XML QUICK START

1. Introduzione

Il linguaggio XML fornisce una serie di regole per definire e memorizzare strutture di dati, ma non come visualizzarli. E' *case sensitive*, richiede la chiusura di ogni tag e proibisce l'annidamento misto (i.e. le tag vanno chiuse nell'ordine opposto a quello di apertura). E' obbligatorio racchiudere i valori degli attributi tra apici (singoli o doppi) ed usare le *reference entity* (< > & ' "). L'unica tag che non deve essere chiusa è quella che definisce versione e codifica XML, collocata all'inizio del documento, ad esempio:

```
<?xml version="1.0" encoding="utf-8" ?>
```

2. Concetti base

Le tag XML (o *elementi* XML) possono contenere attributi. E' considerata una buona pratica usare gli attributi solamente per i meta-data: i dati veri e propri andrebbero sempre inseriti come elementi XML. Un documento XML che rispetta le regole discusse nell'introduzione è detto ben formato. Un documento XML che rispetta le specifiche di un DTD (Document Type Definition) si dice validato. Le specifiche del DTD vanno indicate subito dopo la tag di apertura del documento, ad esempio:

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE myTag SYSTEM "MyTagDefinition.dtd">
```

Al posto delle specifiche DTD è possibile usare un documento **XML Schema**, che fornisce le stesse informazioni utilizzando il formato XML. Ad esempio:

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema xmlns:xs="http://aaa" targetNamespace="http://bbb"
      xmlns="http://ccc" elementFormDefault="qualified">
<myTag>…</myTag>
</xs:schema>
```

Per visualizzare un documento XML in modo user-friendly è possibile definire un foglio di stile

```
<?xml version="1.0" encoding="utf-8" ?>
<?xml-stylesheet type="text/css" href="style_01.css"?>
```

usando come identificatori gli stessi identificatori delle tag XML: ad esempio, lo stile della tag `<test>foo</test>` può essere definito dal blocco `test {color:red; font-size:16pt;}` all'interno del file `style_01.css`. E' comunque preferibile usare il linguaggio **XSLT** che permette di specificare le regole per trasformare un qualsiasi documento XML direttamente in HTML. Questo compito può essere svolto dal browser mediante la seguente dichiarazione:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="my_style.xsl"?>
```

In alternativa la trasformazione XSLT può essere svolta lato server, in tal caso al browser viene spedito il documento HTML risultante (elaborato da una pagina dinamica).

Nota: il contenuto di ogni tag XML viene esaminato perché potrebbe contenere altre tag: perciò i caratteri all'interno di un elemento XML sono dei PCDATA (Parsed Character Data). Eventuali caratteri non parsabili vanno definiti come CDATA (Character Data), ovvero vano inclusi all'interno come indicato qui sotto:

```
<![CDATA[ … … … ]]>
```


3. XMLHttpRequest

L'oggetto JavaScript `XMLHttpRequest` (presente ormai in tutti i browers) consente di richiedere un file XML specificando un generico URL, dove una pagina dinamica o un Web Service si occupa di rispondere alterando lo stato dell'oggetto `XMLHttpRequest` stesso. Mediante questo meccanismo il client può accedere all'attributo `responseText` dell'oggetto `XMLHttpRequest` , che solitamente contiene il file XML richiesto.
Tramite JavaScript è poi possibile effettuare il parsing del documento XML e alterare dinamicamente il documento HTML caricato nel brower, senza che avvenga la richiesta di un'intera pagina via HTTP da parte del browser (è l'oggetto `XMLHttpRequest` ad effettuare la chiamata HTTP, non il browser).
Tale meccanismo è alla base della tecnologia *Ajax*.

4. Namespaces

Per risolvere le ambiguità tra elementi XML con lo stesso nome è possibile dichiarare il *namespace* di riferimento utilizzato all'interno di un certo elemento, specificando l'attributo **xmlns** (in realtà *xmlns* non è un attributo, perché non compare come tale nel DOM, ma si specifica come se fosse un attributo). Ad esempio:

```
<myTag xmlns:sa="http://aaaa">
```

In tal modo tutte le tag usate all'interno di `myTag` e aventi come prefisso `sa` faranno riferimento al namespace precisato (esempio: `<sa:myChild>foo</sa:myChild>`). E' possibile definire più namespaces (e relativi prefissi) all'interno di una stessa tag:

```
<rootTag xmlns:p1="http://aaa" xmlns:p2="http://bbb">
```

Inoltre può essere conveniente specificare un **default namespace**, che verrà utilizzato per tutte gli elementi sprovvisti di prefisso. Il namespace di default va definito senza indicare il prefisso:

```
<rootTag xmlns="http://aaa">
```

Notare infine che l'indicazione del namespace è già attiva anche sulla tag corrente. Ad esempio

```
<xsl:stylesheet  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

definisce il prefisso **xsl** che viene usato per identificare la tag stessa: perciò il nome locale `stylesheet` è riferito proprio al namespace definito come "http://www.w3.org/1999/XSL/Transform".

Alcune note:

- I nomi delle tag XML espresse tramite prefisso sono detti **Qualified Name** (`Qname`). Perciò `<sa:myTag>` è un elemento *qualified*, mentre `<myTag>` è un elemento *unqualified*.

- L'URL utilizzato per definire il namespace (esempio: *http://www.w3.org/2001/XMLSchema*) è solamente una stringa, in generale può non esistere alcun dominio registrato a quell'indirizzo. In altre parole la stringa tra virgolette è un identificatore astratto che per convenzione deve rispettare la sintassi degli URL. Ciò significa che i prefissi *aa* e *bb* definiti come *xmlns:aa="http://foo"* e *xmlns:bb="http://foo"* indicano lo stesso identico namespace.

- La scelta di esprimere i namespace usando la sintassi degli URL permette di registrare il dominio corrispondente al namespace per pubblicare l'eventuale documentazione del namespace, ma ciò non è affatto obbligatorio.

5. Note

- Per annullare la dichiarazione del namespace di default si scrive:

```
<rootTag xmlns="">
```

  Non è possibile usare questa sintassi per "annullare" la dichiarazione dei prefissi, quindi è sbagliato scrivere `<rootTag xmlns:sa="">` per tentare di annullare la dichiarazione del prefisso `sa`.

- Il namespace di default non si vale per gli attributi, ad esempio

```
<rootTag  xmlns=" http://foo"  xmlns:sa=" http://temp">
  <myTag sa:mode="2">
  <myTag type="1">
</rootTag>
```

  in questo caso l'elemento `myTag`, essendo senza prefisso, è associato al namespace di default (`http://foo`). L'attributo `mode` è correttamente associato al namespace `http://temp`, mentre l'attributo `type` è sconosciuto. Questo comportamento è cruciale quando si devono definire degli XML Schema (vedasi il documento *XSD_quick_start*).

- Evitare di usare la tag `<xml src="…">` (usata solamente da Internet Explorer).

- Evitare di usare l'attributo `behavior` nelle dichiarazioni di stile (idem come sopra).

Appendice: XPath

XPath è il linguaggio utilizzato per identificare un qualsiasi nodo, elemento o attributo di un documento XML. In questa trattazione vengono forniti solamente esempi riferiti a *nomi locali*: se vengono usati dei prefissi la sintassi cambia leggermente (esempio: `/sa:myTag/sa:myNode`).

La sintassi ricorda le regole utilizzate per identificare file e cartelle in un file system, ad esempio `myTag` indica tutti i figli dell'elemento `myTag`, mentre `/myNode/myTag` indica un percorso assoluto. La sintassi `//myTag` identifica tutti gli elementi `myTag` ovunque si trovino nell'albero, mentre `@myField` indica l'attributo `myField` di una tag (è possibile usare "`*`"come wildcard e " |" come operatore OR).

I predicati XPath utilizzano la sintassi degli array, ad esempio:

- `/myNode/myTag[1]`: primo elemento `myTag` figlio di `myNode`.
- `/myNode/myTag[last()]`: ultimo elemento `myTag` figlio di `myNode`.
- `/myNode/myTag[position()<3]`: primi due elementi `myTag` figli di `myNode`.
- `//myTag[@lang]`: tutti gli elementi `myTag` aventi un attributo `lang`.
- `/one[value > 42]/two`: tutti gli elementi `two` figli delle tag `one` con l'attributo `value` > 42.

E' possibile selezionare un sottogruppo (detto *axes*) del nodo corrente utilizzando la sintassi `<axes>::<target>`, ad esempio:

- `child::node()` : tutti i figli del nodo corrente.
- `child::myTag` : tutte le tag `myTag` figlie del nodo corrente.
- `attribute::lang` : tutti gli attributi `lang` del nodo corrente.
- `ancestor::myTag` : tutte le tag `myTag` antenate del nodo corrente.
- `child::*/child:: myTag` : tutte le tag `myTag` nipoti del nodo corrente.

