XML Schema

Esempio: il file XML

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

- Cosa deve specificare il DTD?
- L'elemento `message` è composto da:
  - Un elemento `to` contenente testo
  - Un elemento `from` contenente testo
  - Un elemento `heading` contenente testo
  - Un elemento `body` contenente testo

Che cos'è XML Schema?

- E' un'alternativa ai DTD basata su XML
  - Gli schemi XML (XSD) sono in formato XML quindi possono essere analizzati da un parser XML
- Un XML Schema permette di definire:
  - Elementi
  - Attributi
  - Quali elementi sono elementi figli
  - L'ordine e il numero degli elementi figli
  - Se un elemento è vuoto, oppure contiene testo o altri elementi
  - **Tipi di dati** per elementi e attributi
  - Valori fissi o di default per elementi e attributi

Estensibilità

- Creazione di tipi di dato personalizzati tramite derivazione dai tipi di dato disponibili
- Utilizzo di più schemi per la validazione di un singolo documento
- Riutilizzo di schemi in altri schemi

Gestione dei tipi

- Importante la possibilità di gestire in modo completo e flessibile i tipi:
  - Supporto per i tipi di dati primitivi e possibilità di crearne di nuovi
  - Supporto i namespace
  - Supporto per l'ereditarietà dei tipi ed il polimorfismo
- È possibile descrivere il contenuto in maniera puntuale: integer, float, date, string, …
- È possibile lavorare in modo sicuro con dati estratti da DB: Strong Typing
- È semplice la definizione di restrizioni sui dati: espressioni regolari, enumerativi, numero caratteri, intervalli numerici, …

Esempio: il file XML

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

- Cosa deve specificare lo schema?
- L'elemento `message` è composto da:
  - Un elemento `to` contenente una stringa
  - Un elemento `from` contenente una stringa
  - Un elemento `heading` contenente una stringa
  - Un elemento `body` contenente una stringa

Esempio: il file XSD

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema
 xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="message" type="messageType"/>
  <xs:complexType name="messageType">
    <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:schema>
```

- E' un documento XML → è costituito da elementi
- Gli elementi svolgono un ruolo analogo alle dichiarazioni nei DTD

Gli elementi dell'XSD

- L'elemento `schema`:
  - E' la radice dei documenti XSD
  - Contiene la dichiarazione del **namespace** degli schemi
- Altre dichiarazioni:
  - Elemento `element`: dichiarazione di elemento di nome name e di tipo type
  - Elemento `complexType`: definizione di tipo di nome name
  - Elemento `sequence`: specifica di un content-model di tipo sequenza

Collegamento di un XML ad uno schema

- Il collegamento allo schema viene fatto mediante un attributo inserito nel tag dell'elemento radice:

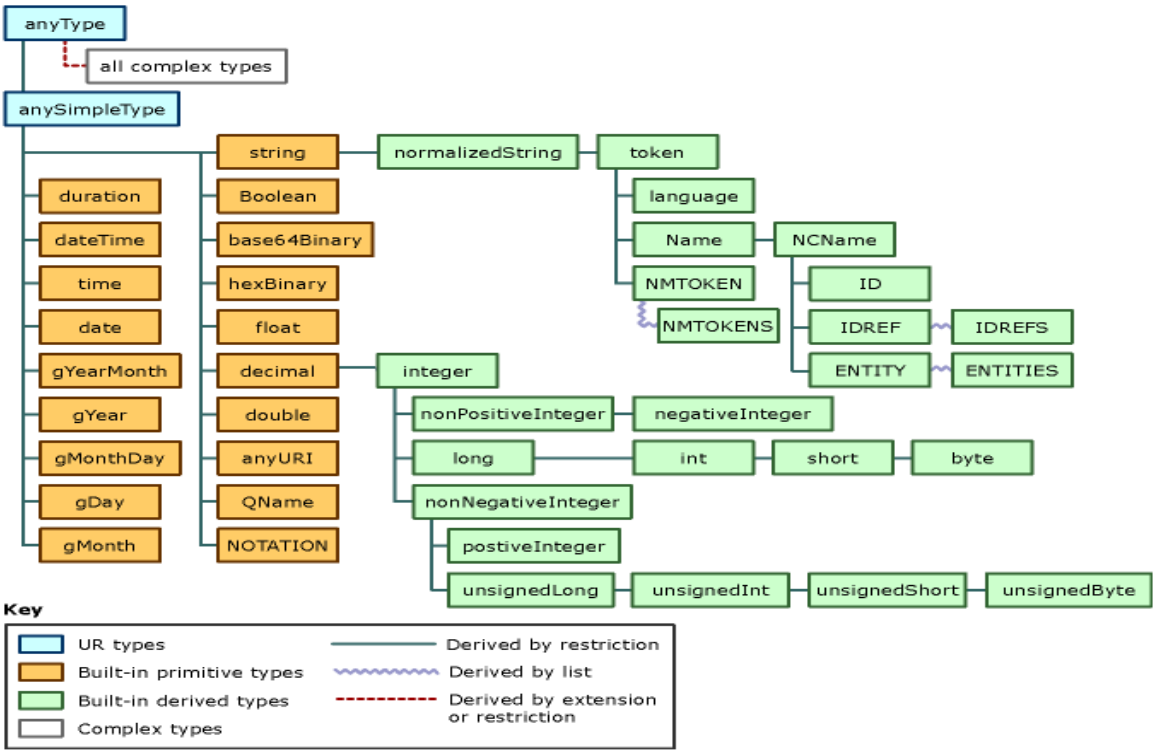
```
<?xml version="1.0"?>
<message
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="http://mysite.it/msg.xsd">
  <to>Bob</to>
  <from>Janet</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend</body>
</message>
```

- **Attenzione:** è solo un collegamento e non implica la validazione automatica: la cosa importante è il **namespace**
- Il documento XML associato ad uno schema prende il nome di **documento istanza**

Tipi di dati (Data Type)

- XML schema permette di attribuire un tipo ad elementi ed attributi (sono come i tipi di Java)
- **Tipi semplici** (simpleType): valore
  - **Tipi primitivi**: predefiniti nella specifica XML Schema (string, float, integer, date…)
  - **Tipi derivati**: sono definiti in termini di tipi primitivi (derivazione per restrizione)
- **Tipi complessi** (complexType): struttura
  - Definizione di nuovi tipi "da zero"
  - Derivazione per estensione o restrizione
- Gli elementi possono essere di tipo semplice o complesso mentre gli attributi possono essere solo di tipo semplice

Tassonomia dei tipi di dati



Definizione e dichiarazione

- Vale anche qui la distinzione fra definizione e dichiarazione che troviamo nei linguaggi di programmazione (es. C)
  - **Definizione**: crea un nuovo tipo di dato semplice o complesso
  - **Dichiarazione**: fa riferimento ad una definizione per creare un'istanza
- La definizione di un tipo può essere inline nella dichiarazione: **definizione anonima**
- Una dichiarazione ha la seguente sintassi:

```
<xs:element name="elementName"
            type="elementType" />
```

Esempio: definizione e dichiarazione

Definizione

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <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 name="message" type="messageType"/>

</xs:schema>
```

Dichiarazione

Esempio: definizione inline

Dichiarazione

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="message">

    <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>
```

Definizione inline

Dichiarazione di tipo semplice predefinito

- XSD

```
<xs:element name="Nome" type="xs:string" />
<xs:element name="Eta" type="xs:positiveInteger" />
<xs:element name="DataNascita" type="xs:date" />
```

- Istanza

```
<Nome>Gabriele</Nome>
<Eta>31</Eta>
<DataNascita>1971-10-07</DataNascita>
```


Dichiarazione di tipo complesso

- XSD

```
<xs:complexType name="PersonaType">
  <xs:sequence>
    <xs:element name="Nome" type="xs:string"/>
    <xs:element name="DataNascita" type="xs:date"/>
  </xs:sequence>
</xs:complexType>
<xs:element name="Persona" type="PersonaType"/>
```

- Istanza

```
<Persona>
  <Nome>Gabriele</Nome>
  <DataNascita>1971-10-07</DataNascita>
</Persona>
```

Definizione inline di tipo complesso

- XSD

```
<xs:element name="Persona">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Nome" type="xs:string"/>
      <xs:element name="DataNascita" type="xs:date"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

- Istanza

```
<Persona>
  <Nome>Gabriele</Nome>
  <DataNascita>1971-10-07</DataNascita>
</Persona>
```


Tipi semplici: elementi costitutivi

- Un tipo di dato consiste di:
  - **Uno spazio dei valori**: insieme dei valori che un certo tipo di dato può assumere
  - **Uno spazio lessicale**: rappresentazioni dei valori che un certo tipo di dato può assumere (insieme delle stringhe che rappresentano i valori)
  - **Un insieme di facet** (aspetti): un facet è una proprietà che definisce il tipo di dato. Normalmente si utilizzano per restringere lo spazio dei valori del tipo base e creare un tipo derivato

Tipi predefiniti

- `string`: stringa di caratteri esclusi i caratteri di controllo di XML
- `decimal`: numero di precisione arbitraria (xxx.yy)
  - Tipi derivati: `integer`, `positiveInteger`, `negativeInteger`, …
- `float`: numero reale a singola precisione (32 bit)
- `double`: numero reale a doppia precisione (64 bit)
- `boolean`: valore logico true o false
- `dateTime`: rappresenta uno specifico momento temporale nel formato `CCYY-MM-DDThh:mm:ss`
- `date`: rappresentazione di una data
- `time`: rappresentazione di un'ora
- Esistono altri tipi per rappresentare URI, colori ecc.

Derivazione di tipi semplici

- Sono DataType di tipo "valore"
- Gli elementi di tipo semplice possono contenere solo "caratteri alfanumerici" e non altri elementi
- La definizione di nuovi tipi avviene derivando per restrizione dai tipi predefiniti
- La restrizione avviene specificando vincoli (facet) sullo spazio dei valori o sullo spazio lessicale
- La sintassi per definire un tipo semplice derivato è:

```
<xs:simpleType name="derivedType">
  <xs:restriction base="baseType">
    …facets…
  </xs:restriction>
</xs:simpleType>
```

Derivazione di tipi semplici

- È possibile derivare anche da tipi definiti dall'utente
- Vale il **subtyping** (ereditarietà di interfaccia): un'istanza del tipo di dato derivato verrà validata correttamente se utilizzata al posto di un'istanza del tipo di dato base
- È possibile impedire la derivazione per restrizione da un tipo definito dall'utente specificando, nel tipo stesso, l'attributo `final` con valore `restriction`.

```
<xs:simpleType name="minStr"
final="restriction">
  <xs:restriction base="xs:string">
    …
  </xs:restriction>
</xs:simpleType
```


Tipologie di derivazione

- I facet individuano diverse tipologie di derivazione
- La loro applicabilità dipendono dal tipo base da cui si deriva.
- Tipologie di derivazioni (e di restrizioni):
  - Intervalli numerici e di date (aperti e chiusi)
  - Limitazioni di lunghezza sulle stringhe
  - Rappresentazioni specifiche di tipi numerici
  - Enumerazioni
  - Vincoli sul formato delle stringhe espressi mediante espressioni regolari (pattern)

Facet - Intervalli

- Per definire intervalli numerici si usano le facet:
  - `maxExclusive - minExclusive`
  - `maxInclusive - minInclusive`
- Definiscono estremi di intervalli aperti (exclusive) e chiusi (inclusive)
- Sono applicabili a tutti i valori numerici compresi dateTime, duration, ecc.
- È un errore se `maxExclusive` compare insieme a `maxInclusive` o se `minExclusive` compare insieme a `minInclusive`
- **Vanno in AND** con altri facet sia presenti in una stessa derivazione, sia presenti in derivazioni successive

Esempio di tipo semplice derivato - 1

- Definiamo un tipo semplice derivato dal tipo predefinito `positiveInteger` in modo tale che un elemento o un attributo dichiarato di questo tipo possa assumere valori compresi fra 13 e 19 estremi inclusi.

XSD

```
<xs:simpleType name="teenAgeType">
  <xs:restriction base="xs:positiveInteger">
    <xs:minInclusive value="13"/>
    <xs:maxInclusive value="19"/>
  </xs:restriction>
</xs:simpleType>

<xs:element name="teenAge" type="teenAgeType"/>
```

Facets in AND

Istanza

```
<teenAge>15</teenAge>
```

Facet – Lunghezza delle stringhe

- Per limitare la lunghezza delle stringhe si usano le facet:
  - `length`
  - `maxLength – minLength`
- Definiscono rispettivamente una lunghezza fissa o un intervallo di lunghezze
- Sono applicabili a tutti i valori di tipo stringa e derivati ed anche a hexBinary, base64Binary, ecc.
- È un errore se `length` compare insieme a `minLength` o `maxLength`
- **Vanno in AND** con altri facet sia presenti in una stessa derivazione, sia presenti in derivazioni successive

Esempio di tipo semplice derivato - 2

- Dichiariamo un tipo semplice derivato minMaxStr in modo tale che un elemento di questo tipo possa contenere stringhe di lunghezza variabile fra 7 e 14.
- Operiamo in due passi derivando prima minStr da string e poi minMaxStr da minStr: le facets vanno comunque in AND

```
<xs:simpleType name="minStr">
  <xs:restriction base="xs:string">
     <xs:minLength value="7"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="minMaxStr">
  <xs:restriction base="minStr">
    <xs:maxLength value="14"/>
  </xs:restriction>
</xs:simpleType>
```

Facet – rappresentazione dei tipi numerici

- Possiamo definire il numero di cifre complessive e dei decimali nella rappresentazione dei tipi numerici usando le facet:
  - `totalDigits`
  - `fractionDigits`
- Si applicano a `decimal` e derivati
- Vanno in AND con altri facet
- Esempio: tipo derivato che accetta numeri con al più due cifre decimali (utile per gli importi in Euro)

```
<xs:simpleType name="EuroType">
  <xs:restriction base="xs:decimal">
    <xs:fractionDigits value="2" />
  </xs:restriction>
</xs:simpleType>
```


Facet – Enumerazioni

- `enumeration` consente di definire tipi enumerati
- Applicabile a tutti i tipi predefiniti
- Va in OR con altri enumeration e in AND con altri facet nello stesso step di derivazione, in AND in step diversi

```
<xs:simpleType name="AVType">
 <xs:restriction base="xs:string">
   <xs:enumeration value="VHS" />
   <xs:enumeration value="DVD" />
   <xs:enumeration value="DIVX" />
   <xs:enumeration value="BETAMAX" />
   <xs:enumeration value="MINIDV" />
   <xs:enumeration value="VCD" />
 </xs:restriction>
</xs:simpleType>
```

Formati di audiovisivi

Formati su supporto ottico (sottoinsieme dei formati di audiovisivi)

```
<xs:simpleType name="AVDiscType">
 <xs:restriction base="AVType">
   <xs:enumeration value="DVD" />
   <xs:enumeration value="DIVX" />
   <xs:enumeration value="VCD" />
 </xs:restriction>
</xs:simpleType>
```

Facet - Pattern

- La facet `pattern` consente di restringere i valori ammissibili mediante espressioni regolari (**regular espressions**)
- Applicabile a tutti i tipi predefiniti
- Va in OR con altri pattern e in AND con altri facet presenti in uno stesso step di derivazione, in AND se presente in step diversi
- **Le espressioni regolari sono molto utilizzate e quindi le esaminiamo in dettaglio**

Espressioni regolari - alfabeto

- Un'espressione regolare è una sequenza di caratteri che identifica un insieme di stringhe
- Se viene utilizzata per vincolare uno spazio lessicale solo le stringhe appartenenti all'insieme identificato rappresentano valori validi
- Un'espressione regolare può essere suddivisa in:
  - **caratteri ordinari**: trovano una corrispondenza diretta nelle stringe dell'insieme denotato
  - **metacaratteri**: caratteri speciali che assumono caratteristiche di controllo sui caratteri ordinari
- Lista dei metacaratteri:

`[ ] \ { } | ( ) ^ ? + * .`

Espressioni regolari – Elementi di base

- **Atomo**: un carattere ordinario, un gruppo di caratteri (delimitato da []) o un'espressione regolare fra parentesi tonde
  - Sono atomi:

    `a, [abc], (abc), (a*b+c?) ...`

➔ Una qualunque espressione racchiusa da () diventa un atomo

- **Parte**: un atomo eventualmente seguito da un quantificatore
  - Sono parti:

    `a, a*, a{5,6}, (a*b+), (a*b+)?`

Espressioni regolari - Quantificatori

- Un **quantificatore** (quantifier) vincola l'atomo che lo precede a comparire tante volte quanto indicato (m e n sono valori numerici generici)
  - `{n}`: esattamente n volte
  - `{m,n}`: almeno m e al più n volte
  - `{0,n}`: al più n volte
  - `{m,}`: almeno m volte
- Quindi: `{3,7}` significa da 3 a 7 volte
- Sono previste anche forme abbreviate (sono quelle usate anche dai DTD:
  - `*` equivale a `{0,}`
  - `+` equivale a `{1,}`
  - `?` equivale a `{0,1}`
- **Attenzione:** il quantificatore {1} si può omettere

Espressione banale

- **Espressione banale**: è la forma più semplice (e più banale) di espressione regolare costituita solo da caratteri ordinari (senza metacaratteri)
- Identifica un insieme composto da un solo elemento: la stringa uguale all'espressione stessa
- Esempio: l'espressione `banana` identifica solo la stringa `"banana"`

Esempi di espressioni con quantificatori

- `a*b+c?`
  - Interpretazione: 0 o più occorrenze di a, seguite da 1 o più b seguite da 0 o 1 occorrenza di c.
  - Stringhe ammissibili: `a`, `ab`, `abc`, `bb`, `abb`, `abbc`, `aa`
- `Ar(har){1,2}ha`
  - Interpretazione: "Ar" seguito da 1 o due occorrenze di "har" seguita da "ha"
  - Stringhe ammissibili: `Arharha`, `Arharharha`
- `Ar*gh`
  - Interpretazione: "A" seguita da un numero qualunque di "r" seguita da "gh"
  - Stringhe ammissibili: `Argh`, `Arrrgh`, `Arrrrgh...`

💣 Il quantificatore si applica all'atomo che lo precede:
- Se scriviamo `ab+` il quantificatore agisce solo su 'b'
- Se vogliamo applicarlo ad "ab" scriveremo (ab)+

Gruppi di caratteri - 1

- L'atomo `[elenco-caratteri]` definisce un insieme da cui possono essere scelti tanti caratteri quanti indicati dal quantificatore applicato
- Esempio: `[ciao]{4}`
  - Interpretazione: tutte le stringhe lunghe esattamente 4 composte con i 4 caratteri: c, i, a, o
  - Stringhe ammissibili: `ciao`, `oaic`, `iaoc`, ...
- `[`c_1`-`c_2`]` indica un insieme composto da un intervallo di caratteri, compreso fra c_1 e c_2
- Esempio: `[A-Z]{1,10}` tutte le "parole" da 1 a 10 lettere componibili con le lettere maiuscole
- È possibile concatenare intervalli:
  `[A-Za-z]` = **tutte le lettere dell'alfabeto**

Gruppi di caratteri - 2

- Con ^ è possibile negare un gruppo di caratteri ovvero richiedere che il carattere non sia fra quelli indicati.

  `[^elenco-caratteri]` oppure `[^intervallo]`
- Esempio: `[^A-Z]{5}` = tutte le "parole" di 5 caratteri che non contengono le lettere maiuscole
- Esistono abbreviazioni che rappresentano gruppi di caratteri predefiniti (tra parentesi le equivalenze):
  - `\d`: qualsiasi cifra decimale (`[0-9]`)
  - `\D`: qualsiasi carattere escluse le cifre (`[^0-9]`)
  - `\w`: qualsiasi carattere alfanumerico, compresi '_' e spazio (`[a-zA-Z0-9_ ]`)
  - `\W`: qualsiasi carattere non alfanumerico (`[^a-zA-Z0-9_ ]`)

Altri metacaratteri

- . corrisponde a un carattere qualsiasi
  - Esempio: [.] {10} rappresenta tutte le parole lunghe 10 caratteri
- | separa espressioni regolari in OR (branch)
- Esempi:
  - Se la regola è `abc|def|xyz` vanno bene: `abc`, `def` e `xyz`
  - Se la regola è `abc(def|xyz)` vanno bene: `abcdef` e `abcxyz`
- \ consente di inserire un metacarattere come carattere ordinario (\ si dice carattere di escape)
- Esempi:
  - `\.` indica il carattere punto
  - `\(` indica la parentesi aperta

Esempio: codice fiscale

- Un codice fiscale ha questa struttura:

  RSS MRA 80B21 H501V
- Quindi: 6 lettere maiuscole, 2 cifre, 1 lettera maiuscola, 2 cifre, 1 lettera maiuscola, 3 cifre e 1 lettera maiuscola
- Esprimendo questa regola sotto forma di espressione regolare possiamo definire un pattern di validazione per i codici fiscali:

```
<xs:simpleType name="CodFiscType">
 <xs:restriction base="xs:string">
  <xs:pattern
   value="[A-Z]{6}\d{2}[A-Z]\d{2}[A-Z]\d{3}[A-Z]"/>
 </xs:restriction>
</xs:simpleType>
```

Esempio

- Definiamo un pattern che valida i numeri che terminano per '0' o '5'
- Utile quando c'era la lira e il taglio minimo era 5 lire

```
<xs:simpleType name="LireType">
 <xs:restriction base="xs:positiveInteger">
  <xs:pattern value="\d*[50]"/>
 </xs:restriction>
</xs:simpleType>
```

Tutte le cifre che vogliamo
basta che l'ultima cifra sia 0 o 5

Esempio

- Euro – tipo derivato che accetta numeri con **esattamente** due cifre decimali

```
<xs:simpleType name="StrictEuroType">
 <xs:restriction base="EuroType">
  <xs:pattern value="/d*\./d{2}" />
 </xs:restriction>
</xs:simpleType>
```

Tutte le cifre che vogliamo basta che ci sia
il carattere "." seguito da 2 cifre.
Per inserire "." abbiamo dovuto usare
un carattere di escape \.

Facet: whitespace

- `whitespace` Indica al processore come trattare i caratteri spazio (#x20), tab (#x9), line feed (#xA), carriage return (#xD) nel tipo di dato derivato.
- Può assumere i valori:
  - `preserve`: nessuna operazione
  - `replace`: i caratteri tab, line feed, carriage return vengono sostituiti da spazi
  - `collapse`: viene effettuato il replace, le sequenze di spazi spazio vengono collassate in un unico spazio e gli spazi all'inizio ed alla fine vengono eliminati

```
<xs:simpleType name="myStr">
 <xs:restriction base="xs:string">
  <xs:whiteSpace value="collapse" />
 </xs:restriction>
</xs:simpleType>
<xs:element name="S" type="myStr" />
```

```
<S> C   i   a   o </S>

diventa

<S>C i a o</S>
```


Punti di attenzione

- **Attenzione:** non tutte le combinazioni di facet danno un XSD valido!
- In linea di principio sono legali le combinazioni in cui:
  - Lo spazio dei valori del tipo di dato derivato è più ristretto rispetto a quello del tipo di base
  - Lo spazio dei valori del tipo di dato derivato NON è vuoto
- **Attenzione:** Non tutte le combinazioni "illegali" vengono rifiutate dal processore!!!

Valori di default

- È possibile specificare un valore di default per un elemento con la sintassi:

```
<xs:element name="nome" type="tipo" default="valore" />
```

- Il valore di default viene utilizzato se l'elemento **è presente ed è vuoto**
- Il valore deve essere compatibile col tipo di dato
- La definizione di **vuoto** varia in base al tipo di dato nella dichiarazione dell'elemento
- In tutti i tipi che ammettono come valore valido la stringa vuota il valore di default non è mai utilizzato
- I tipi numerici invece non ammettono un valore vuoto e quindi usano il default

```
<xs:element name="nome" type="xs:integer" default="0"/>
```


Valori fissi (fixed)

- E' possibile assegnare ad un elemento un valore fisso con la sintassi:

```
<xs:element name="nome" type="tipo" fixed="valore" />
```

- In questo caso:
  - **Elemento vuoto** (tenendo conto del significato di vuoto descritto per il default): viene inserito automaticamente il valore fisso
  - **Elemento con valore**: il valore inserito deve corrispondere al valore fisso
- Esempio:

```
<xs:element name="domenica"
    type="xs:integer" fixed="7"/>
```

Attenzione: `default` e `fixed` sono mutuamente esclusivi!!

Valori nulli

- È possibile specificare valori `nil` con significato identico ai valori NULL nell'ambito dei database
- Nella dichiarazione dell elemento occorre specificare la possibilità che assuma il valore nil valorizzando a true il valore dell'attributo `nillable`
- Nel documento istanza si specifica il valore nil valorizzando a true l'attributo `xsi:nil` dove `xsi` è il prefisso di namespace associato all'URL: `http://www.w3.org/2001/XMLSchema-instance`
- XSD:

```
<xs:element name="size" type="xs:integer" nillable="true"/>
```

- Istanza:

```
<size xsi:nil="true"/>
```
Ok

```
<size xsi:nil="true">10</size>
```
Errore!

Nil, fixed e default

- Nella combinazioni di attributi che si ottengono usando insieme `nil` con `fixed` e `default`valgono alcune regole di compatibilità
- Se `nillable="true"` non è possibile specificare un valore `fixed`
- Se `nillable="true"` ed è specificato un valore di `default`:
  - Se `xsi:nil="true"` il valore di default non entra in gioco
  - Se `xsi:nil="false"` o non compare, il valore di default entra in gioco
- **Attenzione**: pur avendo a che fare con simpleType che in generale non supportano attributi, è comuqnue sempre possibile specificare l'attributo `xsi:nil`

Tipi complessi

- Gli elementi dichiarati di tipo complesso possono avere attributi e, in alternativa, elementi figli o contenuto di tipo semplice.
- Abbiamo quindi quattro possibilità:
  - **Contenuto semplice**: solo testo e non elementi figli
  - **Solo elementi** (specifica di content model, oppure derivazione): solo elementi figli e non caratteri
  - **Contenuto mixed**: sia caratteri, sia elementi figli
  - **Nessun contenuto:** gli elementi devono essere vuoti

➔ **Attenzione:** ricordiamo che gli attributi non possono mai essere di tipo complesso, ma solo di tipo semplice.

Tipi con nome e tipi anonimi

- Ricordiamo che è possibile definire:
  - **Tipi con nome**: definiti separatamente e utilizzati successivamente in una o più dichierazioni
  - **Tipi anonimi** (inline): definiti all'interno della dichiarazione di un elemento

```
<xs:complexType name="typeName">
  …tipo di contenuto…
  …attributi…
</xs:complexType>
```

Definizione con nome

Definizione anonima

```
<xs:element name="myElement">
  <xs:complexType>
    …tipo di contenuto…
    …attributi…
  </xs:complexType>
</xs:element>
```

Solo elementi

- Nel caso di tipi che comprendono solo elementi figli la definizione può comprendere tre sezioni:
  - **sequence**: gli elementi dichiarati in questa sezione devono comparire nel documento istanza (XML) nell'**ordine** indicato e **con le cardinalità specificate**
  - **choice**: nel documento istanza deve comparire uno solo degli elementi dichiarati in questa sezione, **con la cardinalità specificata**
  - **all**: tutti gli elementi dichiarati nella sezione all possono comparire **al più una volta** con ordine qualsiasi

Cardinalità

- La cardinalità viene espressa mediante gli attributi `minOccur` e `maxOccur` inseriti all'interno dei vari elementi che compongono il tipo complesso
- `minOccur`: indica il numero **minimo** di volte che l'elemento può comparire:
  - Il valore di default è 1
- `maxOccur`: indica il numero **massimo** di volte che l'elemento può comparire:
  - Il valore di default è 1
  - Per specificare una massima cardinalità pari ad infinito si usa la parola chiave `unbounded`

💣 **Attenzione:** il valore di default **non è zero**, è 1!

Esempio di sequence

```
<xs:complexType name="mySeq">
 <xs:sequence>
  <xs:element name="e1" type="xs:string"
    minOccur="0" maxOccur="unbounded"/>
  <xs:element name="e2" type="xs:string"
     maxOccur="2"/>
 </xs:sequence>
</xs:complexType>

<xs:element name="seq1" type="mySeq"/>
```

Cardinalità: 0..n

Cardinalità: 1..2

XSD

Istanza

```
<seq1>
  <e1>Ciao</e1>
  <e1>Riciao</e1>
  <e2>A tutti</e2>
</seq1>
```


Esempio di choice

```
<xs:complexType name="myCh">
 <xs:choice>
  <xs:element name="e1" type="xs:string"
     minOccur="0" maxOccur="unbounded"/>
  <xs:element name="e2" type="xs:string"
     maxOccur="2"/>
 </xs:choice>
</xs:complexType>

<xs:element name="ch1" type="myCh"/>
```

Cardinalità: 0..n

Cardinalità: 1..2

XSD

Istanza

```
<ch1>
  <e2>Ecco qua</e2>
</ch1>
```

Cardinalità di gruppo

- I gruppi `sequence` e `choice` possono a loro volta avere una cardinalità
- Si usano sempre gli attributi `minOccur` e `maxOccur`
- Nell'esempio seguente la sequenza deve essere ripetuta da 2 a 3 volte e ogni ripetizione deve contenere esattamente una volta l'elemento e1 e una volta l'elemento e2 (cardinalità di default = 1)

```
<xs:complexType name="typeName">
 <xs:sequence minOccur="2" maxOccur="3">
  <xs:element name="e1" type="xs:string"/>
  <xs:element name="e2" type="xs:string"/>
 </xs:sequence>
</xs:complexType>
```


Combinazione di sequence e choice

- I gruppi `sequence` e `choice` possono essere innestati:

```
<xs:complexType name="typeName">
 <xs:sequence>
   <xs:choice>
    <xs:element name="a" type="xs:string"/>
    <xs:element name="b" type="xs:string"/>
   <xs:choice>
   <xs:choice>
    <xs:element name="c" type="xs:string"/>
    <xs:element name="d" type="xs:string"/>
   <xs:choice>
 </xs:sequence>
</xs:complexType>
```

All

- Consente di indicare che tutti gli elementi conformi a quelli dichiarati al suo interno possono comparire
  - in qualsiasi ordine
  - al più una volta
- Può contenere solo dichiarazioni di elementi
- Non può comparire all'interno di altri gruppi (es: `sequence`, `choice`)
- Non è possibile specificare cardinalità con `minOccur` e `maxOccur` a livello di gruppo
- I valori validi di `minOccur` e `maxOccur` negli elementi contenuti nel gruppo sono rispettivamente (0,1) e 1

Esempio di all

```
<xs:complexType name="myAll">
 <xs:all>
  <xs:element name="e1" type="xs:string"/>
  <xs:element name="e2" type="xs:string"/>
  <xs:element name="e3" type="xs:string"
     minOccur="0" maxOccur="1"/>
 </xs:all>
</xs:complexType>
<xs:element name="all1" type="myAll"/>
```

Cardinalità 1 (default)

Cardinalità: 0..1

XSD

Istanza

```
<all1>
  <e2>A tutti</e2>
  <e1>Ciao</e1>
</all1>
```

e1 ed e2 devono per forza comparire (in qualsiasi ordine, mentre e3 può non comparire

Content model mixed

- Ha lo stesso significato che ha in DTD: consente la presenza di caratteri e di elementi
- Ha senso parlare di contenuto mixed solo per tipi complessi
- Per avere un modello mixed è sufficiente indicare nella definizione del tipo complesso l'attributo `mixed` e attribuirgli il valore `"true"`
- La specifica di contenuto mixed non è alternativa alla specifica di un modello di contenuto come in DTD
  - il modello di contenuto **deve** essere rispettato
  - il modello mixed di DTD e quello di XML Schema **non si equivalgono**

Esempio di mixed

```
<xs:complexType name="LetterType" mixed="true">
  <xs:sequence>
    <xs:element name="nome" type="xs:string"/>
    <xs:element name="cognome" type="xs:string"/>
    <xs:element name="prodotto" type="xs:string"/>
    <xs:element name="taglia" type="xs:positiveInteger"/>
  </xs:sequence>
</xs:complexType>
<xs:element name="letter" type="LetterType"/>
</xs:schema>
```

Schema

Istanza

```
<letter>
  Sono <nome>Mario</nome>
  <cognome>Rossi</cognome>
  e compro un <prodotto>maglione</prodotto>
  taglia <taglia>50</taglia>
</letter>
```

La sequenza degli elementi deve essere rispettata!!

Content model empty

- Per avere un content model empty è sufficiente definire un complexType privo di contenuto
- Gli elementi di questo tipo devono essere vuoti

```
<xs:complexType name="myEmpty">
</xs:complexType
```


Attributi

- Gli attributi possono essere contenuti solo da elementi di tipo complexType
- Devono essere dichiarati dopo il modello di contenuto
- Si usa la sintassi:

```
<xs:attribute name="attributeName"
  type="attributeSimpleType"
  use="optional | prohibited | required"/>
```

- Dove:
  - `name`: nome dell'attributo
  - `type`: tipo dell'attributo (**solo simpleType**)
  - `use`:
    - `optional`: l'attributo può non comparire
    - `prohibited`: l'attributo non deve comparire
    - `required`: l'attributo deve comparire

Esempio

```
<xs:complexType name="WAttrType">
 <xs:sequence>
   <xs:element name="a" type="xs:string"/>
   <xs:element name="b" type="xs:string"/>
 </xs:sequence>
 <xs:attribute name="at" type="xs:string"/>
</xs:choice>
```


Dichiarazione di attributi con definizione del tipo inline

- Gli attributi possono anche essere di un tipo semplice derivato
- Può essere un tipo definito in precedenza oppure si può ricorrere anche in questo caso a una definizione inline con questa sintassi:

```
<xs:attribute name="attributeName"

  use="optional|prohibited|required">
  <xs:simpleType>
  …
  </xs:simpleType>
</xs:attribute>
```

Default e fixed

- E possibile definire valori di default o fissi per un attributo usando la sintassi:

```
<xs:attribute name="attrName"
  type="attrType" default="value"/>
<xs:attribute name="attrName"
  type="attrType" fixed="value"/>
```

- La logica è uguale a quella delle dichiarazioni DTD:
  - **Default**: se l'attributo non è presente, viene inserito il valore di default, altrimenti il valore di default non entra in gioco
  - **Fixed**: se l'attributo non è presente, viene inserito il valore fixed, altrimenti il valore nel documento istanza deve essere uguale al valore fixed

`fixed` e `default` sono mutuamente esclusivi!

Elementi a contenuto semplice e attributi

- Gli attributi possono essere dichiarati solo su elementi complessi
- E' però possibile derivare un tipo complesso da un tipo semplice ed estenderlo aggiungendo attributi
- Per far ciò si utilizza il content model `simpleContent` con questa sintassi:

```
<xs:complexType name="typeName">
<xs:simpleContent>
  <xs:extension base="baseType">
    <xs:attribute name="attName" type="attType"/>
  </xs:extension>
 </xs:simpleContent>
</xs:complexType>
```

Esempio di simpleContent con estensione

- Dichiariamo un tipo “taglia” utile per gestire le taglie dei vestiti, il cui valore dipende dalla nazione
- Estendiamo il tipo semplice **`integer`** come tipo complesso in modo da poter aggiungere l’attributo **`nazione`**

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

Definizione inline

Schema

Istanza

```
<taglia nazione="Italia">48</taglia>
```


Dichiarazioni globali e riferimenti

- È possibile effettuare dichiarazioni globali di elementi ed attributi e referenziare tali dichiarazioni per effettuare altre dichiarazioni.
- Le dichiarazioni globali compaiono al primo livello dello schema, come figli diretti dell’elemento **`schema`**
- La sintassi per i riferimenti a dichiarazioni globali è:

  `<xs:element ref="aGlobalElement"/>`
- L’elemento o attributo dichiarato per riferimento ha le stesse proprietà (nome, tipo, ecc.) dell’elemento o attributo riferito
- Tali proprietà **non** possono essere ridefinite

Limitazioni delle dichiarazioni globali

- In una dichiarazione globale NON è possibile:
  - Utilizzare un riferimento per la dichiarazione: occorre effettuare dichiarazioni che utilizzino un tipo predefinito o definito dall'utente oppure utilizzare una definizione inline
  - Esprimere vincoli di cardinalità negli elementi: `minOccurs` e `maxOccurs` non possono comparire
  - Esprimere vincoli di uso negli attributi: la proprietà `use` non può comparire
- Eventuali vincoli di cardinalità per gli elementi o di uso per gli attributi vanno eventualmente specificati nelle dichiarazioni di elemento/attributo che si riferiscono (usano l'attributo ref) all'elemento/attributo globale per la loro dichiarazione.

Esempio di dichiarazioni globali e riferimenti

```
<xs:schema …>
  <xs:element name="comment" type="xs:string"/>
  <xs:attribute name="att" type="xs:integer"/>
  <xs:element name="myRoot">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="a" type="xs:float"/>
        <xs:element ref="comment"/>
      </xs:sequence>
      <xs:attribute ref="att"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Dichiarazioni globali

Riferimenti

Ok in base al riferimento

```
<myRoot att="5">
  <a/>
  <comment>Sono nato stanco!</comment>
</myRoot>
```

Namespace

- Uno schema è un insieme di definizioni e dichiarazioni i cui nomi appartengono ad un particolare namespace detto **targetNamespace (namespace obiettivo)**
  - Ogni schema può avere un solo targetNamespace
  - L'attributo `targetNamespace` va posto nell'elemento `schema`
- Usando i namespace è possibile distinguere i nomi del vocabolario XML Schema dai nomi del vocabolario che si sta definendo
- In un documento XMLSchema che vuole "definire" un namespace dovremo dichiarare due namespace:
  - Il **namespace degli schemi**: http://www.w3.org/2001/XMLSchema
  - Il **namespace obiettivo**

Esempio di namespace

```
<schema
  xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:po="http://example.com/PO1"
  targetNamespace="http://example.com/PO1"
  ... >
...
</schema>
```

- Tutte le definizioni e le dichiarazioni all'interno del documento faranno parte del namespace obiettivo
- Gli identificatori presi dal namespace degli schemi saranno senza prefisso (lo abbiamo definito come namespace di default)

Qualificazione -1

- La specifica XML Schema impone che tutti gli elementi globali (di primo livello) definiti in uno schema devono essere qualificati con un prefisso di namespace
- Quest'obbligo viene meno se nel documento istanza il namespace dello schema è dichiarato come namespace di default
- Esiste invece la possibilità di definire una politica di uso dei presfissi per quanto riguarda le **dichiarazioni locali** ovvero:
  - gli elementi interni ai tipi complessi **esclusi quelli definiti per riferimento**
  - gli attributi

Qualificazione -2

- Per far ciò si usano due attributi
- `elementFormDefault="unqualified|qualified"`
  `attributeFormDefault="unqualified|qualified"`
- Per default il loro valore è `unqualified`
- Vengono di soliti impostati nell'elemento `schema` e quindi valgono per l'intero schema
- Possono però essere impostati anche negli elementi più interni consentendo così la definizione di politiche più flessibili