресов является расширением типа Address, полученным добавлением элемента country, то переопределение типа Address будет прибавлять к модели содержимого Address элемент с тем же самым именем. Наличие двух элементов с одинаковыми именами (в одном и том же целевом именном пространстве) но различными моделями содержимого - недопустимо. Поэтому попытка переопределить Address вызвала бы ошибку. Вообще, переопределение не защищает от таких ошибок, поэтому этот механизм должен использоваться осторожно.

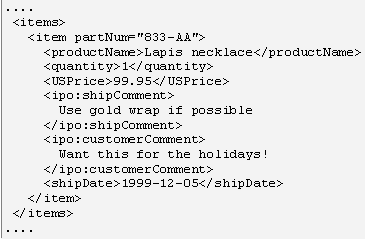
**4.6 Группы замены**

XML-схема обеспечивает механизм, называемый группами замены, который позволяет заменять одни элементы другими. Более точно, элементы могут быть объединены в специальную группу элементов, которые, как говорят, являются замещающими для элемента, называемого головным. Обратите внимание, что головной элемент должен быть объявлен как глобальный. Для иллюстрации, мы объявим два элемента customerComment и shipComment, и объединим их в группу замены, главный элемент которой - comment. В результате customerComment и shipComment можно использоваться везде, где мы можем использовать comment. Элементы в группе замены должны иметь тот же самый тип, что и головной элемент, или они могут иметь тип, который был порожден из типа головного элемента. Ниже приводится пример, в котором два новых элемента заменяют элемент comment. Объявление заменяющих элементов для элемента comment:



Когда эти объявления будут добавлены к схеме международного заказа на покупку, тогда в документе элемент comment можно заменить элементами shipComment и customerComment.

Отрывок ipo.xml с замещающими элементами:



Обратите внимание, что когда документ содержит элементы замены, типы которых получены из их главных элементов, то нет необходимости идентифицировать производные типы с помощью конструкции xsi:type, которую мы описали в подразделе 4.3. Существование группы замены не требует обязательного использования какого-либо из элементов этой группы, и при этом не препятствует использованию главного элемента. Это просто позволяет элементам быть взаимозаменяемыми.

**4.7 Абстрактные элементы и типы**

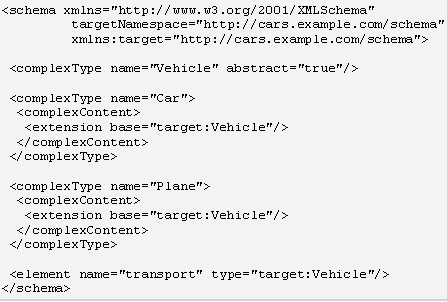
Язык XML-схемы обеспечивает возможность принудительной замены специфических элементов или типов. Если элемент или тип объявлены "абстрактными", то они не могут использоваться в документе. Когда абстрактным объявляется элемент, то в документе должны использоваться элементы, входящие в его группу замены. Если элемент соответствующего типа объявлен абстрактным, то все экземпляры этого элемента должны использовать xsi:type для указания производного типа, который не является абстрактным.

В примере группы замены, описанном в подразделе 4.6, запретим использование элемента comment так, чтобы в документах можно было использовать только элементы customerComment и shipComment. Для этого объявим элемент comment абстрактным, и изменим его первоначальное объявление в схеме международного заказа на покупку, ipo.xsd, следующим образом:

< element name="comment" type="string" abstract="true"/>

С элементом comment объявленным абстрактным, документы международных заказов на покупку будут синтаксически правильными, если они содержат комментарии в элементах customerComment и shipComment.

Объявление абстрактного элемента требует использования группы замены. Объявление абстрактного типа требует использования в документе производного (идентифицированного атрибутом xsi:type) от него типа. Рассмотрим приведенный ниже пример. Схема описания транспортных средств:



Элемент transport не абстрактен, поэтому может появиться в документах. Однако его определение типа абстрактно, и поэтому элемент transport не может появляться в документе без атрибута xsi:type, который ссылается на производный тип. Это означает, что следующий фрагмент не соответствует схеме (так как элемент transport имеет абстрактный тип):

< transport xmlns="http://cars.example.com/schema"/>

Однако следующий фрагмент синтаксически правилен, поскольку элемент transport имеет не абстрактный тип Car, замещающий абстрактный тип Vehicle.

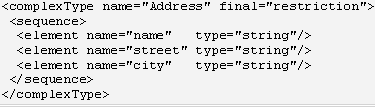
http://citforum.ru/internet/xml/scheme/51.gif

**4.8 Управление созданием и использованием производных типов**

До этого мы порождали новые типы и использовали их в документах, без каких либо ограничений. В действительности авторы схем могут захотеть управлять порождением новых типов и использованием их в документах.

Язык XML-схемы обеспечивает несколько механизмов, которые управляют образованием типов. Один из этих механизмов позволяет автору схемы ограничить получение новых типов из данного комплексного типа: (a) - новые типы не могут быть получены с использованием ограничений, (b) - новые типы не могут быть получены с использованием расширений, (c) - новые типы не могут быть получены вообще. Чтобы проиллюстрировать сказанное, предположим, что мы хотим предотвратить любое образование новых типов из типа Address методом ограничения. Будем считать, что мы намереваемся использовать этот тип только как базовый для расширенных типов типа USAddress и UKAddress. Чтобы предотвратить любые такие образования немного изменим первоначальное определение Address так, как показано ниже.

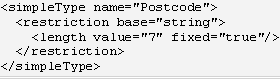
Предотвращение образования типов методом ограничения типа Address:



Значение restriction атрибута final предотвращает образования новых типов методом ограничения. Значение #all предотвращает образование новых типов вообще. Значение extension предотвращает образования новых типов методом расширения. Кроме того, у элемента schema существует необязательный атрибут finalDefault, который может принимать одно из значений указанных для атрибута final. Появление атрибута finalDefault эквивалентно определению атрибута final для каждого определения типа и объявления элемента в схеме.

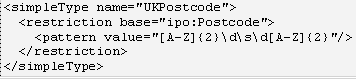
Другой механизм контроля над образованием типов, заключается в указании того, какие фасеты могут быть применены при образовании нового простого типа. При определении простого типа к любому из его фасетов может быть применен атрибут fixed. Этот атрибут предотвращает порождение нового типа путем изменения значения фиксированного фасета. Например, определим простой тип Postcode.

Защита изменений фасетов для простого типа:



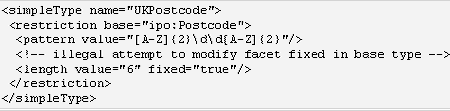
Как только тип Postcode определен, мы можем получить из него новый тип почтового кода. Новый тип получим путем применения фасета, неограниченного в базовом типе Postcode (см., пример ниже).

Корректное образование нового типа из Postcode:



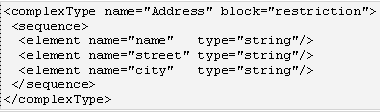
Однако мы не можем получить новый почтовый код, в котором мы повторно переопределяем какой-либо фасет, зафиксированный в базовом определении.

Некорректное образование нового типа из Postcode:



В дополнение к механизмам, которые управляют порождением новых типов, язык XML-схемы обеспечивают механизм, который управляет тем, какие образования и группы замены могут использоваться в документах. В подразделе 4.3, мы описали, как производные типы USAddress и UKAddress, могли использоваться в документах через элементы shipTo и billTo. Поскольку эти производные типы получены из типа Address, то они могут заменить модель содержимого, обеспеченную типом Address. Однако замену производными типами можно контролировать с помощью атрибута block, задаваемого в определении базового типа. Например, если мы хотим блокировать использование любого, образованного методом ограничения типа, вместо Address (возможно, поэтому мы определили Address с final="restriction"), то мы можем изменить первоначальное определение Address указанным ниже образом.

Защита от использования в документах типа Address, полученного методом ограничения:



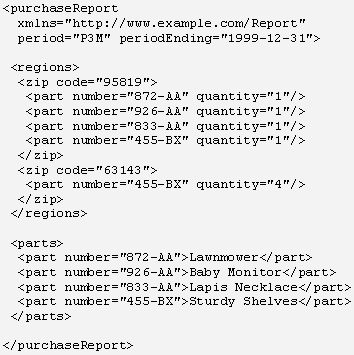
Атрибут block="restriction" препятствует замене в документах типа Address всеми порожденным из него, в соответствии с методом ограничения, типами. Однако это не препятствовало бы UKAddress и USAddress заменять Address, поскольку они были получены методом расширения. Предотвращение замены модели содержимого базового типа порожденными типами вообще, или типами, образованными методом расширения, обозначено значениями #all и extension соответственно. Как и в случае с final, у элемента schema существует дополнительный атрибут blockDefault, значение которого может быть одним из значений, которые может принимать атрибут block. Эффект от определения атрибута blockDefault эквивалентен определению атрибута block в каждом определении типа и объявлении элемента в схеме.

**5 Дополнительные понятия III: квартальный отчет**

Программы формирования заказов и выписки счетов обычно могут генерировать отчеты, в которых указывается, сколько и каких изделий было реализовано в заданном временном диапазоне. Пример такого отчета, за четвертый квартал 1999 года, приведен в 4Q99.xml.

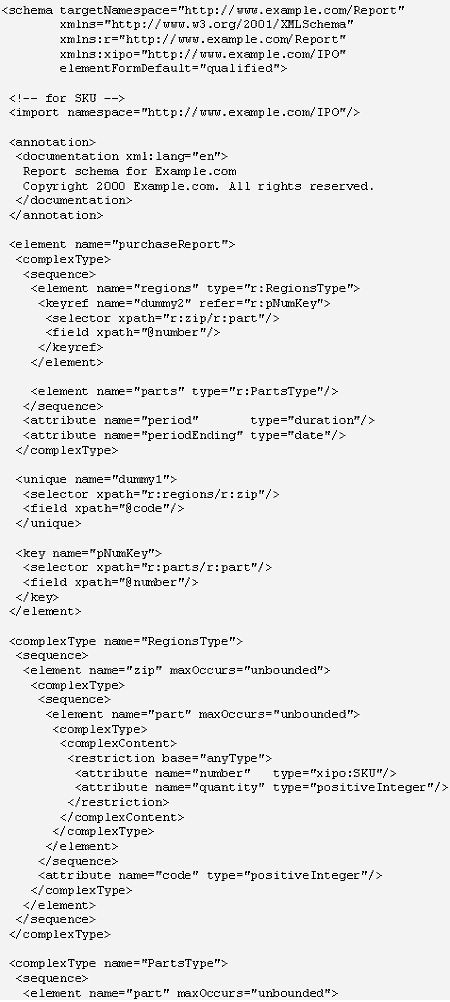
Обратите внимание, что в этом разделе (где возможно), в схеме мы используем квалифицированные элементы, а в документах именное пространство, заданное по умолчанию.

Квартальный отчет 4Q99.xml:



Отчет содержит раздел с данными об изделиях (номер изделия и наименование) и раздел с описанием партий изделий, отгруженных по различным почтовым адресам. Для каждого такого раздела, в отчете имеется соответствующее описание. В разделе отгруженных партий товаров отсутствуют итоговые данные по почтовым адресам. Это связано с действующими ограничениями на оформление отчета. Например, каждый почтовый индекс может появиться только однажды (ограничение уникальности). Точно так же описание каждого отгруженного в данный адрес изделия появляется только однажды, хотя отдельные изделия могут входить в различные партии товара, имеющие разные почтовые индексы (ссылочное ограничение). Например, изделие с номером 455-BX. В последующих разделах мы рассмотрим, как, используя XML-схему задать указанные выше ограничения.

XML-схема отчета, report.xsd:



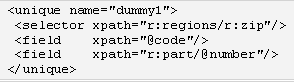
**5.1 Определение уникальности**

XML-схема дает нам возможность указать, что значение атрибута или элемента в пределах некоторой области должны быть уникальны. Для определения уникальности используют элемент unique: во первых, для того чтобы выбрать ("select") набор элементов в пределах которого должна быть обеспечена уникальность, и во вторых чтобы идентифицировать поле ("field") (атрибут или элемент), которое должно быть уникально в пределах выбранного набора элементов. В схеме report.xsd атрибут xpath элемента selector содержит Xpath-выражение равное regions/zip, которое задает список всех элементов zip в отчете. Аналогично, атрибут xpath элемента field содержит второе Xpath-выражение равное @code, которое определяет, что значения атрибута code элементов zip должны быть уникальны. Обратите внимание, что Xpath-выражения ограничивают область того, что должно быть уникально. Отчет мог бы содержать другой атрибут code, но его значение не обязательно должно быть уникальным, если он находится вне области, определенной Xpath-выражениеми. Также обратите внимание, что Xpath-выражения, которые Вы можете использовать в атрибуте xpath, ограничены подмножеством (<http://web2.w3.org/TR/2001/REC-xmlschema-1-20010502/>) полной версии языка Xpath, определенного в XML Path Language 1.0 (<http://www.w3.org/TR/1999/REC-xpath-19991116>).

Мы также можем указать комбинации полей, которые должны быть уникальны. Например, предположим, что мы хотим ослабить ограничение установленное для почтовых индексов. Ранее ограничение состояло в том, что почтовые индексы могли быть указаны только однажды, хотя мы все еще хотим оставить ограничение, состоящее в том, что любое изделие может быть в пределах данного почтового индекса перечислено только однажды. Мы можем достигнуть такого ограничения, определяя, что комбинация почтового индекса и номера изделия должна быть уникальна. В отчете 4Q99.xml, имеются следующие комбинации значений почтового индекса и номеров изделий: {95819 872-AA}, {95819 926-AA}, {95819 833-AA}, {95819 455-BX}, и {63143 455-BX}. Конечно, эти комбинации могут встретиться как в пределах одного списка рассылки, так и в пределах нескольких. Но использование этих комбинаций позволяет однозначно определить изделие, встречающееся в пределах данного почтового индекса более одного раза. Другими словами обработчик схемы сможет обнаружить нарушение уникальности.

Чтобы определить комбинации значений, мы просто добавляем элементы field, которые идентифицируют все возможные значения полей. Чтобы добавить номер изделия к существующему определению, мы включаем еще один элемент field, чей атрибут xpath со значением равным part/@number идентифицирует атрибут number элементов part, которые являются дочерними подэлементами элементов zip. При этом набор уникальных элементов включает элементы zip, отбор которых задан Xpath-выражением regions/zip.

Уникальное составное значение:



**5.2 Определение ключей и ссылок на них**

В ежеквартальном отчете за 1999 год, описание каждой отгруженной партии товара появляется только однажды. Мы могли бы установить это ограничение, используя элемент unique. Однако мы также хотим гарантировать, что каждый элемент, задающий количество изделий, включенных в партию с данным почтовым индексом, имеет соответствующий раздел с описанием изделия. Для установки этого ограничения, используем элементы key и keyref. В схеме отчета, report.xsd, показано применение механизма key и keyref, использующего почти тот же самый синтаксис, что и механизм unique. Элемент key применен к значению атрибута number элемента part, который является подэлементом элемента parts. Объявление атрибута number в качестве ключа означает, что его значение должно быть уникально и не может быть пустым (nil). Имя pNumKey, используется для ссылки на ключ из других мест схемы.

Чтобы гарантировать, что элементы, задающие количество изделий будут иметь соответствующие разделы описания, устанавливаем, что атрибут number ( < field xpath="@number" ) этих элементов ( < selector> zip/part < /selector> ) должен быть объявлен как ссылка (keyref) на ключ с именем pNumKey. Объявление атрибута number как keyref не подразумевает, что его значение должно быть уникально. Это только означает, что он должен иметь то же самое значение, что и ключ с именем pNumKey.

Возможно, вы уже догадались, что по аналогии с unique можно определить комбинации значений keyref и key. Используя этот механизм, мы можем не просто обеспечить эквивалентность идентификаторов изделий в различных разделах отчета, но и определить комбинацию значений, которые должны быть эквивалентны. Такие значения могут включать комбинации значений нескольких типов (string, integer, date, и т.д.), при условии, что порядок и тип ссылок в элементах field одинаков в определениях key и keyref.

**5.3 Ограничения XML-схемы в сравнении с атрибутом ID XML 1.0**

Для обеспечения механизма уникальности XML 1.0 (<http://www.w3.org/TR/2000/REC-xml-20001006>), использует атрибут типа ID и связанные c ним атрибуты типа IDREF и IDREFS. Этот механизм поддержан языком XML-схемы через простые типы ID, IDREF, и IDREFS, которые могут использоваться для того, чтобы объявить атрибуты в стиле XML 1.0. Кроме того, язык XML-схемы вводит новые механизмы, которые более гибки и мощны. Например, они могут быть применены к содержимому любого элемента и атрибута, независимо от его типа. Напротив, ID задает тип атрибута и поэтому не может быть применен к атрибутам, элементам или их содержимому. Кроме того, язык XML-схемы дает возможность определить область, в пределах которой обеспечивается уникальность, тогда как действие ID распространяется на весь документ. Наконец, XML-схема дает возможность создать ключи (key) и ссылки (keyref) из комбинаций содержимого элементов и атрибутов, тогда как ID не имеет такого средства.

**5.4 Импортирование типов**

Схема, report.xsd, использует простой тип xipo:SKU, который определен в другой схеме, и в другом целевом именном пространстве. Механизм повторного вызова, который мы задействовали с помощью элемента include в схеме ipo.xsd, позволяет использовать определения и объявления из address.xsd. В данном случае мы не можем использовать элемент include в report.xsd, потому что он может размещаться в определениях и объявлениях схемы, целевое именное пространство которой то же самое что и целевое именное пространство включаемой подсхемы. Следовательно, элемент include не идентифицирует именное пространство (хотя требует schemaLocation). Механизм импорта, который мы описываем в этом разделе - важный механизм, который дает возможность элементам схемы из различного целевого именного пространства использоваться совместно, и, следовательно, допускает проверку правильности схемы документа, определенной в нескольких именных пространствах. Чтобы импортировать тип SKU и использовать его в схеме отчета, необходимо идентифицировать именное пространство, в котором SKU определен, и ассоциировать это именное пространство с префиксом который будет использован в схеме отчета. Для этого мы используем элемент import, который идентифицирует целевое именное пространство http://www.example.com/IPO элемента SKU, и стандартным способом в элементе schema ассоциируем это именное пространство с префиксом xipo. При этом в определении или объявлении схемы отчета на простой тип SKU, определенный в именном пространстве <http://www.example.com/IPO>, можно сослаться как xipo:SKU.

В нашем примере, мы импортировали один простой тип из одного внешнего именного пространства, и использовали его для объявления атрибутов. Фактически XML-схема разрешает импортировать множество элементов из множества именных пространств, при этом они могут быть упомянуты и в определениях и в объявлениях. Например, в report.xsd мы можем повторно использовать элемент comment, объявленный в ipo.xsd, ссылаясь на тот элемент в объявлении:

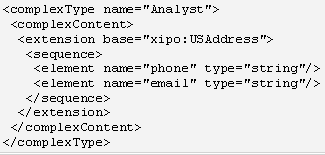
< element ref="xipo:comment"/>

Обратите внимание, что мы не можем многократно использовать элемент shipTo из po.xsd, и приведенный ниже пример является ошибочным, так как только глобальные элементы схемы могут быть импортированы:

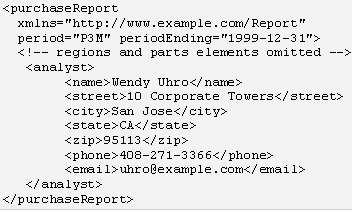
< element ref="xipo:shipTo"/>

В ipo.xsd, comment объявлен как глобальный элемент, другими словами он объявлен как элемент схемы. Напротив, shipTo объявлен локально, другими словами - это элемент, объявленный в комплексном определении типа, а именно PurchaseOrderType. Комплексные типы также могут быть импортированы, и использованы как исходные для образования новых типов. Только поименованные комплексные типы могут быть импортированы; локальные и непоименованные - нет. Предположим, что мы хотим включить в наш отчет наряду с информацией о контракте имя аналитика. Мы можем повторно использовать (глобально определенный), комплексный тип USAddress из address.xsd, и методом расширения получить из него новый тип с именем Analyst. Для этого добавим новые элементы phone и email.

Определение Analyst расширением USAddress:



Используя этот новый тип, объявляем элемент analyst как часть объявления элемента purchaseReport (объявление не показано) в схеме отчета. Тогда, приведенный ниже документ будет соответствовать измененной схеме сообщения. Документ, соответствующий схеме отчета с типом Analyst:



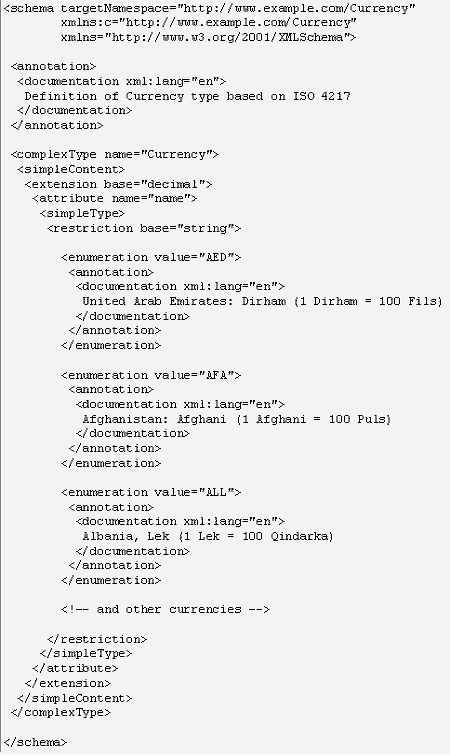
Когда элементы схемы импортированы из нескольких именных пространств, то каждое именное пространство должно быть идентифицировано отдельным элементом import. Элементы import должны быть подэлементами элемента schema. Кроме того, с помощью стандартного объявления именного пространства каждое именное пространство должно быть ассоциировано с префиксом, и префикс должен использоваться для квалификации ссылок к любым элементам схемы, принадлежащим этому именному пространству. Наконец, элементы import содержат необязательный атрибут schemaLocation, который подсказываем местоположение ресурса, связанного с именным пространством. Более подробно атрибут schemaLocation обсудим позднее.

**5.4.1 Библиотеки типов**

Поскольку XML-схемы становятся широко распространенным, то авторам схем понадобится возможность создания простых и комплексных типов, которые могут быть повторно использованы как стандартные блоки для создания новых схем. XML-схема обеспечивает типы, которые играют эту роль, в частности типы, описанные в приложении "Простые типы" и во ведении в библиотеку типов (<http://www.w3.org/2001/03/XMLSchema/TypeLibrary.xsd>).

Собственные библиотеки типов могут создаваться, например, для того, чтобы описать денежные форматы, единицы измерения, адреса, и так далее. Каждая библиотека могла бы состоять из схемы, содержащей одно или более определений. Ниже приводится пример схемы, содержащей определение валют.

Пример определения типов валют в библиотеке типов:



Пример элемента, появляющегося в документе и имеющего указанный выше тип:

< convertFrom name="AFA">199.37< /convertFrom>

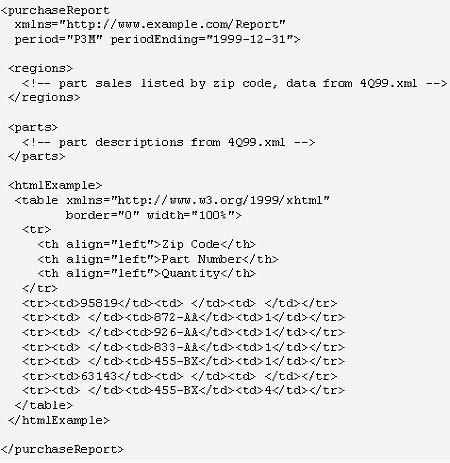
Как только тип валют определен, то он становится доступным для многократного использования в других схемах через только что описанный механизм импорта.

**5.5 Любой элемент, любой атрибут**

В предыдущих разделах мы рассмотрели несколько механизмов для расширения модели содержимого комплексных типов. Например, смешанная модель содержимого может включать в дополнение к элементам произвольные символьные данные, а также элементы, типы которых импортированы из внешнего именного пространства. Однако эти механизмы обеспечивают, соответственно, либо очень широкие, либо очень узкие возможности контроля. Цель этого раздела состоит в том, чтобы описать гибкий механизм, который дает возможность моделям содержимого быть расширенными любыми элементами и атрибутами, принадлежащими указанному именному пространству.

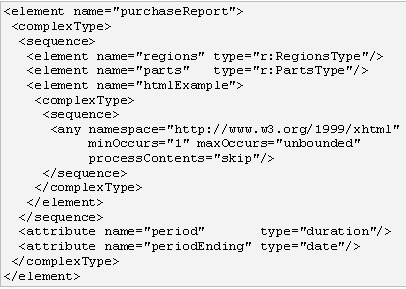
Например, рассмотрим вариант ежеквартального отчета, 4Q99html.xml, в который мы внедрили HTML-представление раздела с данными об отгруженных изделиях. HTML-текст появляется как содержание элемента htmlExample. Для того чтобы все HTML-элементы принадлежали именному пространству HTML (<http://www.w3.org/1999/xhtml>), изменено значение по умолчанию именного пространства у самого внешнего HTML-элементе (table).

Квартальный отчет с HTML, 4Q99html.xml:



Чтобы разрешать появление HTML-элементов в документе, мы изменяем схему отчета, объявляя новый элемент htmlExample, чье содержание определено оператором any. Элемент any определяет, что любой элемент, удовлетворяющий правилам языка XML, допустим в модели содержимого данного типа. В нижеприведенном примере, мы определяем, что XML-элементы принадлежат именному пространству http://www.w3.org/1999/xhtml, другими словами, эти XML-элементы должны быть HTML-элементами. В примере также задано, что в документе может быть не менее одного такого элемента, на что указывают значения minOccurs и maxOccurs.

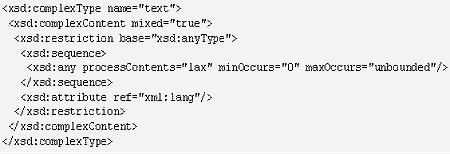
Модификация объявления purchaseReport для включения в документ HTML-фрагмента:



В результате указанного выше изменения, любому XML-элементу, принадлежащему именному пространству <http://www.w3.org/1999/xhtml>, разрешено появляться в элементе htmlExample. В связи с этим 4Q99html.xml допустим, так как в нем есть один элемент table (с дочерними элементами), который является допустимым, и этот элемент появляется в элементе htmlExample, и в документе указано, что элемент table и его содержимое принадлежат требуемому именному пространству. Однако фактически HTML-фрагмент может содержать неверные данные, потому что в 4Q99html.xml ничто не гарантирует их достоверность. Если такая гарантия требуется, то значение атрибута processContents должно быть установлено в strict (значение по умолчанию). В этом случае XML-процессор должен получить схему, связанную с требуемым именным пространством, и проверить правильность HTML-фрагмента, появляющегося внутри элемента htmlExample.

В другом примере, мы определяем тип text, который подобен текстовому типу, определенному во введении в библиотеку типов XML-схемы (<http://www.w3.org/2001/03/XMLSchema/TypeLibrary.xsd>) (см., также подраздел 5.4.1), и подходящий для работы с интернационализированным текстом. Текстовый тип с дополнительным атрибутом xml:lang разрешает произвольную смесь символов и элементов из любого именного пространства, например текст в нотации Ruby (<http://www.w3.org/TR/ruby/>). Значение lax атрибута processContents предписывает XML-процессору проверить правильность содержимого элемента. Процессор проверит правильность элементов и атрибутов, для которых он может получить информацию схемы, но процессор не будет сообщать об ошибках для тех компонент, для которых отсутствует информация схемы.

Тип Text:

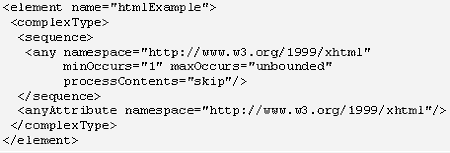


Именное пространство может использоваться различными способами, в зависимости от значения атрибута namespace (см. таблицу 4) для того чтобы разрешить или запретить содержимое элемента:

Таблица 4. Атрибут namespace элемента any

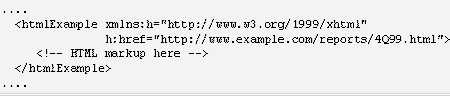
|  |  |
| --- | --- |
| **Значения атрибута Namespace** | **Допустимое содержимое элемента** |
| ##any | Любой XML-элемент из любого именного пространства (значение по умолчанию) |
| ##local | Любой XML-элемент, который не квалифицирован, то есть, не объявлен в именном пространстве |
| ##other | Любой XML-элемент не из целевого именного пространства определяемого типа |
| "http://www.w3.org/1999/xhtml ##targetNamespace" | Любой XML-элемент, принадлежащий любому именному пространству в (разделитель - пробел) списке; ##targetNamespace - псевдоним для обозначения целевого именного пространства определяемого типа |

В дополнение к элементу any, который допускает содержимое элемента согласно именному пространству, имеется похожий элемент anyAttribute, который позволяет элементам иметь произвольные атрибуты. Например, мы можем разрешить любому HTML-атрибуту появиться как часть htmlExample элемента, прибавляя anyAttribute к его объявлению. Модификация объявления htmlExample разрешающая появление HTML-атрибутов:



Это объявление разрешает HTML-атрибуту, скажем href, появляться в htmlExample элементе.

Пример HTML-атрибута у элемента htmlExample:



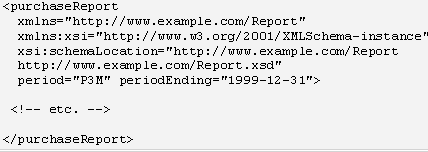
Атрибуту namespace в элементе anyAttribute может быть присвоено любое из значений, перечисленных в таблице 4 для элемента any. У anyAttribute может быть задан атрибут processContents. В отличие от элемента any, элемент anyAttribute не содержит ограничение на число атрибутов, которые могут появиться у элемента.

**5.6 schemaLocation**

XML-схема использует атрибуты schemaLocation и xsi:schemaLocation в трех случаях.

1. В документе атрибут xsi:schemaLocation обеспечивает подсказку автора обработчику документа относительно размещения схемы документов. Автор гарантирует, что схема документов являются уместной для проверки документа на предмет его соответствия данной схеме. Например, мы можем указать для обработчика документа "Квартальный отчет" размещение его схемы. Атрибут schemaLocation содержит пару значений: первая часть пары - задает именное пространство; вторая часть пары - подсказка к первой части, описывающая, где находится схема, соответствующая данному документу. Присутствие этой подсказки не требует от обработчика документа обязательного использования указанной схемы. Обработчик документа свободен в своем выборе: он может использовать другие схемы, полученные любыми подходящими средствами, или не использовать никакой схемы вообще. Схема не обязана иметь именного пространства (см. подраздел 3.4), поэтому имеется атрибут noNamespaceSchemaLocation, который используется, чтобы обеспечить подсказку для указания местонахождения схемы документов, которая не имеет целевого именного пространства.

Применение schemaLocation в документе "Квартальный отчете", 4Q99html.xml:



2. Элемент include языка XML-схемы имеет обязательный атрибут schemaLocation, который содержит URI-ссылку, идентифицирующую схему документа. В результате формируется конечная схема, объединяющая объявления и определения исходной схемы и включенных схем. Например, в разделе 4 для создания единой схемы документа были объединены определения типов Address, USAddress, UKAddress, USState (наряду с объявлениями их атрибутов и локальных элементов) из address.xsd с объявлениям элементов purchaseOrder, comment и определениям типов PurchaseOrderType, Items, SKU (наряду с объявлениями их атрибутов и локальных элементов) из ipo.xsd.

3. Элемент import языка XML-схемы имеет необязательные атрибуты namespace и schemaLocation. Если атрибут schemaLocation задан, то его интерпретируют также как xsi:schemaLocation (см., пункт 1). А именно, он обеспечивает подсказку автора схемы обработчику документа относительно размещения схемы документа, в которой автор гарантирует компоненты именного пространства идентифицированного атрибутом namespace. Чтобы импортировать элементы, которые не находятся ни в каком целевом именном пространстве, элемент import, используются без атрибута namespace (или без атрибута schemaLocation). Ссылки к элементам, импортированным этим способом неквалифицированны. Обратите внимание, что schemaLocation - это только подсказка и некоторые обработчики документов или приложения могут не использовать эти данные. Например, HTML-редактор для работы с HTML-документами может иметь встроенную HTML-схему.

**5.7 Соответствие**

Документ может быть обработан в соответствии с его схемой для того, чтобы проверить, соответствует ли он правилам, указанным в его схеме. Обычно такая обработка делает две вещи: 1) проверяет документ на соответствие правилам. Это называется этапом верификации схемы; 2) добавляет дополнительную информацию, вроде типов и значений по умолчанию, явно не присутствующую в документе. Это называется этапом создания информационной среды (XML-Infoset).

Автор документа, такого как "Заказ на покупку", может объявить в самом документе, что он соответствует правилам данной схемы. Автор может сделать это с помощью рассмотренного выше атрибута schemaLocation. Но независимо от того, присутствует ли атрибут schemaLocation, приложение свободно в обработке документа в независимости от какой либо схемы. Например, программное приложение покупателя может иметь алгоритм, который всегда использует определенную схему заказа на закупку, независимо от значений schemaLocation.

Верификацию можно представить как пошаговую операцию. Сначала проверяется, что корневой элемент документа имеет допустимое содержание. Затем проверяется каждый подэлемент на соответствие его описанию в схеме, и так далее пока весь документ не будет проверен. Обработчик обязан сообщать, какая проверка была выполнена.

Чтобы проверить элемент на соответствие, обработчик сначала определяет местоположение объявления элемента в схеме, затем проверяет, что атрибут targetNamespace в схеме задает действительный URI целевого именного пространства элемента. В противном случае он может решить, что схема не имеет атрибута targetNamespace и элемент в документе неквалифицирован. Предположим, что именное пространство задано, тогда обработчик проверяет тип элемента на соответствие объявлению в схеме или на соответствие значению атрибута xsi:type в документе. В последнем случае, тип, указанный в документе должен быть допустимой заменой для типа, заданного в схеме, что контролируется атрибутом block в объявлении элемента.

Затем обработчик проверяет атрибуты и содержимое элемента, сравнивая их с разрешенными для данного типа элемента атрибутами и содержимым. Например, рассматривая элемент shipTo (см., подраздел 2.1), обработчик проверяет, что он соответствует типу Address (элемент shipTo имеет тип Address).

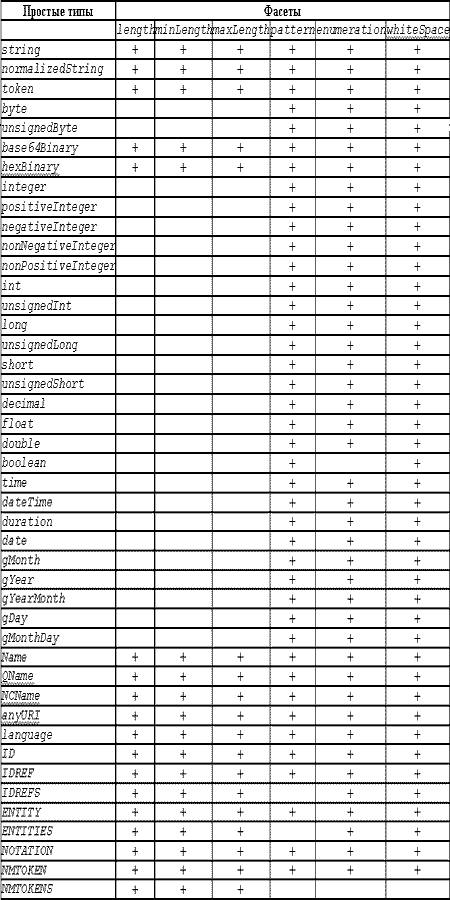
Если элемент имеет простой тип, обработчик проверяет, что элемент не имеет никаких атрибутов или вложенных элементов, и что его символьное содержание соответствует правилам установленным для простого типа. Иногда может выполняться проверка символьной последовательности на наличие регулярных выражений или перечислений, а иногда проверка того, что символьная последовательность представляет значение в разрешенном диапазоне.

Если элемент имеет комплексный тип, то обработчик проверяет, что требуемые атрибуты присутствуют, и что их значения соответствуют требованиям соответствующих им простых типов. Также проверяется, что все требуемые подэлементы присутствуют, и что последовательность подэлементов (и любой смешанный текст) соответствует модели содержимого, объявленной для данного комплексного типа.

**Простые типы и их фасеты**

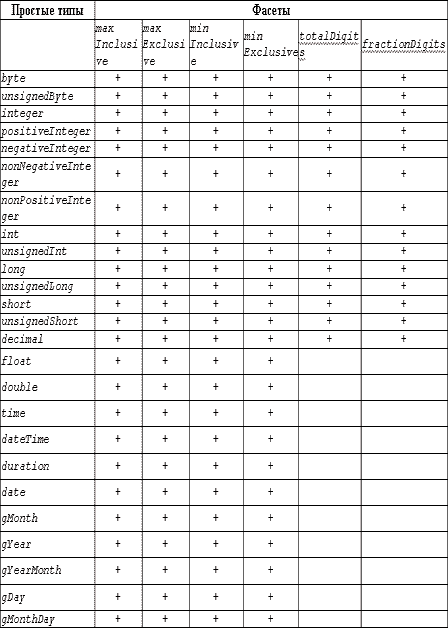
Допустимые значения для каждого простого типа могут ограничиваться через применение одного или более фасетов. Таблицы B1.a и B1.b содержат список всех встроенных в язык XML-схемы простых типов и соответствующих каждому типу фасетов. Более полное описание простых типов и фасетов имеется в документе "XML-схема. Часть 2: Типы данных".

Таблица B1.a. Простые типы и соответствующие им фасеты



Фасеты, перечисленные в Таблице B1.b применяются только к простым типам, которые в ней перечислены. Не все простые типы могут иметь указанные в таблице фасеты и поэтому таблица B1.b содержит не все простые типы.

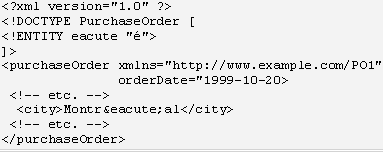
Таблица B1.b. Простые типы и соответствующие им фасеты



**C Использование сущностей**

XML 1.0 обеспечивает различные типы сущностей, которые представляют собой поименованный фрагмент с содержимым, который может использоваться и в DTD и в документах. В подразделе 2.7, мы отметили, как поименованные группы могут использоваться взамен сущностей параметров. В этом разделе мы покажем, как сущности могут быть объявлены в документе, и какие функциональные эквиваленты сущностей могут быть использованы в схемах.

Предположим, что мы хотим объявить и использовать сущность в документе, и что этот документ также (наряду с DTD) ограничивается схемой. Например:

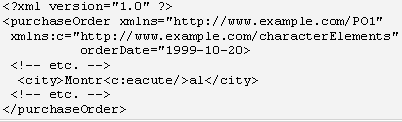


В приведенном выше примере, мы объявляем сущность с именем eacute как часть внутреннего (DTD) набора, и ссылаемся на нее в содержимом элемента city. Обратите внимание, что, когда документ начнет обрабатываться, то сущность будет разыменована прежде, чем начнется проверка на соответствие схеме. Другими словами, обработчик схемы при проверке допустимости содержимого элемента city получит значение Montreal.

Мы можем достигнуть подобного эффекта, но не идентичного результата, объявляя в схеме элемент eacute, и соответственно устанавливая содержание элемента:

< xsd:element name="eacute" type="xsd:token" fixed="e"/>

Этот элемент может следующим образом использоваться в документе:



В этом случае, обработчик схемы обработает два элемента: city и eacute. Обратите внимание, что дополнительный элемент усложнит строковое соответствие: две формы названия "Montreal", приведенные выше при использовании нормальных методов строкового сравнения не будут соответствовать друг другу.

**D Регулярные выражения**

Фасет pattern XML-схемы использует язык регулярных выражений, который поддерживает Unicode (более подробное описание приводится в документе "XML-схема. Часть 2: Типы данных"). Язык регулярных выражений напоминает аналогичный язык, используемый в языке программирования Perl, хотя выражения скорее приспособлены для полных лексических конструкций, а не для лексических конструкций ориентированных на пользователя (строки и параграфы). По этой причине, язык выражений не содержит метасимволов ^ и $, хотя ^ используется, чтобы выразить исключение, например [^0-9]x.

Таблица D1. Примеры регулярных выражений

|  |  |
| --- | --- |
| **Выражение** | **Возможные значения** |
| Chapter \d | Chapter 0, Chapter 1, Chapter 2 .... |
| Chapter\s\d | Строка Chapter, за которой следует один пробельный символ (пробел, табуляция, перевод строки, и т.д.), за которым следует одна цифра |
| Chapter\s\w | Строка Chapter , за которой следует один пробельный символ (пробел, табуляция, перевод строки, и т.д.), за которым следует алфавитный символ (символ или цифра XML 1.0) |
| Espanñola | Espanola |
| \p{Lu} | любой символ верхнего регистра, значение \p{} (например "Lu") определяется в Unicode |
| \p{IsGreek} | Любой греческий символ, конструкция 'Is' может быть применена к любому имени блока (например "Greek") как это определено в Unicode |
| \P{IsGreek} | Любой не греческий символ, конструкция 'Is' может быть применена к любому имени блока (например "Greek") как это определено в Unicode |
| a\*x | x, ax, aax, aaax .... |
| a?x | ax, x |
| a+x | ax, aax, aaax .... |
| (a|b)+x | ax, bx, aax, abx, bax, bbx, aaax, aabx, abax, abbx, baax, babx, bbax, bbbx, aaaax .... |
| [abcde]x | ax, bx, cx, dx, ex |
| [a-e]x | ax, bx, cx, dx, ex |
| [-ae]x | -x, ax, ex |
| [ae-]x | ax, ex, -x |
| [^0-9]x | любой нецифровой символ, за которым следует символ x |
| \Dx | любой нецифровой символ, за которым следует символ x |
| .x | любой символ, за которым следует символ x |
| .\*abc.\* | 1x2abc, abc1x2, z3456abchooray .... |
| ab{2}x | abbx |
| ab{2,4}x | abbx, abbbx, abbbbx |
| ab{2,}x | abbx, abbbx, abbbbx .... |
| (ab){2}x | ababx |

**E Индекс**

Элементы XML-схемы. Названия элемента сопровождаются одной или более ссылками (указывающими номер подраздела) к примерам их использования.

all: 2.7

annotation: 2.6

any: 5.5

anyAttribute: 5.5

appInfo: 2.6

attribute: 2.2

attributeGroup: 2.8

choice: 2.7

complexContent: 2.5.3

complexType: 2.2

documentation: 2.6

element: 2.2

enumeration: 2.3

extension: 2.5.1, 4.2

field: 5.1

group: 2.7

import: 5.4

include: 4.1

key: 5.2

keyref: 5.2

length: 2.3.1

list: 2.3.1

maxInclusive: 2.3

maxLength: 2.3.1

minInclusive: 2.3

minLength: 2.3.1

pattern: 2.3

redefine: 4.5

restriction: 2.3, 4.4

schema: 2.1

selector: 5.1

sequence: 2.7

simpleContent: 2.5.1

simpleType: 2.3

union: 2.3.2

unique: 5.1

Атрибуты XML-схемы. Каждое название атрибута сопровождается одной или более ссылок на то, в каком месте данного документа приводится пример использования атрибута.

abstract: объявление элемента,   
определение комплексного типа;

attributeFormDefault: элемент schema;

base: определение простого типа,   
определение комплексного типа;

block: определение комплексного типа;

blockDefault: элемент schema;

default: объявление атрибута;

default: объявление элемента;

elementFormDefault: элемент schema;

final: определение комплексного типа;

finalDefault: элемент schema;

fixed: объявление атрибута;

fixed: объявление элемента;

fixed: определение простого типа;

form: объявление элемента, объявление атрибута;

itemType: определение типа List;

memberTypes: определение типа Union;

maxOccurs: объявление элемента;

minOccurs: объявление элемента;

mixed: определение комплексного типа;

name: объявление элемента, объявление атрибута,   
определение комплексного типа, определение простого типа;

namespace: элемент any, элемент include;

noNamespaceSchemaLocation: элемент instance;

xsi:nil: элемент instance;

nillable: объявление элемента;

processContents: элемент any, элемент anyAttribute;

ref: объявление элемента;

schemaLocation: спецификация include,   
спецификация redefine, спецификация import;

xsi:schemaLocation: атрибут документа;

substitutionGroup: объявление элемента;

targetNamespace: элемент schema;

type: объявление элемента, объявление атрибута;

xsi:type: элемент instance;

use: объявление атрибута;

xpath: элементы selector & field.