

**DTD**



# XML 문서의 검증

## ❖ 제대로 만든 XML이 되려면

### ● Well-formed Document

- ◆ XML의 기본 문법이 정확할 것

### ● Valid Document

- ◆ Well-formed Document인가?
- ◆ DTD 또는 Schema에 정한 형식대로 잘 작성되어 있는가
  - 문서의 구조와 적절한 요소, 속성들의 개수, 순서 등을 정의
- ◆ 프로그램 방식으로 데이터 처리 시 문서가 valid 해야 안정성 제공 가능

### ● DTD: Document Type Definition

- ◆ 하나의 시스템 내에서 사용할 XML 데이터의 구조 정의

### ● Schema:

- ◆ 서로 다른 시스템 사이의 데이터를 주고 받아 사용할 때 유용

# DTD vs Schema

## ❖ DTD vs Schema

	DTD	스키마
문법	XML과 유사한 문법	XML 1.0 표준에 만족
DOM 지원	DOM 기술을 지원하지 않음	DOM을 통한 조작 가능
컨텐츠 모델	순차, 선택 리스트만 제공	순차, 리스트 복합적으로 사용 가능
데이터 타입	문자열, 토큰, ID와 그 외에 한정된 타입 지원	문자열, 숫자, 날짜/시간 및 새로운 형식 정의를 할 수 있음
Namespace	전역 이름만 사용	전역/ 로컬 이름 모두 사용 가능
상속성	제공하지 않음	제공
확장성	한계가 있음	제한 없음
기본 제약 조건	있음	없음
동적 스키마	불가능 - 읽기만 가능	가능 - 런타임에 상호작용으로 변경 가능

- Schema는 DTD를 대체하고 있음

## ❖ DTD를 사용하는 이유

- 응용 프로그램은 DTD를 사용해 XML 데이터가 유효한지 검증 가능
  - ◆ 독립적인 그룹의 사람들이 표준 DTD를 이용해 데이터 교환이 가능케 함

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE note [
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
```

이 형식대로 작성해야 valid

```
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend</body>
</note>
```

내부 DTD의 예

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM "note.dtd">
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

외부 DTD의 예

# DTD – XML building block

## ❖ DTD 관점에서 XML 구성 요소

- **element:** <>로 감싸진 태그

```
<body>some text</body>
```

```
<message>some text</message>
```

- **attribute:** element가 갖는 추가적인 정보

```

```

- **Entity:** xml 문서 내에서 특별한 의미를 갖는 문자들

Entity References	Character	Entity References	Character
&lt;	<	&gt;	>
&quot;	"	&apos;	'
&amp;	&		

- **Pcdata(Parced Character DATA ):** 시작/종료 태그 사이의 텍스트

- ◆ 파서에 의해 파싱되는 텍스트로 entity와 다른 markup으로 구성되었는지 점검됨

- **cdata:** 파서에 의해 파싱되지 않는 텍스트

- ◆ 요소의 콘텐츠에는 PCDATA만, 속성의 값에는 CDATA만 사용 가능

# DTD - Elements

## ❖ DTD에서의 element 선언

```
<!ELEMENT element-name category>  
<!ELEMENT element-name (element-content)>
```

### ● 비어있는 elements

```
<!ELEMENT element-name EMPTY>
```

Example: <!ELEMENT br EMPTY>

XML example: <br />

### ● pCDATA를 갖는 elements

```
<!ELEMENT element-name (#PCDATA)>
```

Example: <!ELEMENT from (#PCDATA)>

### ● 어떤 contents라도 허용하는 elements

```
<!ELEMENT element-name ANY>
```

Example: <!ELEMENT note ANY>

# DTD - Elements

## ❖ DTD에서의 element 선언

- 순서대로 등장하는 자식을 갖는 elements

`<!ELEMENT element-name (child1)>`

or

`<!ELEMENT element-name (child1,child2,...)>`

Example: `<!ELEMENT note (to,from,heading,body)>`

- 선택적으로 자식을 갖는 element

`<!ELEMENT note (to,from,header,(message|body))>`

- 0개 이상의 여러 형태의 내용을 갖는 element

`<!ELEMENT note (#PCDATA|to|from|header|message)*>`

# DTD - Elements

## ❖ 등장 회수의 지정

### ● 1회만 등장

`<!ELEMENT element-name (child-name)>`

Example: `<!ELEMENT note (message)>`

### ● 최소 1회 이상 등장

`<!ELEMENT element-name (child-name+)>`

Example: `<!ELEMENT note (message+)>`

### ● 0회 이상 등장

`<!ELEMENT element-name (child-name*)>`

Example: `<!ELEMENT note (message*)>`

### ● 0회 또는 1회 등장

`<!ELEMENT element-name (child-name?)>`

Example: `<!ELEMENT note (message?)>`



# DTD - Attributes

## ❖ 기본 Attribute 선언 방식

`<!ATTLIST element-name attribute-name attribute-type attribute-value>`

DTD example: `<!ATTLIST payment type CDATA "check">`

XML example: `<payment type="check" />`

## ● attribute 타입 항목

타입	설명	타입	설명
CDATA	값은 CDATA이다.	( <i>en1 en2 ..</i> )	값은 en1, en2, ... 중 하나이다.
ID	값은 유일한 id이다.		
IDREF	값은 다른 element의 id이다.	IDREFS	값은 다른 id들의 목록이다.
ENTITY	값은 entity이다.	Entity	값은 entity의 목록이다.

## ● attribute value 항목

Value	설명	Value	설명
<i>value</i>	미 선언 시 사용할 attribute의 기본 값	#FIXED <i>value</i>	attribute의 값이 고정됨
#REQUIRED	attribute에 값이 필요함	#IMPLIED	attribute의 값이 옵션임

# DTD - Attributes

## ❖ Attribute 적용 예

DTD: `<!ATTLIST square width CDATA "0">`

Valid XML: `<square width="100" />`

DTD: `<!ATTLIST person number CDATA #REQUIRED>`

Valid XML: `<person number="5677" />`

Invalid XML: `<person />`

DTD: `<!ATTLIST contact fax CDATA #IMPLIED>`

Valid XML: `<contact fax="555-667788" />`

Valid XML: `<contact />`

DTD: `<!ATTLIST sender company CDATA #FIXED "Microsoft">`

Valid XML: `<sender company="Microsoft" />`

Invalid XML: `<sender company="W3Schools" />`

DTD: `<!ATTLIST payment type (check|cash) "cash">`

XML example: `<payment type="check" />`

`<payment type="cash" />`

# DTD - Entity

## ❖ Entity

- 특별한 문자열에 대한 short cut 지정
- &gt; 등 기본 내장 Entity
- internal entity

`<!ENTITY entity-name "entity-value">`

DTD Example: `<!ENTITY writer "Donald Duck.">`  
`<!ENTITY copyright "Copyright W3Schools.">`

XML example: `<author>&writer;&copyright;</author>`

- external entity

`<!ENTITY entity-name SYSTEM "URI/URL">`

DTD Example:  
`<!ENTITY writer SYSTEM "https://www.w.com/Entity.dtd">`  
`<!ENTITY copyright SYSTEM "https://www.w.com/Entity.dtd">`

XML example: `<author>&writer;&copyright;</author>`

# DTD - Examples

## ❖TVSCHEDULE

```
<!DOCTYPE TVSCHEDULE [  
  
  <!ELEMENT TVSCHEDULE (CHANNEL+)>  
  <!ELEMENT CHANNEL (BANNER, DAY+)>  
  <!ELEMENT BANNER (#PCDATA)>  
  <!ELEMENT DAY (DATE, (HOLIDAY!PROGRAMSLOT+)+)>  
  <!ELEMENT HOLIDAY (#PCDATA)>  
  <!ELEMENT DATE (#PCDATA)>  
  <!ELEMENT PROGRAMSLOT (TIME, TITLE, DESCRIPTION?)>  
  <!ELEMENT TIME (#PCDATA)>  
  <!ELEMENT TITLE (#PCDATA)>  
  <!ELEMENT DESCRIPTION (#PCDATA)>  
  
  <!ATTLIST TVSCHEDULE NAME CDATA #REQUIRED>  
  <!ATTLIST CHANNEL CHAN CDATA #REQUIRED>  
  <!ATTLIST PROGRAMSLOT VTR CDATA #IMPLIED>  
  <!ATTLIST TITLE RATING CDATA #IMPLIED>  
  <!ATTLIST TITLE LANGUAGE CDATA #IMPLIED>  

```

# DTD - Examples

## ❖ NEWSPAPER

```
<!DOCTYPE NEWSPAPER [  
  
  <!ELEMENT NEWSPAPER (ARTICLE+)>  
  <!ELEMENT ARTICLE (HEADLINE,BYLINE,LEAD,BODY,NOTES)>  
  <!ELEMENT HEADLINE (#PCDATA)>  
  <!ELEMENT BYLINE (#PCDATA)>  
  <!ELEMENT LEAD (#PCDATA)>  
  <!ELEMENT BODY (#PCDATA)>  
  <!ELEMENT NOTES (#PCDATA)>  
  
  <!ATTLIST ARTICLE AUTHOR CDATA #REQUIRED>  
  <!ATTLIST ARTICLE EDITOR CDATA #IMPLIED>  
  <!ATTLIST ARTICLE DATE CDATA #IMPLIED>  
  <!ATTLIST ARTICLE EDITION CDATA #IMPLIED>  
  
  <!ENTITY NEWSPAPER "Vervet Logic Times">  
  <!ENTITY PUBLISHER "Vervet Logic Press">  
  <!ENTITY COPYRIGHT "Copyright 1998 Vervet Logic Press">  
]>
```

# Schema – 기본 스키마 문서

## ❖ Simple Schema

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="https://www.example.com"
  xmlns="https://www.example.com"
  elementFormDefault="qualified">
```

note.xsd

```
<xs:element name="note">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      <xs:element name="from" type="xs:string"/>
      <xs:element name="heading" type="xs:string"/>
      <xs:element name="body" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

# Schema – 기본 스키마 문서

## ❖ <schema> 태그의 선언 속성

### ● XML Schema 파일의 root element

특성	의미
xmlns:xs	스키마 문서에서 사용될 element와 데이터 타입의 출처 (namespace)를 밝히며 prefix 지정
targetNamespace	이 스키마에 선언된 태그들(to, from, note, ..)의 출처 (namespace)를 밝힘 → XML 문서에서 사용
xmlns	접두사를 사용하지 않는 문서의 기본 namespace 지정 일반적으로 targetNamespace 사용
elementFormDefault	qualified 또는 unqualified 사용 qualified: 모든 요소는 targetNamespace에서 선언한 네임스페이스를 사용해야 함 Unqualified: targetNamespace에서 선언한 네임스페이스를 사용하지 않아도 됨

# Schema - 참조

## ❖ XML 문서에서 Schema 참조

```
<?xml version="1.0"?>
<note xmlns="https://www.example.com"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="https://www.example.com note.xsd">

  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

특성	의미
xmlns	문서의 태그들이 기술된 기본 namespace 지정
xmlns:xsi	XML Schema 를 만들 때 사용한 태그들이 기술된 namespace 여기서는 schemaLocation태그를 사용하기 위해 필요
xsi:schemaLocation	사용할 namespace와 사용할 xsd 파일을 공백으로 구분해서 표시



# Schema – element 선언

## ❖ Simple element

- 단지 텍스트 값으로만 이루어진 element로 다양한 타입 선언 가능

```
<xs:element name="xxx" type="yyy"/>
```

- 타입의 종류

- ◆ xs:string

- ◆ xs:decimal

- ◆ xs:integer

- ◆ xs:boolean

- ◆ xs:date

- ◆ xs:time

```
<xs:element name="lastname" type="xs:string"/>
```

```
<xs:element name="age" type="xs:integer"/>
```

```
<xs:element name="dateborn" type="xs:date"/>
```

```
<lastname>Refsnes</lastname>
```

```
<age>36</age>
```

```
<dateborn>1970-03-27</dateborn>
```

- 기본 값과 고정값 표현

```
<xs:element name="color" type="xs:string" default="red"/>
```

```
<xs:element name="color" type="xs:string" fixed="red"/>
```

# Schema – attribute 선언

## ❖ attribute 선언

```
<xs:attribute name="xxx" type="yyy"/>
```

### ● 타입의 종류(element와 동일)

◆ xs:string

```
<xs:attribute name="lang" type="xs:string"/>
```

◆ xs:decimal

◆ xs:integer

◆ xs:boolean

```
<lastname lang="EN">Smith</lastname>
```

◆ xs:date

◆ xs:time

### ● 기본 값과 고정 값 표현, 필수 표현

```
<xs:attribute name="lang" type="xs:string" default="EN"/>
```

```
<xs:attribute name="lang" type="xs:string" fixed="EN"/>
```

```
<xs:attribute name="lang" type="xs:string" use="required"/>
```

◆ 기본은 필수 입력 아님

# Schema – 값의 제한

## ❖ restriction

### ● 제약 타입들

Constraint	Description
enumeration	사용 가능한 값들의 열거
length	문자열 또는 리스트 아이템의 개수로 0 이상의 값
maxExclusive	사용하려는 숫자 값의 상한 지정(포함하지는 않음: >)
maxInclusive	사용하려는 숫자 값의 상한 지정(포함함: >=)
maxLength	문자열 또는 리스트 아이템의 최대 개수로 0 이상의 값
minExclusive	사용하려는 숫자 값의 하한 지정(포함하지는 않음: <)
minInclusive	사용하려는 숫자 값의 하한 지정(포함함: <=)
minLength	문자열 또는 리스트 아이템의 최소 개수로 0 이상의 값
pattern	문자열의 패턴을 정규 표현식 형태로 표현
totalDigits	전체 자리수로 0보다 큰 값
whiteSpace	공백 문자에 대한 처리 규칙(preserve, replace, collapse)

# Schema – 값의 제한

## ❖ restriction

- XML 요소가 가질 수 있는 값의 제약 사항
- 값에 대한 제약

```
<xs:element name="age">  
  <xs:simpleType>  
    <xs:restriction base="xs:integer">  
      <xs:minInclusive value="0"/>  
      <xs:maxInclusive value="120"/>  
    </xs:restriction>  
  </xs:simpleType>  
</xs:element>
```

Element 타입 지정

age의 값은 integer 타입인데  
 $0 \leq \text{age} \leq 120$ 이다.

# Schema – 값의 제한

## ❖ restriction

### ● 입력 가능한 값의 제약

```
<xs:element name="car">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="Audi"/>
      <xs:enumeration value="Golf"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

car의 값은 문자열인데  
Audi, Golf 중 하나

```
<xs:element name="car" type="carType"/>
```

```
<xs:simpleType name="carType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Audi"/>
    <xs:enumeration value="Golf"/>
  </xs:restriction>
</xs:simpleType>
```

별도의 사용자 지정 타입 사용  
- 다른 element에서 재사용 가능

# Schema – 값의 제한

## ❖ restriction

### ● 정규 표현식을 이용한 패턴 지정

```
<xs:element name="letter">  
  <xs:simpleType>  
    <xs:restriction base="xs:string">  
      <xs:pattern value="[a-z]"/>  
    </xs:restriction>  
  </xs:simpleType>  
</xs:element>
```

소문자 1자

### ● 다양한 패턴의 활용

- ◆ [xyz]: x 또는 y 또는 z
- ◆ [a-zA-Z][A-Z][0-9][가-힣] : 대/소문자1, 대문자 1자, 0~9의 숫자 1, 한글 1자
- ◆ ([a-z])\* : 소문자 0개 이상
- ◆ ([a-zA-Z])+ : 소문자 하나, 대문자 하나가 1개 이상 : aB, aBcD...
- ◆ male|female: male 또는 female
- ◆ [a-zA-Z0-9]{8}: 대소문자, 숫자 8자
  - {8, }: 8자 이상, {8, 10}: 8~10자

# Schema – 값의 제한

## ❖ restriction

### ● 공백 문자 처리

```
<xs:element name="address">  
  <xs:simpleType>  
    <xs:restriction base="xs:string">  
      <xs:whiteSpace value="preserve"/>  
    </xs:restriction>  
  </xs:simpleType>  
</xs:element>
```

공백을 있는 그대로 처리

### ● 사용 가능한 값

- ◆ preserve: 기존의 공백문자를 그대로 유지
- ◆ replace: line feeds, tabs, spaces, carriage returns 등을 space로 변경
- ◆ collapse: 맨 앞, 뒤의 공백은 제거되고 나머지 모든 공백 문자들은 공백 하나로 대체

# Schema – 값의 제한

## ❖ restriction

### ● 길이에 대한 제약

```
<xs:element name="password">  
  <xs:simpleType>  
    <xs:restriction base="xs:string">  
      <xs:length value="8"/>  
    </xs:restriction>  
  </xs:simpleType>  
</xs:element>
```

8글자로 한정

```
<xs:element name="password">  
  <xs:simpleType>  
    <xs:restriction base="xs:string">  
      <xs:minLength value="5"/>  
      <xs:maxLength value="8"/>  
    </xs:restriction>  
  </xs:simpleType>  
</xs:element>
```

최소 5글자에서  
최대 8글자



# Schema – 복합 요소의 표현

## ❖ Complex Element

- 다른 element 또는 attribute를 갖는 element

- ◆ 빈 element

```
<product pid="1345"/>
```

- ◆ 다른 element를 포함하는 element

```
<employee>  
  <firstname>John</firstname>  
  <lastname>Smith</lastname>  
</employee>
```

- ◆ 단순한 문자열과 속성을 갖는 element

```
<food type="dessert">Ice cream</food>
```

- ◆ element와 text를 갖는 element

```
<description>  
  It happened on <date lang="norwegian">03.03.99</date> ....  
</description>
```

# Schema – 복합 요소의 표현

## ❖ Complex Element의 선언

### ● element 선언에 직접 선언하는 방식

```
<xs:element name="employee">  
  <xs:complexType>  
    <xs:sequence>  
      <xs:element name="firstname" type="xs:string"/>  
      <xs:element name="lastname" type="xs:string"/>  
    </xs:sequence>  
  </xs:complexType>  
</xs:element>
```

complexType: 복합 타입 선언

### ● 별도의 타입으로 지정하는 방식

```
<xs:element name="employee" type="personinfo"/>  
  
<xs:complexType name="personinfo">  
  <xs:sequence>  
    <xs:element name="firstname" type="xs:string"/>  
    <xs:element name="lastname" type="xs:string"/>  
  </xs:sequence>  
</xs:complexType>
```

동일한 구성의 여러  
element 정의에서 재사용 가능

# Schema – 복합 요소의 표현

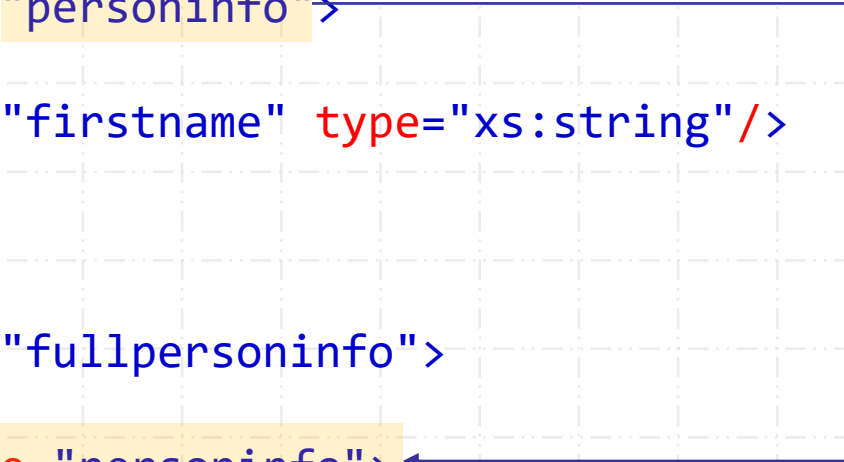
## ❖ Complex Element의 선언

### ● 다른 타입의 상속

```
<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="fullpersoninfo">
  <xs:complexContent>
    <xs:extension base="personinfo">
      <xs:sequence>
        <xs:element name="address" type="xs:string"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:element name="employee" type="fullpersoninfo"/>
```



A blue line with an arrow originates from the `<xs:extension base="personinfo">` line in the `fullpersoninfo` definition and points to the `<xs:complexType name="personinfo">` definition, illustrating the inheritance relationship.

**complexContent:** complexType의 내용을 제한하거나 확장할 때 사용

# Schema – 복합 요소의 표현

## ❖ Complex Empty Element

```
<product prodid="1345" />
```

```
<xs:element name="product">
```

```
  <xs:complexType>
```

```
    <xs:attribute name="prodid" type="xs:positiveInteger"/>
```

```
  </xs:complexType>
```

```
</xs:element>
```

# Schema – 복합 요소의 표현

## ❖ Containing Elements Only

- xs:sequence

- ◆ 하위 element 들이 순서대로 나와야함

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<person>
  <firstname>John</firstname>
  <lastname>Smith</lastname>
</person>
```

# Schema – 복합 요소의 표현

## ❖ Text-Only Elements

- xs:simpleContent 내에서 기존 요소에 제약 (restriction) 을 가하거나 확장 (extension)해서 사용

```
<xs:element name="shoesize">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:integer">
        <xs:attribute name="country" type="xs:string" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<shoesize country="france">35</shoesize>
```

# Schema – 복합 요소의 표현

## ❖ with Mixed Content

- attributes, elements, text를 모두 포함하는 element

```
<letter>
```

Dear Mr. <name>John Smith</name>.

Your order <orderid>1032</orderid>

will be shipped on <shipdate>2001-07-13</shipdate>.

```
</letter>
```

```
<xs:element name="letter">
```

```
  <xs:complexType mixed="true">
```

```
    <xs:sequence>
```

```
      <xs:element name="name" type="xs:string"/>
```

```
      <xs:element name="orderid" type="xs:positiveInteger"/>
```

```
      <xs:element name="shipdate" type="xs:date"/>
```

```
    </xs:sequence>
```

```
  </xs:complexType>
```

```
</xs:element>
```

# Schema - Indicators

## ❖ Indicators(지시자)

### ● 순서 관련 : Element의 순서를 지정

indicator	설명
all	하위 요소들이 아무런 순서 없이 모두 등장할 수 있음
choice	하위 요소들 중 하나만 선택해서 등장할 수 있음
sequence	하위 요소들이 순서대로 등장해야 함

```
<xs:element name="person">
  <xs:complexType>
    <xs:all>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    </xs:all>
  </xs:complexType>
</xs:element>
```



# Schema - Indicators

## ❖ Indicators(지시자)

- 그룹 관련: 관련된 element 또는 attribute의 그룹 지정

indicator	설명
group name	여러 element 들을 그룹핑해서 하나로 사용
attributeGroup name	여러 attribute들을 그룹핑해서 하나로 사용

```
<xs:group name="persongroup">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
</xs:group>
<xs:element name="person" type="personinfo"/>
<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:group ref="persongroup"/>
    <xs:element name="nickname" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

# Schema - Indicators

## ❖ Indicators(지시자)

### ● 등장 회수 관련

indicator	설명
maxOccurs	최대 등장 회수 지정, unbounded를 사용하면 계속 추가 가능
minOccurs	최소 등장 회수 지정

### ◆ 순서 및 그룹 관련 지시자에서 기본 값은 모두 1

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="full_name" type="xs:string"/>
      <xs:element name="child_name" type="xs:string"
        minOccurs="10" maxOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

child\_name은 0명 이상 10명 까지

# Schema - <any>

## ❖ <any> Element

- 기존에 스키마에 선언된 element 이외의 어떤 요소로도 확장 가능

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
      <xs:any minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

```
<person>
  <firstname>Hege</firstname>
  <lastname>Refsnes</lastname>
  <children>
    <childname>Cecilie</childname>
  </children>
</person>
```

```
<person>
  <firstname>Stale</firstname>
  <lastname>Refsnes</lastname>
</person>
```

<children> element가 선언되어 있다면..

# Schema - <anyAttribute>

## ❖ <anyAttribute>

- 기존에 스키마에 선언된 attribute 이외의 어떤 요소로도 확장 가능

```
<xs:element name="person">  
  <xs:complexType>  
    <xs:sequence>  
      <xs:element name="firstname" type="xs:string"/>  
      <xs:element name="lastname" type="xs:string"/>  
    </xs:sequence>  
    <xs:anyAttribute/>  
  </xs:complexType>  
</xs:element>
```

```
<person eyecolor="green">  
  <firstname>Hege</firstname>  
  <lastname>Refsnes</lastname>  
</person>
```

```
<person eyecolor="blue">  
  <firstname>Stale</firstname>  
  <lastname>Refsnes</lastname>  
</person>
```