

# Keretrendszer alapú programozás

Fazekas Csaba

[fazekas.csaba@uni-eszterhazy.hu](mailto:fazekas.csaba@uni-eszterhazy.hu)

**XML**

# XML története

- Standard Generalized Markup Language (SGML) egyszerűbb változata az XML.

“Az SGML-t eredetileg arra tervezték, hogy lehetővé tegyék a nagy kormányzati és ipari projektek dokumentumainak közzétételét számítógép által is beolvasható formában, azzal a megkötéssel, hogy a dokumentumoknak több évtizedig is olvashatóknak kell lenniük, ami az informatikában nagyon hosszú időnek számít. A formátumnak nagy hasznát vették továbbá a nyomdatechnikai iparágak, de a bonyolultsága megakadályozta, hogy széles körben elterjedjen kisebb méretű általános alkalmazásokban.

Elsősorban szöveges alapú adatbázisok kezeléséhez és közzétételéhez tervezték, és első nagyobb alkalmazásainak egyike az **Oxford English Dictionary** (OED) második kiadása volt, ami teljes egészében SGML jelölésrendszert használt. Ma leggyakrabban **HTML** dokumentum formájában találkozunk az SGML-lel, tekintve, hogy a **W3C** (World Wide Web Consortium) támogatásával a HTML az internetes dokumentumok egyik fő szabványává vált.” (Wikipedia)

# XML története

- “A GML-t 1960-as években fejlesztette ki az IBM-nél Charles Goldfarb, Edward Mosher és Raymond Lorie (családnevük kezdőbetűi alapján találta ki Goldfarb a GML nevet).” (Wikipedia)
- Egy GML jelölései (tag-ek) jelölik a fejezeteket, paragrafusokat kiemelik a fontosabb részeket, vagy elnyomják a kevésbé érdekeseket, továbbá listákat, táblázatokat jelenítenek meg.
- A dokumentum egyéb részéire, mint a fontok, sorköz, az oldal elhelyezése, nem fektet hangsúlyt.
- Nyomtathatóvá az IBM számára a Document Composition Facility (DCF) alkalmazás tette.

# XML

- A GML-re alapozva jött létre a Standard GML 1986-ban.
- SGML legfőbb célja a dokumentumok megosztása volt számítógépek között.
- SGML-en alapul például a HTML.
- 1998-ban az SGML-re alapozva létrejött az XML.
- A World Wide Web Consortium fejlesztette.

# XML

- Az 'XML' név egy rövidítés ami a 'E**x**ensible **M**arkup **L**anguage'-ből ered (esztétikai okokból 'X'-et használnak az 'E' helyett).
- Az XML mindig egy megnevezett karakter kódolással készül, mint az ASCII, vagy az **UTF-8**.

# XML

- Hasonlít a HTML-re de a célja **strukturált adatok** leírása,
- sokkal inkább mint adatok megjelenítése egy web browser-rel.
- Megőrizte az SGML-ből azt a célt, hogy az adatok megosztását könnyítse.

# XML

- **Hierarchikus** felépítésű.
- **Fa struktúrát alkot**, amiben elemek más elemeket tartalmazhatnak.



# Példa

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

```
<HighScore>
```

```
<AngryBirds>
```

```
<first>Bill Gates</first>
```

```
<second>Paul Alen</second>
```

```
<third>Csaba Fazekas</third>
```

```
</AngryBirds>
```

```
<FlappyBird>
```

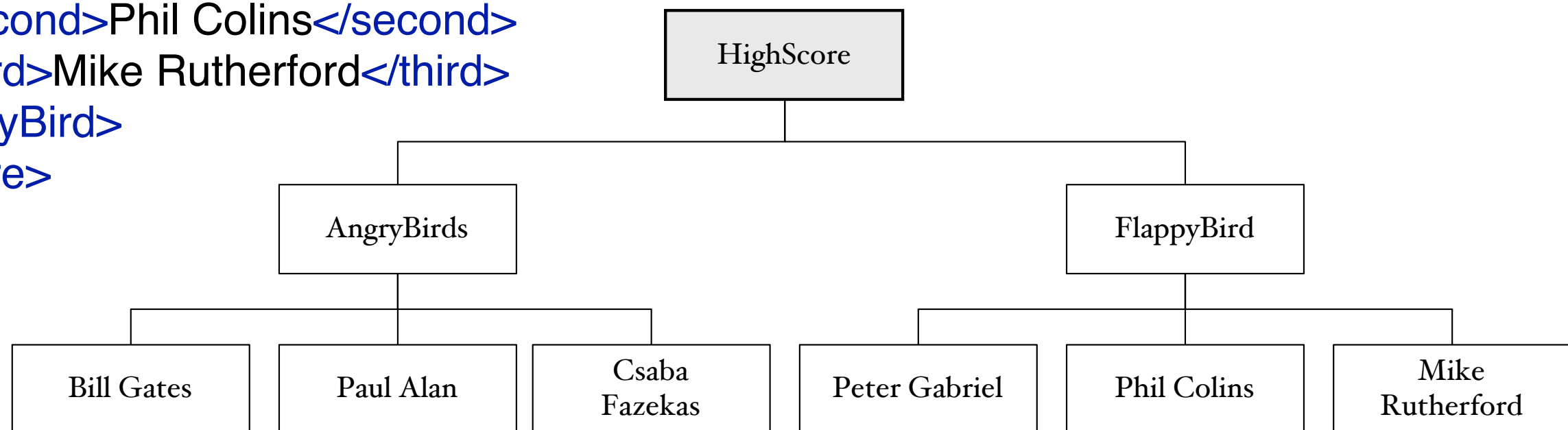
```
<first>Peter Gabriel</first>
```

```
<second>Phil Colins</second>
```

```
<third>Mike Rutherford</third>
```

```
</FlappyBird>
```

```
</HighScore>
```



# Mire jó az XML?

- XML-nek számos felhasználási területe van.
- Két kiemelt terület:
  - **adatok cseréje,**
  - **dokumentumok reprezentálása.**
- De programozásban kiemelt fontossága van
  - **adatbázis reprezentálására,**
  - **alkalmazások konfigurálására,**
  - stb.

# Miért kell?

00001780:	DC	B8	20	6E	A7	14	E7	97	26	0D	0D	10	1D	2D	41	C3	нз¶чЧ&♪♪▶↔-A└
00001790:	DE	DF	A8	17	FC	DF	39	1D	A6	A6	19	AF	06	B8	9C	61	Ги‡№■9↔жж↓п♠¶ба
000017A0:	83	88	99	51	97	D5	95	8D	66	63	A7	52	41	E7	33	5D	ГИЩQЧ└XHfсзRAчЗ]
000017B0:	CF	7A	FB	E4	B8	D9	6A	62	95	EB	75	62	3C	C0	A3	FE	└z√φ¶└j bХыub<└г■
000017C0:	19	90	06	5A	C2	3E	A7	0E	3B	56	8C	BA	C6	0D	CC	3E	↓P♠Z└└>з♪; VM  └└└>
000017D0:	87	AD	24	A6	CE	E4	FC	32	D5	8B	AE	47	C0	A4	40	DE	Зн\$ж└└φ№2└ЛоG└д@■
000017E0:	45	99	9D	F7	3C	DF	65	C5	22	B1	9D	F9	57	4B	C9	D8	ЕЩЭў<■e└"██Э•WK└└
000017F0:	F7	95	AE	A6	1A	ED	F1	35	26	36	F3	03	72	0E	FF	6B	ўХож→эё5&6є♥r♪ k
00001800:	33	EC	A0	61	33	B4	C8	A8	24	B1	86	15	9D	32	1B	73	Зьаа3└└Ли\$██Ж\$Э2←s
00001810:	4B	E4	62	44	DB	14	FE	92	5D	3D	30	1A	17	B0	63	C6	КφbD■¶■T]=0→‡██с└
00001820:	02	96	6F	53	BF	44	9A	FF	FC	27	76	73	8A	33	E8	E1	☉ЦoS└Dб №'vsKЗшс
00001830:	21	27	FE	10	7C	F8	12	79	BB	48	CD	DE	57	EF	A0	AB	!'■▶└└°‡y¶H=└Wяал
00001840:	A7	39	54	B0	42	29	E2	61	40	5D	BF	0B	58	22	B9	61	з9T██B)та@└└♂X"└└a
00001850:	0A	32	A3	25	2E	4C	91	77	F0	FE	6E	C1	91	0C	08	55	○2г%.LCwЁ■n└C♀■U
00001860:	9D	1E	D7	20	A6	91	69	05	B4	9C	32	6E	35	F6	F7	5B	Э▲└└жCi♣└└б2п5ўў[
00001870:	DB	B3	32	B1	BD	5F	6D	B4	4E	9A	8F	6A	19	30	1B	3A	■└2██└└_m└└NьПj↓0←:
00001880:	AD	BC	65	00	F2	73	0D	6F	0A	91	50	A2	93	6C	B8	FA	н└└e└└єs└└o○CPBУ└└.
00001890:	CD	E9	22	2E	5E	B4	C0	87	CC	BC	89	A7	AA	A0	3F	43	=щ".^└└└3└└└Йзка?C
000018A0:	54	DE	E8	A6	00	AF	A8	10	6C	AA	AF	D1	23	21	B6	8A	T└└шж└└пи▶└└кп└└#└└!└└K
000018B0:	6E	FD	B1	BF	61	8E	5F	8D	35	55	F2	0A	48	67	7C	95	п⌘██└└a0_H5UE○Hg└└X

# Miért kell?

- Régebben országonként egyedi karakter kódolás volt.
- Az ASCII (7 bites) valósított meg először standard karakter kódolást, amiben 128 karakter már mindig ugyanaz volt. Az ASCII 8 bites változatában a felső 128 karakter egyedi lehetett.
- Az UTF-8 és UNICODE megjelenése sokat segített.

# ASCII (also 128)

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	<b>NUL</b> (null)	32	20	040	&#32;	<b>Space</b>	64	40	100	&#64;	<b>@</b>	96	60	140	&#96;	<b>`</b>
1	1	001	<b>SOH</b> (start of heading)	33	21	041	&#33;	<b>!</b>	65	41	101	&#65;	<b>A</b>	97	61	141	&#97;	<b>a</b>
2	2	002	<b>STX</b> (start of text)	34	22	042	&#34;	<b>"</b>	66	42	102	&#66;	<b>B</b>	98	62	142	&#98;	<b>b</b>
3	3	003	<b>ETX</b> (end of text)	35	23	043	&#35;	<b>#</b>	67	43	103	&#67;	<b>C</b>	99	63	143	&#99;	<b>c</b>
4	4	004	<b>EOT</b> (end of transmission)	36	24	044	&#36;	<b>\$</b>	68	44	104	&#68;	<b>D</b>	100	64	144	&#100;	<b>d</b>
5	5	005	<b>ENQ</b> (enquiry)	37	25	045	&#37;	<b>%</b>	69	45	105	&#69;	<b>E</b>	101	65	145	&#101;	<b>e</b>
6	6	006	<b>ACK</b> (acknowledge)	38	26	046	&#38;	<b>&amp;</b>	70	46	106	&#70;	<b>F</b>	102	66	146	&#102;	<b>f</b>
7	7	007	<b>BEL</b> (bell)	39	27	047	&#39;	<b>'</b>	71	47	107	&#71;	<b>G</b>	103	67	147	&#103;	<b>g</b>
8	8	010	<b>BS</b> (backspace)	40	28	050	&#40;	<b>(</b>	72	48	110	&#72;	<b>H</b>	104	68	150	&#104;	<b>h</b>
9	9	011	<b>TAB</b> (horizontal tab)	41	29	051	&#41;	<b>)</b>	73	49	111	&#73;	<b>I</b>	105	69	151	&#105;	<b>i</b>
10	A	012	<b>LF</b> (NL line feed, new line)	42	2A	052	&#42;	<b>*</b>	74	4A	112	&#74;	<b>J</b>	106	6A	152	&#106;	<b>j</b>
11	B	013	<b>VT</b> (vertical tab)	43	2B	053	&#43;	<b>+</b>	75	4B	113	&#75;	<b>K</b>	107	6B	153	&#107;	<b>k</b>
12	C	014	<b>FF</b> (NP form feed, new page)	44	2C	054	&#44;	<b>,</b>	76	4C	114	&#76;	<b>L</b>	108	6C	154	&#108;	<b>l</b>
13	D	015	<b>CR</b> (carriage return)	45	2D	055	&#45;	<b>-</b>	77	4D	115	&#77;	<b>M</b>	109	6D	155	&#109;	<b>m</b>
14	E	016	<b>SO</b> (shift out)	46	2E	056	&#46;	<b>.</b>	78	4E	116	&#78;	<b>N</b>	110	6E	156	&#110;	<b>n</b>
15	F	017	<b>SI</b> (shift in)	47	2F	057	&#47;	<b>/</b>	79	4F	117	&#79;	<b>O</b>	111	6F	157	&#111;	<b>o</b>
16	10	020	<b>DLE</b> (data link escape)	48	30	060	&#48;	<b>0</b>	80	50	120	&#80;	<b>P</b>	112	70	160	&#112;	<b>p</b>
17	11	021	<b>DC1</b> (device control 1)	49	31	061	&#49;	<b>1</b>	81	51	121	&#81;	<b>Q</b>	113	71	161	&#113;	<b>q</b>
18	12	022	<b>DC2</b> (device control 2)	50	32	062	&#50;	<b>2</b>	82	52	122	&#82;	<b>R</b>	114	72	162	&#114;	<b>r</b>
19	13	023	<b>DC3</b> (device control 3)	51	33	063	&#51;	<b>3</b>	83	53	123	&#83;	<b>S</b>	115	73	163	&#115;	<b>s</b>
20	14	024	<b>DC4</b> (device control 4)	52	34	064	&#52;	<b>4</b>	84	54	124	&#84;	<b>T</b>	116	74	164	&#116;	<b>t</b>
21	15	025	<b>NAK</b> (negative acknowledge)	53	35	065	&#53;	<b>5</b>	85	55	125	&#85;	<b>U</b>	117	75	165	&#117;	<b>u</b>
22	16	026	<b>SYN</b> (synchronous idle)	54	36	066	&#54;	<b>6</b>	86	56	126	&#86;	<b>V</b>	118	76	166	&#118;	<b>v</b>
23	17	027	<b>ETB</b> (end of trans. block)	55	37	067	&#55;	<b>7</b>	87	57	127	&#87;	<b>W</b>	119	77	167	&#119;	<b>w</b>
24	18	030	<b>CAN</b> (cancel)	56	38	070	&#56;	<b>8</b>	88	58	130	&#88;	<b>X</b>	120	78	170	&#120;	<b>x</b>
25	19	031	<b>EM</b> (end of medium)	57	39	071	&#57;	<b>9</b>	89	59	131	&#89;	<b>Y</b>	121	79	171	&#121;	<b>y</b>
26	1A	032	<b>SUB</b> (substitute)	58	3A	072	&#58;	<b>:</b>	90	5A	132	&#90;	<b>Z</b>	122	7A	172	&#122;	<b>z</b>
27	1B	033	<b>ESC</b> (escape)	59	3B	073	&#59;	<b>;</b>	91	5B	133	&#91;	<b>[</b>	123	7B	173	&#123;	<b>{</b>
28	1C	034	<b>FS</b> (file separator)	60	3C	074	&#60;	<b>&lt;</b>	92	5C	134	&#92;	<b>\</b>	124	7C	174	&#124;	<b> </b>
29	1D	035	<b>GS</b> (group separator)	61	3D	075	&#61;	<b>=</b>	93	5D	135	&#93;	<b>]</b>	125	7D	175	&#125;	<b>}</b>
30	1E	036	<b>RS</b> (record separator)	62	3E	076	&#62;	<b>&gt;</b>	94	5E	136	&#94;	<b>^</b>	126	7E	176	&#126;	<b>~</b>
31	1F	037	<b>US</b> (unit separator)	63	3F	077	&#63;	<b>?</b>	95	5F	137	&#95;	<b>_</b>	127	7F	177	&#127;	<b>DEL</b>

# ASCII (felső 128)

128	Ç	144	É	160	á	176	░	192	Ł	208	⌌	224	α	240	≡
129	ü	145	æ	161	í	177	▒	193	⌐	209	⌑	225	β	241	±
130	é	146	Æ	162	ó	178	▓	194	⌒	210	⌒	226	Γ	242	≥
131	â	147	ô	163	ú	179		195	⌓	211	⌌	227	π	243	≤
132	ä	148	ö	164	ñ	180	⌔	196	—	212	⌍	228	Σ	244	∫
133	à	149	ò	165	Ñ	181	⌕	197	⌕	213	⌎	229	σ	245	∫
134	â	150	û	166	²	182	⌖	198	⌖	214	⌏	230	μ	246	÷
135	ç	151	ù	167	°	183	⌗	199	⌗	215	⌐	231	τ	247	≈
136	ê	152	ÿ	168	¿	184	⌘	200	⌘	216	⌑	232	Φ	248	°
137	ë	153	Ö	169	⌁	185	⌙	201	⌙	217	⌒	233	⊕	249	.
138	è	154	Ü	170	⌂	186	⌚	202	⌚	218	⌓	234	Ω	250	.
139	ï	155	¢	171	½	187	⌛	203	⌛	219	■	235	δ	251	√
140	î	156	£	172	¼	188	⌜	204	⌜	220	■	236	∞	252	π
141	ì	157	¥	173	¡	189	⌝	205	=	221	■	237	φ	253	²
142	Ä	158	ℳ	174	«	190	⌞	206	⌞	222	■	238	ε	254	■
143	Å	159	ℱ	175	»	191	⌟	207	⌟	223	■	239	∩	255	

Source: [www.LookupTables.com](http://www.LookupTables.com)



# Mire lehetett használni a felső 128 karaktert?

```
DOSBox Shell v0.65
DOSBox runs real and protected mode games.
For supported shell commands type: HELP
For a short introduction type: INTRO

If you want more speed, try ctrl-F8 and ctrl-F12.
To activate the keymapper ctrl-F1.
For more information read the README file in the DOSBox direct

HAVE FUN!
The DOSBox Team
```

```
Z:\>SET BLASTER=A220 I7 D1 H5 T6
```

```
Z:\>SET ULTRASND=240,3,3,5,5
```

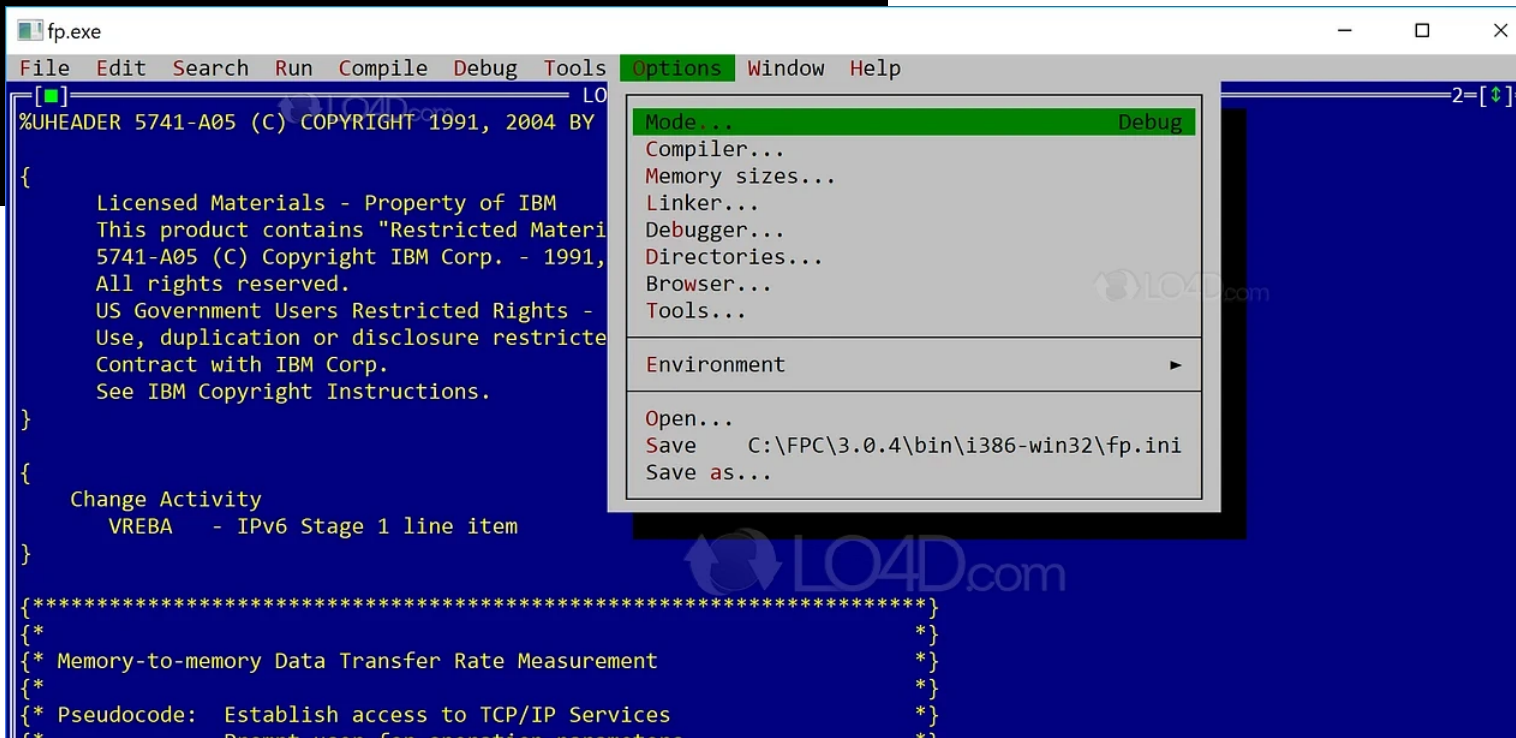
```
Z:\>SET ULTRADIR=C:\ULTRASND
```

```
Z:\>_
```

Left				Right			
File				Command			
Options				Options			
<- /				<- ~/pdfedit/src/gui			
Name	Size	MTime		Name	Size	MTime	
/.mc	1024	Apr 24 01:24		selfdest~dget.cc	971	May 4 15:13	
/bin	3072	Jun 1 02:24		selfdest~idget.h	679	Jul 18 02:08	
/boot	1024	Apr 24 18:16		settings.cc	12308	Jul 18 02:08	
~cdrom	11	Apr 23 19:56		settings.h	2061	Jul 18 02:08	
/dev	29696	Jul 25 07:37		staticse~ings.cc	3283	Jun 10 19:05	
/etc	3072	Jul 25 07:36		<b>staticsettings.h</b>	<b>643</b>	<b>Jun 10 03:51</b>	
/home	1024	Apr 23 20:25		statusbar.cc	2148	Jun 6 04:19	
/initrd	1024	Apr 23 19:58		statusbar.h	959	Jun 12 04:44	
/lib	5120	Jun 1 02:27		stringoption.cc	2015	Jul 18 02:08	
~lib64	4	May 31 09:24		stringoption.h	837	Jul 18 02:08	
/lost+found	12288	Apr 23 19:55		stringpr~erty.cc	2670	Jul 18 02:08	
/media	1024	Apr 23 19:56		stringproperty.h	1025	Jul 18 02:08	
/mnt	1024	Apr 24 18:23		test.qs	1454	Jun 24 02:11	
/opt	1024	Apr 23 19:58		toolbar.cc	2141	Jul 18 02:08	
/proc	0	Jul 25 07:36		toolbar.h	1113	Jul 18 02:08	
/root	1024	Apr 24 20:12		toolbutton.cc	1931	Jul 18 02:08	
/sbin	3072	Jun 1 02:25		toolbutton.h	1214	Jul 18 02:08	
-> /lib				staticsettings.h			

```
bilbo0u64:~/pdfedit/src/gui$
```

```
1Help 2Menu 3View 4Edit 5Copy 6RenMov 7Mkdir 8Delete 9PullDn 10Quit
```



# Egyszerű Jelölés (Markup)

- Ha már van egy ASCII szövegünk, akkor azt kiegészíthetjük olyan jelölésekkel, amikkel strukturált lehet a tartalma.
- Legegyszerűbb módja ha kiválasztunk adott karaktereket, amiknek többlet szerepük van a szövegben.
- Majd egy parser segítségével feldolgozzuk a szöveget.
- Például:
  - CSV szöveg esetén a **vessző** elválasztja az elemeket, a **sorvége** karakter pedig meghatározza az összetartozó adatokat.

```
Name;Neptun code  
Kortjohann Jonathan;QN3W1P  
Shorbila Seif Eldeen;UWRLPC  
Salama Ahmed;YLAYHF  
Ben el Heni Bilel;F62NON  
Bejaoui Ines;HV4IEZ  
Akari Mohamed Sabri;L3BH99  
Labidi Najdi;IL971U
```



# Egyszerű Jelölés (Markup)

- A megoldás gyengeségei:
  - Soronként ugyanazon adatok lehetnek,
  - a vesszők száma ugyanannyi kell legyen, tehát táblázatos az adat.
  - Minden oszlop ugyanolyan adatot tartalmazhat.

# Hogy segít ezen az XML?

- XML lehetővé teszi az adatok kisebb részekre bontását **alkotóelemek (element)** segítségével.
- Kitétel, hogy **legalább egy alkotóelem**mel rendelkeznie kell az XML dokumentumnak, ami **azonosítja** és **magába foglalja** a teljes dokumentumot.

# Konténer Elem (Container Element)

- Olyan elem, ami az általa azonosított adatokat tartalmazza.
- Három része van:
  - start-tag (kezdő címke)
  - adat
  - end-tag (lezáró címke)
- De nincsenek előre definiált alkotóeleme nevek.

# Start- és End-tag

- A **start-tag** a '<' és '>' jelek közé írt címkével valósul meg.
- Ehhez hasonlít az **end-tag** ami annyiban különbözik, hogy a '</' karakterekkel kezdődik és a '>' zárja le.

# Alapelem nevek megkötései

- A nevek kis-nagybetű érzékenyek,
- a névnek **betűvel kell kezdődnie**,
- az aláhúzás '\_' és a kettőspont engedélyezett,
- tartalmazhat számokat és néhány egyéb karaktert ('.' és '-').
- Példák:  

<p>, <X:123>, <ALongElementName>.

# Üres Elem

- Az elemeknek nem kell más elemet tartalmaznia.
- Ha ilyet hozunk létre az **helykitöltőként** (place holder) működik:

`<pageBreak></pageBreak>`

ennek rövidebb verziója pedig

`<pageBreak/>`

# Speciális Karakterek

A `<` és `>` vezérlő karakterek. Hogyan tudjuk akkor a következőt leírni XML-ben?

```
<code>if ( x < y ) { ... } </code>
```

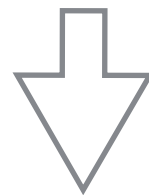
- **'&lt;'** (less than) reprezentálja a **'<'** karaktert
- **'&gt;'** (greater than) reprezentálja a **'>'** karaktert

*FIGYELEM! Pontosvessző (;) van a végén*

```
<code>if ( x &lt; y ) { ... } </code>
```

# Speciális Karakterek

The **&lt;name>** tag identifies a name.



The **<name>** tag identifies a name.



# Speciális Karakterek

- < megvalósítása: **&lt;**
- > megvalósítása: **&gt;**
- & megvalósítása: **&amp;**
- " megvalósítása: **&quot;**
- ' megvalósítása: **&apos;**

# Megjegyzések (Comments)

- A megjegyzés kezdetét ez jelzi: '`<!--`'
- Végét pedig a '`-->`'.
- Például:

```
<?xml version='1.0' encoding='UTF-8'?>
```

```
<!-- Based on an example in Wikipedia, March 2008 -->
```

```
<recipe name='bread' prep_time='5 mins' cook_time='3 hours'>
```

```
  <title>Basic bread</title>
```

```
  ....
```

```
</recipe>
```

# Attribútumok

- Az alkotóelemeknek (elements) lehetnek tulajdonságai (attribútumai), amik további információt tartalmaznak az elemről.
- Az attribútumok a nyitó tag (cimke) jelennek meg a következő formában:

**name='value'**

- Attribútumokat szóköz (space) választja el:  
`<ingredient amount='1' unit='teaspoon'>Salt</ingredient>`
- Lényeges, hogy **ugyanaz** az attribútum **csak egyszer** fordulhat elő az elem tulajdonságai között.

# Gyerek elemek, vagy attribútumok?

```
<match league="Barclays Premiership" date="2006-10-21" homeTeam =  
"Barcelona" awayTeam = "Manchester City" > .... </match>
```

vagy

**Nem mutat semmi relációt, emiatt csak adatnak tekinthető.**

```
<match>
```

```
  <league>Barclays Premiership</league>
```

```
  <date>2017-01-16</date>
```

```
  <teams>
```

```
    <home>Barcelona</home>
```

```
    <away>Manchester City</away>
```

```
  </teams>
```

```
</match>
```

**Az összetartozó elemek csoportosítva vannak.**

# Gyerek elemek, vagy attribútumok?

- Nincs egységes elv az eldöntésére.
- Ugyanakkor az eldöntésben segíthet az, hogy vannak megkötések az attribútumokkal szemben:
  - attribútumok csak rövid szövegek,
  - egy elemen belül egyedinek kell lenniük,
  - a sorrendjük nem lényeges.

# Gyerek elemek, vagy attribútumok?

- Tehát egy attribútum csak az elem egyedi tulajdonságának modellezésére alkalmas, egyszerű adattípussal, belső struktúra nélkül.

# Gyerek elemek, vagy attribútumok?

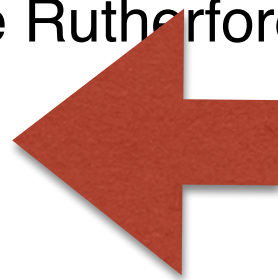
- Viszont amennyiben az adat:
  - rendelkezik struktúrával (egyszerűbb elemekből épül fel),
  - vannak ismétlődő komponensei (mint egy tömbnek),
  - meghatározott sorrendben fordulnak elő a komponensei,
  - hosszú szöveges tartalma van,

akkor az atribútum nem alkalmas a reprezentálására, ilyenkor biztosan **elemeket kell használni**.

# Még valami

- A nyitó elemet megfelelő sorrendben kell zárni is.

```
<?xml version="1.0" encoding="UTF-8"?>
<HighScore>
  <AngryBirds>
    <first>Bill Gates</first>
    <second>Paul Alen</second>
    <third>Csaba Fazekas</third>
  </AngryBirds>
  <FlappyBird>
    <first>Peter Gabriel</first>
    <second>Phil Colins</second>
    <third>Mike Rutherford</third>
  </HighScore>
</FlappyBird>
```

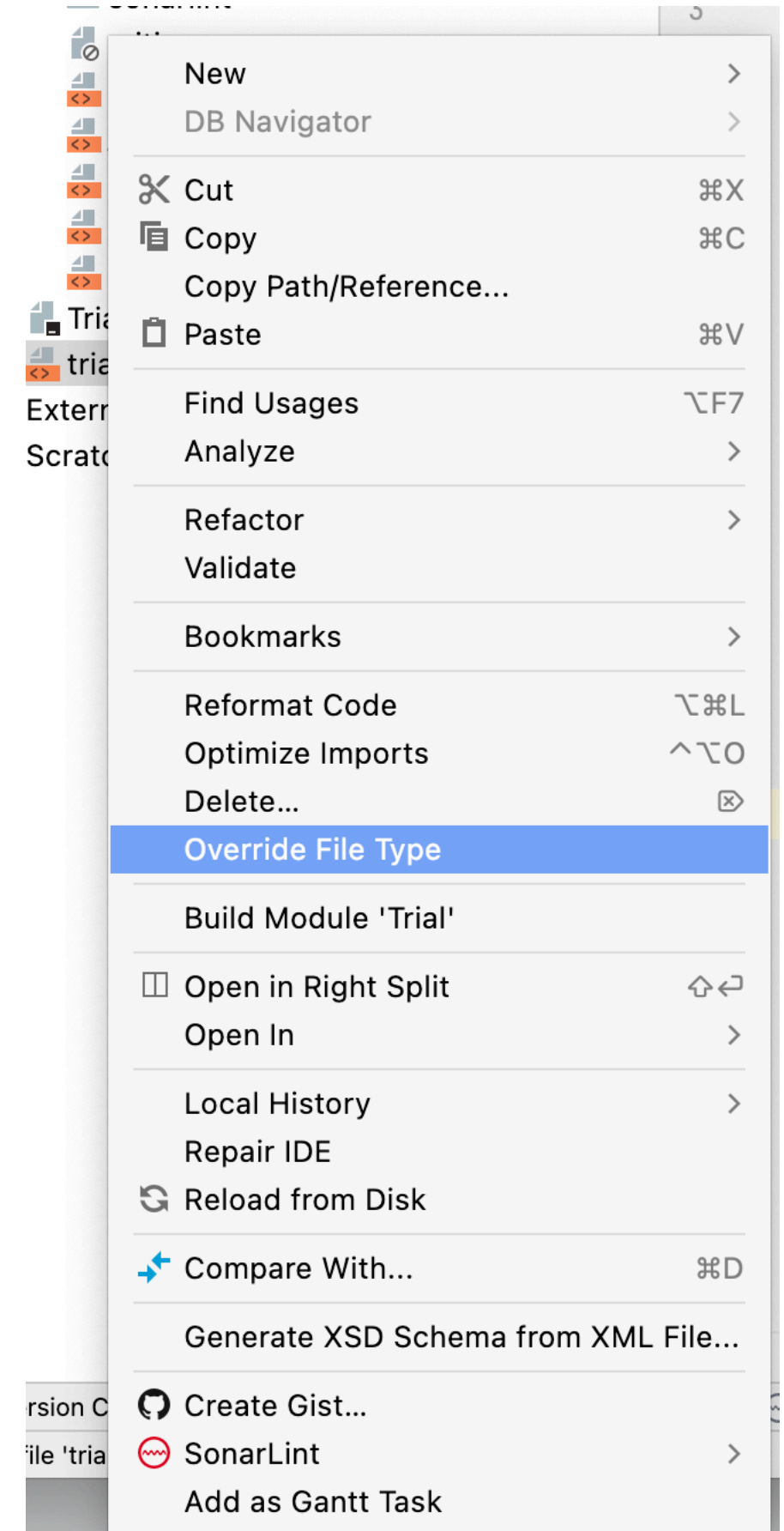


```
<?xml version="1.0" encoding="UTF-8"?>
<HighScore>
  <AngryBirds>
    <first>Bill Gates</first>
    <second>Paul Alen</second>
    <third>Csaba Fazekas</third>
  </AngryBirds>
  <FlappyBird>
    <first>Peter Gabriel</first>
    <second>Phil Colins</second>
    <third>Mike Rutherford</third>
  </FlappyBird>
</HighScore>
```



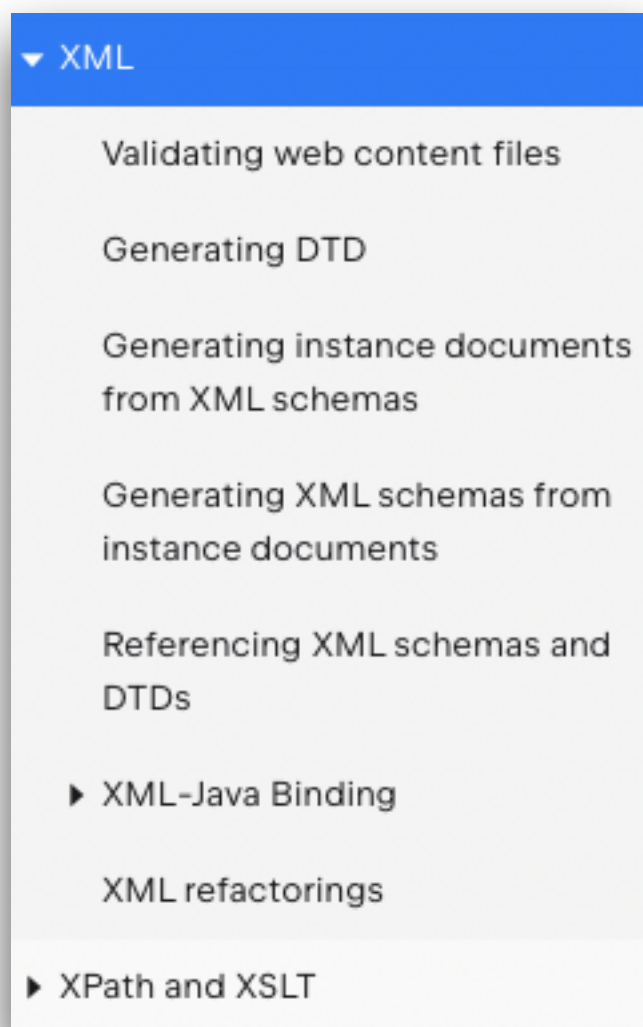
# IntelliJ használata

- Készítsünk egy üres projektet!
- Hozzunk létre egy xml kiterjesztésű fájlt!
- Állítsuk át az alapértelmezett fájl típusának értelmezését az IntelliJ-ben!



# IntelliJ használata

<https://www.jetbrains.com/help/idea/working-with-xml.html>



## XML

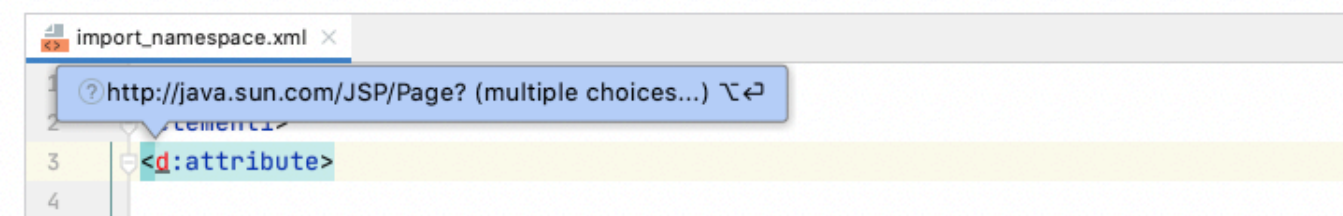
Last modified: 01 September 2023

IntelliJ IDEA brings powerful support for XML that includes structure validation, formatting (`⌘ Cmd` `⌘ Opt` `L`) and indentation (`⌘ Ctrl` `⌘ Opt` `I`) according to the XML code style, importing unbound namespaces, viewing code structure, unwrapping and removing tags (`⌘ Cmd` `⇧ Shift` `⌘ Delete`), generating DTD files and schemas from instance documents, as well as syntax and error highlighting, code completion (`⌘ Ctrl` `Space`), intention actions (`⌘ Opt` `⇧ Enter`), quick documentation look-up (`F1`), and more.

IntelliJ IDEA uses Xerces 2.11, an XML parser developed by Apache Software Foundation Group.

## Import XML namespaces

If you use a tag or an attribute from a namespace that is not bound, IntelliJ IDEA detects the problem and shows a tooltip:



# Készítsük el a következő XML dokumentumot!

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Document : db.xml -->
<Person>
  <name>
    <first>Bill</first>
    <family>Gates</family>
  </name>
  <PersonalDetails gender="male" personellNo="123456">
    <dob>1955-10-23</dob>
    <Address>
      <houseNumber>4</houseNumber>
      <street>Leanyka</street>
      <town>Eger</town>
      <postCode>3300</PostCode>
    </Address>
    <email type="home">info@ektf.hu</email>
    <PhoneNumbers>
      <home>003636123456</home>
      <mobile>003610123456</mobile>
      <work></work>
    </PhoneNumbers>
  </PersonalDetails>
</Person>
```

# 1. feladat

- Hozzunk létre egy XML dokumentumot ami egy könyv adatait tartalmazza a következők szerint:
  - Címe (title): "UML 2.0 in a nutshell".
  - Szerzők (authors): Dan Pilone and Neil Pitman.
  - Kiadó (publisher): O'Reilly.
  - Kiadás éve: 2005.
  - ISBN: 0-596-00795-7.
  - Előszóból (preface) és 12 fejezetből (chapter) áll, két függelék (appendix) van.
    - Fejezetek címe: Chapter 1, Chapter 2,... Chapter 12
    - Függelékek címe: Appendix A, Appendix B

```
<?xml version="1.0" encoding="UTF-8"?>
<book isbn="0-596-00795-7">
  <title>UML 2.0 in a nutshell</title>
  <authors>
    <author>Dan Pilone</author>
    <author>Neil Pitman</author>
  </authors>
  <publisher>O'Reilly</publisher>
  <year>2005</year>
  <content>
    <preface/>
    <chapters>
      <chapter>Chapter 1</chapter>
      <chapter>Chapter 2</chapter>
      <chapter>Chapter 12</chapter>
    </chapters>
    <appendices>
      <A> Appendix A</A>
      <B> Appendix B</B>
    </appendices>
  </content>
</book>
```

# 2. feladat

- Készítsen egy megfelelően formázott XML dokumentumot az önmaga leírásával!
- A gyökér elem legyen: <Person>.
- Miket szeret, utál, hobbik, tanulmányok stb.
- Használjon elemeket és ne attribútumokat!

# Document Type Definition (DTD)

# Validáció és a Document Type Definitions

- Egy **dokumentum modell**ben tudjuk megadni milyen elemeket és attribútumokat lehet használni az XML dokumentumban és milyen kapcsolatban állnak egymással.
- Egy XML dokumentum **valid** emennyiben az megfelel a dokumentum modellnek.

# Mi az a dokumentum modell?

- Megadhatjuk milyen elemeket és attribútumokat tartalmazhat egy XML dokumentum.
- Az eredmény olvasható emberek számára is.
- Azonban ha programok által feldolgozhatóvá akarjuk ezt tenni, akkor definiálni kell:
  - milyen elemek fordulhatnak elő,
  - hogyan vannak ezen elemek strukturálva,
  - erőfordulási sorrendjük,
  - milyen attribútumaik vannak, stb.



# Mi az a dokumentum modell?

- Le kell fektetni néhány szabályt ahhoz, hogy egyező dokumentumokat tudjunk definiálni.
- Lényegében mintha egy új nyelvet definiálnánk.
- Ezen szabályok összességét tartalmazza a **dokumentum modell**.
  - Mintha egy nyelv szintaxisa vagy nyelvtana lenne.

# Mi az a dokumentum modell?

- Olyan az XML dokumentum kapcsolata a dokumentum modellhez, mint az **objektum kapcsolat az osztályhoz**.
  - Az osztály definiálja a metódusokat és változókat.
  - Minden példány számára követendő szabályokat fektetnek le.
  - Ezek alapján létrehozott objektumok csak tartalmukban különbözhetnek.

# Mi az a dokumentum modell?

- Több módja is van a dokumentum modell definiálásának:
  - Document Type Definitions (DTDs)
  - XML Schema
  - stb.

# Egy példa XML fájl

```
<?xml version="1.0" encoding="UTF-8"?>
<email priority="High">
  <sender>fcsaba.eger@gmail.com</sender>
  <date>2017-02-01T11:11:11</date>
  <recipients>
    <recipient>info@youtube.com</recipient>
    <copy>admin@youtube.com</copy>
  </recipients>
  <subject>Bug report</subject>
  <body>I have found a bug ... </body>
</email>
```

# Ennek a DTD fájlja

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- root definition for email -->
<!ELEMENT email (sender, date, recipients, subject?, body?)>
<!ATTLIST email priority CDATA #REQUIRED>
<!-- definitions for level 1 elements -->
<!ELEMENT sender (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT recipients (recipient+, copy*, blind-copy*) >
<!ELEMENT subject (#PCDATA)>
<!ELEMENT body (#PCDATA)>
<!ELEMENT recipient (#PCDATA)>
<!ELEMENT copy (#PCDATA)>
<!ELEMENT blind-copy (#PCDATA)>
```

# DTD - Document Type Definition

- Minden sora '**<!**' kifejezéssel kezdődik és a '**>**' kifejezéssel zárul.
- Minden sor **<!ELEMENT definiálja** (zárójelek között) **az elem tartalmának modelljét** és az elem nevét közvetlenül az <!ELEMENT után.
- **Zárójelek között (vesszővel vagy | karakterrel szeparált lista) kompozíciót jelent.** (Tehát például egy email a következőkből áll: sender, date, recipients, stb., és ebben a sorrendben! Ez mutatja a szülő-gyerek relációt.)

# Elemek

- Formátuma

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

- Üres elem:

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

Példa:

```
<!ELEMENT br EMPTY>
```

XML example:

```
<br />
```

- Azon elemek, amiknek nincs gyerek eleme, tehát csak karakteres tartalma van, a következővel jelöljük: **#PCDATA** (parsed character data).

```
<!ELEMENT from (#PCDATA)>
```

- Gyerek elemek esetén fel kell sorolni azokat:

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

Példa:

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

# Mi van még a DTD-ben?

- **<!ATTLIST** majd az elem neve, és az adott elem attribútumai:

**<!ATTLIST email priority CDATA #REQUIRED>**

Ahol az email elemnek egyetlen attribútuma van: priority. Az attribútum típusa **CDATA** (character data) és szükséges attribútum (**#REQUIRED**).

- Előre megadhatunk adatokat is az attribútumnak listában felsorolva zárójelek között:

**<!ATTLIST email priority (Low|Medium|High) #REQUIRED>**

- A #REQUIRED mellett opcionális is lehet (**#IMPLIED**)
- Alapértelmezett érték is megadható:

**<!ATTLIST email priority (Low|Medium|High) "Low" >**



# Adott elem előfordulás számának megadása

- Amennyiben egyszer fordul elő:

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

Példa:

`<!ELEMENT note (message)>`

- Speciális karaktereket használunk a mennyiség jelzésére.
  - ‘\*’ jelzi a 0 vagy több előfordulást (0 - sok),
  - ‘+’ jelzi 1 vagy több (1- sok),
  - ‘?’ jelzi 0 vagy 1 (opcionális elem).

# Adott elem előfordulás számának megadása

- **Egyik** valamint a **vagy** jelzése:

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

Azt jelzi, hogy a note elemnek tartalmaznia kell "to", "from", "header" elemeket, valamint VAGY egy "message" VAGY egy "body" elemet.

- Változó tartalom jelzése:

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

A "note" elem tartalmazhat 0 vagy több parsed character data tartalmat, "to", "from", "header", vagy "message" elemet.

# Példa

<?xml version="1.0" encoding="UTF-8"?>

<!-- root definition for email -->

<!ELEMENT email (sender, date, recipients, subject?, body?)>

<!ATTLIST email priority CDATA #REQUIRED>

<!-- definitions for level 1 elements -->

<!ELEMENT sender (#PCDATA)>

<!ELEMENT date (#PCDATA)>

<!ELEMENT recipients (recipient+, copy\*, blind-copy\*) >

<!ELEMENT subject (#PCDATA)>

<!ELEMENT body (#PCDATA)>

<!ELEMENT recipient (#PCDATA)>

<!ELEMENT copy (#PCDATA)>

<!ELEMENT blind-copy (#PCDATA)>

Optional  
0 or 1

Optional  
0 or 1

0 or  
more

0 or  
more

1 or more.

There has to be at least  
one!

# Attributes

- Attribútumokat a ATTLIST segítségével sorolhatunk fel.

Szintaxisa:

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

DTD példa:

```
<!ATTLIST payment type CDATA "check">
```

XML példa:

```
<payment type="check" />
```

# attribute-type

Type	Description
CDATA	The value is character data
( <i>en1</i>   <i>en2</i>  ..)	The value must be one from an enumerated list
ID	The value is a unique id
IDREF	The value is the id of another element
IDREFS	The value is a list of other ids
NMTOKEN	The value is a valid XML name
NMTOKENS	The value is a list of valid XML names
ENTITY	The value is an entity
ENTITIES	The value is a list of entities
NOTATION	The value is a name of a notation
xml:	The value is a predefined xml value

# attribute-value

Value	Explanation
<i>value</i>	The default value of the attribute
#REQUIRED	The attribute is required
#IMPLIED	The attribute is optional
#FIXED <i>value</i>	The attribute value is fixed

# #REQUIRED

Használjuk a #REQUIRED kulcsszót amennyiben nincs lehetőség default érték megadásra, de az adott attribútum léte szükséges:

## Szintaxis

```
<!ATTLIST element-name attribute-name attribute-type #REQUIRED>
```

## Példa

DTD:

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

Valid XML:

```
<person number="5677" />
```

Invalid XML:

```
<person />
```

# #IMPLIED

A #IMPLIED használatos amikor nem akarjuk a szerkesztőt kényszeríteni, hogy adjon meg adott attribútumot, de lehetőség sincs alapértelmezett érték megadására:

Szintaxis:

```
<!ATTLIST element-name attribute-name attribute-type  
#IMPLIED>
```

Példa:

DTD:

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

Valid XML:

```
<contact fax="555-667788" />
```

Valid XML:

```
<contact />
```



# #FIXED

Amennyiben adott értéket akarunk használni és nem szeretnénk ha a szerkesztő megváltoztatná azt, akkor a #FIXED kulcsszóval tehetjük ezt meg.

Szintaxisa:

```
<!ATTLIST element-name attribute-name attribute-type #FIXED "value">
```

Példa

DTD:

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

Valid XML:

```
<sender company="Microsoft" />
```

Invalid XML:

```
<sender company="W3Schools" />
```

# Felsorolás (enumerált) értékek

Amennyiben meg akarjuk határozni a választható értékeket azokat felsorolhatjuk zárójelek között, illetve adhatunk neki alapértelmezett értéket is:

Szintaxisa:

```
<!ATTLIST element-name attribute-name (en1|en2|..) default-value>
```

Példa:

DTD:

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

XML example:

```
<payment type="check" />
```

or

```
<payment type="cash" />
```

# Hogyan rendelhetjük az XML fájlhoz a DTD-t?

- Bele tehetjük az XML fájlba egy `<!DOCTYPE` szekcióba. De ezt minden ilyen dokumentumba el kell helyezni ha több van.
- Tehetünk linket az XML dokumentum elejére ami hivatkozik a DTD fájlra helyben vagy egy URL-en.

# Fájlon belül

```
<?xml version="1.0"?>
<!DOCTYPE note [
  <!ELEMENT note (to,from,heading,body)>
  <!ELEMENT to (#PCDATA)>
  <!ELEMENT from (#PCDATA)>
  <!ELEMENT heading (#PCDATA)>
  <!ELEMENT body (#PCDATA)>
]>
<note>
  <to>Csaba Fazekas</to>
  <from>Istvan Fazekas</from>
  <heading>Reminder</heading>
  <body>Don't forget the meeting.</body>
</note>
```

**!DOCTYPE note** defines that the root element of this document is note

**!ELEMENT note** defines that the note element must contain four elements: "to,from,heading,body"

**!ELEMENT to** defines the to element to be of type "#PCDATA"

**!ELEMENT from** defines the from element to be of type "#PCDATA"

**!ELEMENT heading** defines the heading element to be of type "#PCDATA"

**!ELEMENT body** defines the body element to be of type "#PCDATA"

# Hivatkozással

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

```
<!DOCTYPE email SYSTEM "email.dtd">
```

```
<email priority="High">
```

```
  <sender>fcsaba.eger@gmail.com</sender>
```

```
  <date>2017-02-01T11:11:11</date>
```

```
  <recipients>
```

```
    <recipient>info@youtube.com</recipient>
```

```
    <copy>admin@youtube.com</copy>
```

```
  </recipients>
```

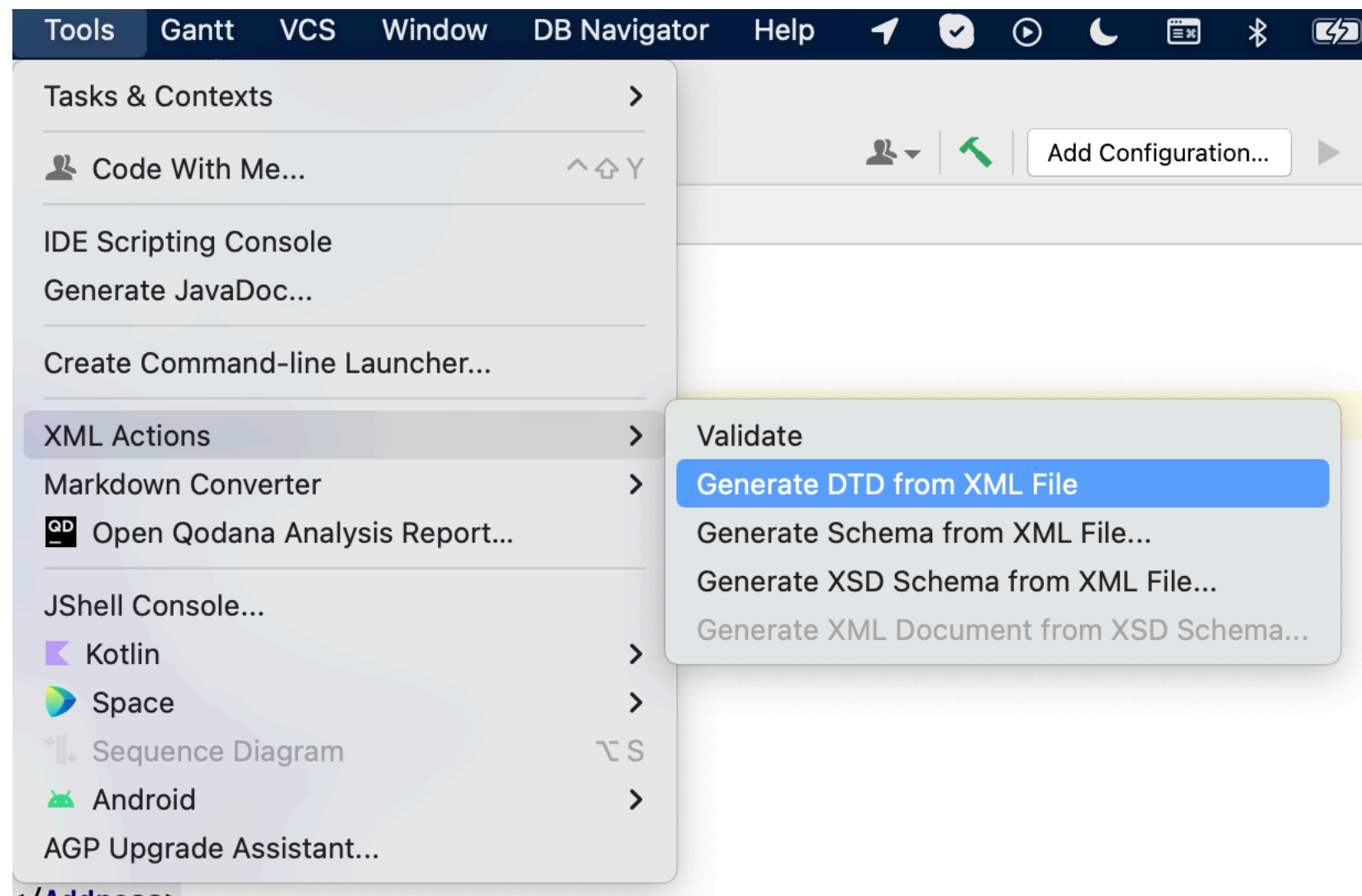
```
  <subject>Bug report</subject>
```

```
  <body>I have found a bug ... </body>
```

```
</email>
```

# IntelliJ használata

- Amennyiben az XML fájlunk alapértelmezett fájl típusa megfelelő, akkor a Tools / XML Actions menüben a DTD létrehozás engedélyezett.



# 1. feladat

- Készítsünk email XML fájlt, DTD-vel.

Create an XML file called *email.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE email SYSTEM "email.dtd">
<email priority="High">
  <sender>fcsaba.eger@gmail.com</sender>
  <date>2017-02-01T11:11:11</date>
  <recipients>
    <recipient>info@youtube.com</recipient>
    <copy>admin@youtube.com</copy>
  </recipients>
  <subject>Bug report</subject>
  <body>I have found a bug ... </body>
</email>
```

Create a DTD file called *email.dtd*

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- root definition for email -->
<!ELEMENT email (sender, date, recipients, subject?,
body?)>
<!ATTLIST email priority CDATA #REQUIRED>
<!-- definitions for level 1 elements -->
<!ELEMENT sender (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT recipients (recipient+, copy*, blind-copy*) >
<!ELEMENT subject (#PCDATA)>
<!ELEMENT body (#PCDATA)>
<!ELEMENT recipient (#PCDATA)>
<!ELEMENT copy (#PCDATA)>
<!ELEMENT blind-copy (#PCDATA)>
```

# 2. feladat

- Készítsen a recipe.xml fájlhoz recipe.dtd fájlt!

Create an XML file called *recipe.xml*

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE recipe SYSTEM "recipe.dtd">
<recipe name='bread' prep_time='5 mins' cook_time='3 hours'>
  <title>Basic bread</title>
  <ingredient amount='3' unit='cups'>Flour</ingredient>
  <ingredient amount='0.25' unit='ounce'>Yeast</ingredient>
  <ingredient amount='1.5' unit='cups' state='warm'>Water</ingredient>
  <ingredient amount='1' unit='teaspoon'>Salt</ingredient>
  <instructions>
    <step>Mix all ingredients together, and knead thoroughly.</step>
    <step>Cover with a cloth, and leave for one hour in warm room.</step>
    <step>Knead again, place in a tin, and then bake in the oven.</step>
  </instructions>
</recipe>
```



# 2. feladat megoldása

- A DTD fájl

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT recipe (title, ingredient+, instructions+) >
<!ATTLIST recipe name CDATA #REQUIRED>
<!ATTLIST recipe prep_time CDATA #REQUIRED>
<!ATTLIST recipe cook_time CDATA #REQUIRED>

<!ELEMENT title (#PCDATA) >

<!ELEMENT ingredient (#PCDATA) >
<!ATTLIST ingredient
    amount CDATA #REQUIRED
    unit CDATA #REQUIRED
    state (warm|cold) #IMPLIED >

<!ELEMENT instructions (step*) >
<!ELEMENT step (#PCDATA) >
```

# 3. feladat

- Készítsen DTD fájlt a match.xml fájlhoz!

Create an XML file called *match.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE match SYSTEM "match.dtd">
<match league="Barclays Premiership" date="2017-03-06">
  <teams>
    <home goals="2">Chelsea</home>
    <away goals="1">Portsmouth</away>
  </teams>
  <scorers>
    <homeScorer>Drogba</homeScorer>
    <homeScorer>Schevchenko</homeScorer>
    <awayScorer>Benjani</awayScorer>
  </scorers>
  <referee>Dermot Gallagher</referee>
  <attendance>45326</attendance>
</match>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT match (teams, scorers, referee, attendance)>
<!ATTLIST match
  date CDATA #REQUIRED
  league CDATA "Barclays Premiership">

<!ELEMENT teams (home, away)>
<!ELEMENT scorers (homeScorer*, awayScorer*)>
<!ELEMENT referee (#PCDATA)>
<!ELEMENT attendance (#PCDATA)>

<!ELEMENT home (#PCDATA)>
<!ATTLIST home
  goals CDATA "0">

<!ELEMENT away (#PCDATA)>
<!ATTLIST away
  goals CDATA "0">

<!ELEMENT homeScorer (#PCDATA)>
<!ELEMENT awayScorer (#PCDATA)>
```

# **XML Namespace**

# XML Namespace

- Létrehozhatjuk saját elemeinket, attribútumainkat.
- De lehet ugyanaz a név már előfordul és más jelentéssel.
- Például:

An XML element:

```
<table>
  <type>Tiled Japanese Low</type>
  <width units='cm'>80</width>
  <length units='cm'>120</length>
  <legs>4</legs>
</table>
```

A HTML table:

```
<table>
  <row>
    < cell> ... </cell>
    < cell  ... </cell>
    <cell> ... </cell>
  </row>
  <row> ... </row>
</table>
```

**table - név ütközést okoz**

# Név előtagok (prefix)

- Ennek kivédésére használhatunk előtagot; például: **furn:table** ahol a **furn:** az előtag,
- és a **struct:table** használható az adat részre, ahol a **struct:** az előtag. Így átalakítható a table:

<furn:table>

<furn:type>Tiled Japanese Low</furn:type>

<furn:width units='cm'>80</furn:width>

<furn:length units='cm'>120</furn:length>

<furn:legs>4</furn:legs>

</furn:table>

# Név előtagok (prefix)

- Ugyanakkor az előtag nem oldja meg a problémát.
- Mivel **nem garantálható**, hogy az Interneten még nincs olyan rendszer amiben ne lenne már `<urn:table>`.
- Az előtagoknak **globálisan egyedinek** kell lennie, de ezen stringek **nem** lehetnek **irreálisan hosszúak**.

# Névtér (namespace)

- Következésképpen, **az előtag csak lokálisan definiált név**, ami egy adott dokumentum és gyerek elemei számára használható.
- De ez egy globálisan egyedi azonosítóhoz tartozhat egy **névtér deklaráció**ban.
- Ami az Uniform Resource Identifier (**URI**) formátumot alkalmazza, mert a névtérnek továbbra is egyedinek kell lennie az Interneten.
- A névtér deklaráció a kezdő címkében történik meg a a gyökér elemnél az **xmlns** előtag használatával.

# Névtér (namespace)

Például:

```
<struct:table xmlns:struct="http://eke-uni.hu/~fcsaba/structure">
```

Ez a következő URL-t deklarálja a struct előtaghoz:

<http://eke-uni.hu/~fcsaba/structure>



# Névtér (namespace)

- Figyeljük meg, hogy a **névtér deklaráció a nyitó elemben történik** és már ez is használja a előtagot (<struct:table>).
- Ugyanis a **névtér hatálya** már az első címkétől (tag) számít az azt lezáró címkét is beleértve.
- Minden gyerek elem és attribútum számára használható ezen belül, bár attribútumoknál szokatlan a használata.
- A név ami prefixet használ mésképpen **minősített névnek (qualified name)** hícjuk, és egy **névtér előtagból (namespace prefix)** ( esetünkben struct) **valamint egy helyi névből** (table) áll, amit a kettőspont választ el egymástól (“:”).
- Így készen is van a megoldás a “table” problémánkra. Két névteret definiálunk a gyökér elemében.

# Névtér (namespace)

- Megoldás tehát:
  - Két névtér.

```
<?xml version="1.0" encoding="UTF-8"?>
<struct:table
  xmlns:struct='http://eke-uni.hu/~fcsaba/structure'
  xmlns:furn='http://eke-uni.hu/~fcsaba/furniture'>
  <struct:row>
    <struct:cell>
      <furn:table>
        <furn:type>Tiled Japanese Low</furn:type>
        <furn:width units='cm'>80</furn:width>
        <furn:length units='cm'>120</furn:length>
        <furn:legs>4</furn:legs>
      </furn:table>
    </struct:cell>
    <struct:cell>
      <furn:table>....</furn:table>
    </struct:cell>
  </struct:row>
</struct:table>
```

# Alapértelmezett Névter (default namespace)

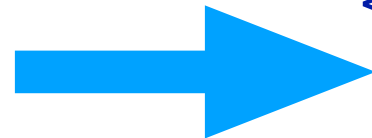
- Ha nem adunk meg prefix-et, akkor **alapértelmezett névteret** definiálunk:

`xmlns="http://eke-uni.hu/~fcsaba/structure"`

- Ekkor feltételezzük, hogy minden név innen érkezik.

# Alapértelmezett Névter (default namespace)

- Előbbi megoldás alternatívája lehet:



```
<?xml version="1.0" encoding="UTF-8"?>
<table
  xmlns='http://eke-uni.hu/~fcsaba/structure'
  xmlns:furn='http://eke-uni.hu/~fcsaba/furniture'>
  <row>
    <cell>
      <furn:table>
        <furn:type>Tiled Japanese Low</furn:type>
        <furn:width units='cm'>80</furn:width>
        <furn:length units='cm'>120</furn:length>
        <furn:legs>4</furn:legs>
      </furn:table>
    </cell>
    <cell>
      <furn:table>....</furn:table>
    </cell>
  </row>
</table>
```

# Miért használhatunk alapértelmezett névteret?

- Ugyanakkor ne feledjük, hogy a névtér deklaráció egy elemen történik, annak nyitó címkéjében és a **névtér hatálya ezen elemre és gyerek elemeire lesz értelmezve.**

# 1. feladat

- An engineering company manufactures assemblies from components supplied by other companies.
- Here is an XML document describing an assembly:

The two `part` elements have different formats because they are based on information provided by the two suppliers: A1 Manufacturing's `part` element has `type`, `length` and `diameter` children, whereas Turbo Engineering has `name`, `size`, etc. Edit this document so that:

- A1's part element and its children come from the namespace with an `a1` prefix:
- `http://a1manufacturing.com/widget`
- Acme's part element and its children come from the namespace with `turbo` prefix:
- `http://turbo-engineering.com/comps/18/gasket`
- Other names come from a default namespace
- `http://bestsupplier.com/assembly/32`

```
<?xml version="1.0" encoding="UTF-8"?>
<assembly>
  <component partNo="1" quantity="3">
    <supplier>A1 Manufacturing</supplier>
    <part>
      <type>widget</type>
      <length units="mm">42</length>
      <diameter units="mm">157</diameter>
    </part>
  </component>

  <component partNo="2" quantity="5">
    <supplier>Turbo Engineering</supplier>
    <part>
      <name>gasket</name>
      <size>
        <unit>cm</unit>
        <dim1>3.1</dim1>
        <dim2>1.4</dim2>
      </size>
    </part>
  </component>
</assembly>
```

# Egy lehetséges megoldás

```
<?xml version="1.0" encoding="UTF-8"?>
<assembly xmlns="http://bestsupplier.com/assembly/32">
  <component partNo="1" quantity="3">
    <supplier>A1 Manufacturing</supplier>
    <a1:part xmlns:a1="http://a1manufacturing.com/widget">
      <a1:type>widget</a1:type>
      <a1:length units="mm">42</a1:length>
      <a1:diameter units="mm">157</a1:diameter>
    </a1:part>
  </component>

  <component partNo="2" quantity="5">
    <supplier>Turbo Engineering</supplier>
    <turbo:part
      xmlns:turbo="http://turbo-engineering.com/comps/18/gasket">
      <turbo:name>gasket</turbo:name>
      <turbo:size>
        <turbo:unit>cm</turbo:unit>
        <turbo:dim1>3.1</turbo:dim1>
        <turbo:dim2>1.4</turbo:dim2>
      </turbo:size>
    </turbo:part>
  </component>
</assembly>
```

# **XML Schema**



# XML Schema

- XML schemas are an alternative to Document Type Definitions for describing a document model.
- Schemas have a number of advantages over DTDs, and are gradually replacing them for all but the simplest of tasks.

# XML Schema

- An XML schema, like a DTD, **describes the structure of a class of XML documents** for a particular application (these are the target documents of the schema).
- In defining a document model, we need ways of describing **what elements can occur** in the target document(s), **what these elements consist of** (other elements, or character data, etc), the **order** in which these elements occur, **how many times** each one can occur, what attributes they have, etc.
- We saw before how all of this could be done using DTDs. A DTD did all of this using a special syntax (<!ELEMENT, <!ATTLIST, etc) with its own peculiarities.

# XML Schema

- An XML schema, however, uses XML itself to describe a document model.
- An **XML schema is itself a valid XML document** (which implies that there is a schema that describes the rules for a schema!). This is an immediate advantage of schemas: we can use all of the XML tools at our disposal (and our understanding of XML) when dealing with them.
- So in an XML schema we would expect to find schema elements and attributes being used to describe the elements and attributes of the target documents. The schema elements and attributes belong to a namespace identified by the URL:

<http://www.w3.org/2001/XMLSchema>

# XML Schema

- It is common practice to refer to this namespace **using the prefix xs** (or **xsd**) so in what follows any name that begins xs: is the name of a schema element.
- The root element of any schema has the name schema, so any schema will have an outer root element that, including the namespace declaration, looks like this:

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

- Conventionally, schemas are stored in files with a **.xsd** extension (presumably for “**X**ML **S**chema **D**efinition”).

# Basic elements of schema

- The most important element used in schema is called **element** because it defines an element in the target document. For example:

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

- Notice that this schema element is an empty element with just attributes. It states that there is an element in the target document with name sender, at this point. The **type** of this document is **xs:string**. This is more or less identical in effect to the DTD line:

`<!ELEMENT sender (#PCDATA)>`

# Basic elements of schema

- There are a number of optional attributes that element can take, in addition to name and type.
- For multiplicity, we have **minOccurs** and **maxOccurs**. These have **default values** of **1** (so *not necessary for elements that only occur once*), and can have any sensible integer value, or the **special value “unbounded” for many with no limit.**

```
<xs:element name="scorer" type="xs:string" minOccurs="0"/> <!-- 0..1 -->
```

```
<xs:element name="copy" minOccurs="0" maxOccurs="unbounded"/>
```

```
<xs:element name="dayTotal" type="xs:string" maxOccurs="7"/> <!-- 1..7 -->
```

# Simple and complex types and content

- XML schema provide a number of simple data types similar to the built-in types of programming languages.
- In addition, we need ways of defining structured types in the target document, elements that have a complex structure, including attributes and child elements.

# Simple and complex types and content

Types in XML schema fall into two categories:

- **xs:simpleType** - any type that just contains character content (and no attributes). We will only need this if we want to restrict the value in some way (e.g. an enumeration like (low|medium|high) in a DTD, or an integer in a particular range).
- **xs:complexType** - any type that contains child elements or attributes, or both. There is a further subdivision here:
  - xs:complexType with xs:simpleContent **for elements with character data and attributes, but no child elements**
  - xs:complexType with xs:complexContent **for elements with children (and possibly attributes)**



# Elements with children

- To define an element that has children in a particular sequence, we use a **complexType** which is a sequence. For example, to define the equivalent, from the DTD, of:

<!ELEMENT email (sender, date, recipients, subject?, body?)>

- we need:

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

```
  <xs:complexType>
```

```
    <xs:sequence>
```

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

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

```
      <xs:element ref="recipients"/>
```

```
      <xs:element name="subject" type="xs:string" minOccurs="0"/>
```

```
      <xs:element name="body" type="xs:string" minOccurs="0"/>
```

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

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

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



recipients may have a  
sequence of children

# Elements with children and attributes

- If there are attributes as well (<email priority="high">), we include each in an **xs:attribute** element inside the complexType, but after closing </xs:sequence> tag, as shown by the priority attribute in the previous example.

```
<xs:complexType>
```

```
  <xs:sequence>
```

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

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

```
    <xs:element ref="recipients"/>
```

```
    <xs:element name="subject" type="xs:string" minOccurs="0"/>
```

```
    <xs:element name="body" type="xs:string" minOccurs="0"/>
```

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

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

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

# Create an XML file for the following contacts.xsd file

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="person">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="name" type="xs:string"/>
        <xs:element name="title" type="xs:string"/>
        <xs:element name="occupation" type="xs:string"/>
      </xs:sequence>

      <xs:attribute name="birthdate" type="xs:date"/>
      <xs:attribute name="weight" type="xs:integer"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

First create this contacts.xsd file :-)

# A valid contacts.xml file:

```
<?xml version="1.0" encoding="UTF-8"?>  
<person birthdate="1964-01-12" weight="65">  
  <name>Jeff Bezos</name>  
  <title>Mr.</title>  
  <occupation>Business Leader, Entrepreneur</occupation>  
</person>
```

# Linking a document to its schema

- Linking a target document to a schema is simplest if the target document does not use namespaces. In this case, an addition to the **root element** will do, like this:

```
<email
```

```
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
  xsi:noNamespaceSchemaLocation="email-basic.xsd"
```

```
  priority="high">
```

- This is the root element of an email document (with a priority attribute).
- The line before the attribute is a **noNamespaceSchemaLocation** attribute identifies where to find the schema for all names in this document which are not in a namespace.
- The value of this attribute is a URL which identifies the location of this schema. In this case, this is just a file in the same folder as the target document, so we have a simple file name (email-basic.xsd).
- The line before this identifies the namespace from which noNamespaceSchemaLocation itself comes, using the prefix xsi (for "XML schema instance").

# Linking a document to its schema

- If the target document uses namespaces, there is a **schemaLocation** attribute, the value of which is a string in which the namespace identifier (the full one, not the prefix!) is **followed by a space and then the URL of the schema.**

<match

xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'

xmlns:m='http://eke-uni.hu/~fcsaba/ex/42'

xsi:schemaLocation='http://eke-uni.hu/~fcsaba/ex/42 match1.xsd'>

- For simplicity, we won't combine schemas and namespaces in this unit, but you will find that most realistic examples do. It is possible (and this is largely the point) to combine names from multiple namespaces in a single document, and have a parser check that their usage is valid.

# Connect the XSD file in the contacts.xml file:

```
<?xml version="1.0" encoding="UTF-8"?>
<person
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="contacts.xsd"
  birthdate="1964-01-12" weight="65">
  <name>Jeff Bezos</name>
  <title>Mr.</title>
  <occupation>Business Leader, Entrepreneur</occupation>
</person>
```

# XML Types

- Character - xs:string
- Logical - xs:boolean
- Number - xs:decimal, xs:integer ...
- Date - xs:date, xs:time, xs:duration ...
- Custom types - xs:simpleType



# XML schema data types

- XML Schema come with a number of built-in simple data types.
- You can find a list of them:  
<http://www.w3.org/TR/xmlschema-0/#SimpleTypeFacets>
- Apart from **string**, these include **various numeric types** (xs:integer, xs:positiveInteger, xs:double), **date and time** (xs:date, xs:dateTime, etc.) and a **Boolean** (xs:boolean).
- This is far more useful than DTDs, where (apart from enumerations for attributes, and some other simple types for attributes), all character data is just treated as a string.

# XML schema data types

- Character based:

- xs:string

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

`<name>Csaba Fazekas</name>`

- xs:normalizedString

`<xs:element name="customer" type="xs:normalizedString"/>`

An element in your document might look like this:

`<customer>John Smith</customer>`

Or it might look like this:

`<customer>            John Smith            </customer>`

# XML schema data types

- Derived from xs:string:

Name	Description
ENTITIES	
ENTITY	
ID	A string that represents the ID attribute in XML (only used with schema attributes)
IDREF	A string that represents the IDREF attribute in XML (only used with schema attributes)
IDREFS	
language	A string that contains a valid language id
Name	A string that contains a valid XML name
NCName	
NMTOKEN	A string that represents the NMTOKEN attribute in XML (only used with schema attributes)
NMTOKENS	
QName	
token	A string that does not contain line feeds, carriage returns, tabs, leading or trailing spaces, or multiple spaces

# XML schema data types

- Boolean: xs:boolean - “true” / “false” or “1” / “0”

<xs:attribute name=“retired” type=“xs:boolean”/>

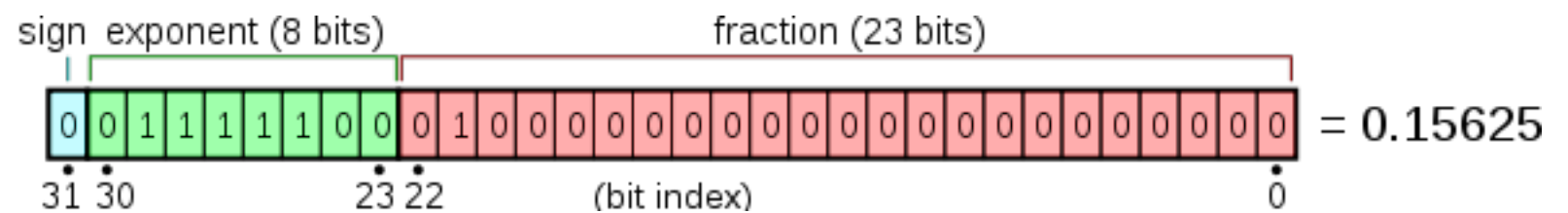
<person retired=“false”>

    <name>Csaba Fazekas</name>

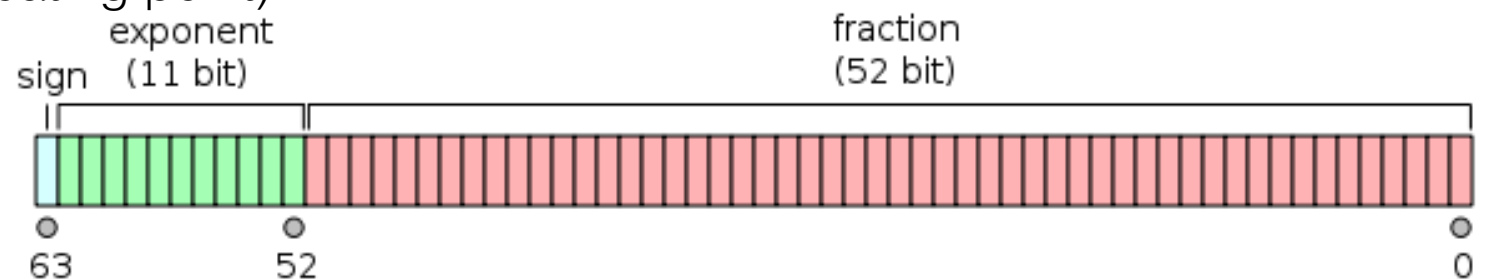
</person>

# XML schema data types

- xs:integer (3)
- xs:decimal (3.14)
- xs:float (32 bit simple precision floating point : maximum value of  $2^{31} - 1 = 2,147,483,647$ )



- xs:double (64 bit double precision floating point)



<xs:attribute name="height" type="xs:decimal"/>

<xs:attribute name="weight" type="xs:integer"/>

<person height="6.5" weight="180">

    <name> XY </name>

</person>

# XML schema data types

- xs:time - an instant of time that recurs every day (24h format!)
- xs:duration (PyyYymmMddDThhHmmMssS
  - `<period>P5Y2M10D</period>` = 5 years 2 months 10 day
  - `<period>PT15H</period>` = 15 hours)
- xs:date (1970-01-01 (Its format: ccyy-mm-dd)
- xs:month (ccyy-mm)
- xs:year (ccyy)
- xs:century (cc)
- xs:recurringDate (- -mm-dd)
- xs:recurringDay (- -dd)
- xs:dateTime (ccyy-mm-ddThh:mm:ss = `<startdate>2002-05-30T09:00:00</startdate>`)

`<xs:attribute name="birthdayparty" type="xs:recurringDate"/>`

`<xs:attribute name="height" type="xs:decimal"/>`

`<xs:attribute name="weight" type="xs:integer"/>`

`<person birthdayparty="--03-14" height="6.5" weight="180">`

`<name> XY </name>`

`</person>`

# XML Types

- **xs:simpleType** - any type that just contains character content (and no attributes).
- we can refine the base types with **restrictions** with the Simple Type
- Simple types can be restricted in various ways, to limit the values which can be taken.

```
<xs:simpleType name="name">  
  <xs:restriction base="xs:integer">  
  </xs:restriction>  
</xs:simpleType>
```

# xs:simpleType

- simpleType - we can set the bounds for values
  - xs:minInclusive
  - xs:minExclusive
  - xs:maxInclusive
  - xs:maxExclusive

```
<xs:simpleType name="oneToTen">  
  <xs:restriction base="xs:int">  
    <xs:minInclusive value="1"/>  
    <xs:maxInclusive value="10"/>  
  </xs:restriction>  
</xs:simpleType>
```

```
<xs:simpleType name="oneToTen">  
  <xs:restriction base="xs:int">  
    <xs:minInclusive value="1"/>  
    <xs:maxExclusive value="11"/>  
  </xs:restriction>  
</xs:simpleType>
```

```
<xs:simpleType name="greaterThanTen">  
  <xs:restriction base="xs:int">  
    <xs:minExclusive value="10"/>  
  </xs:restriction>  
</xs:simpleType>
```



# Enumeration with simple type

- Defining an **enumeration** like (low|medium|high) for the priority in the email element (`<email priority="high"></email>`), we would restrict a string like this:

```
<xs:element name='priority'>
  <xs:simpleType>
    <xs:restriction base='xs:string'>
      <xs:enumeration value='low'/>
      <xs:enumeration value='medium'/>
      <xs:enumeration value='high'/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

# List as a simple type

- While enumeration restrict the element to have one value, a list allows several values:

```
<xs:element name='rainfall'>  
  <xs:simpleType>  
    <xs:list base='xs:decimal'>  
      <xs:length value='5' />  
    </xs:restriction>  
  </xs:simpleType>  
</xs:element>
```

An example XML value:

```
<rainfall> 1.25, 2.0, 3.0, 4.25, 3.75 </rainfall>
```

# Restriction of simple types

- We can **restrict an integer** to come from a limited range:

```
<xs:element name='testMark'>
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="10"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

- Any testMark element in a target document will have its content checked for values '0' to '10'.

# Restriction of simple types

- Restrict the **length of a string**:

```
<xs:element name='password'>
```

```
  <xs:simpleType>
```

```
    <xs:restriction base="xs:string">
```

```
      <xs:length value="8"/>
```

```
    </xs:restriction>
```

```
  </xs:simpleType>
```

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

- The password field has to have **exactly** 8 characters.

# Restriction of simple types

- Restrict the **length of a string**:

```
<xs:element name='password'>
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:minLength value="8"/>
      <xs:maxLength value="12"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

- The password field has to have **minimum 8 characters, but no more than 12**.

# Restriction of simple types

- Setting the **precision of a number**:

```
<xs:element name='balance'>
```

```
  <xs:simpleType>
```

```
    <xs:restriction base="xs:decimal">
```

```
      <xs:precision value="6"/> <!-- The total amount of numbers. -->
```

```
      <xs:scale value="2"/>
```

```
    </xs:restriction>
```

```
  </xs:simpleType>
```

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

- Example:
  - <balance>3.14</balance>
  - <balance>123.45</balance>
  - <balance>9999.99</balance>

# Restrictions on a Series of Values

- To limit the content of an XML element to define a series of numbers or letters that can be used, we would use the pattern constraint.
- The example below defines an element called "letter" with a restriction. The only acceptable value is ONE of the LOWERCASE letters from a to z:

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

# Restrictions on a Series of Values

- The next example defines an element called "initials" with a restriction. The only acceptable value is THREE of the UPPERCASE letters from a to z:

```
<xs:element name="initials">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[A-Z][A-Z][A-Z]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```



# Restrictions on a Series of Values

- The next example defines an element called "choice" with a restriction. The only acceptable value is ONE of the following letters: x, y, OR z:

```
<xs:element name="choice">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[xyz]" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

# Restrictions on a Series of Values

- The next example defines an element called "prodid" with a restriction. The only acceptable value is FIVE digits in a sequence, and each digit must be in a range from 0 to 9:

```
<xs:element name="prodid">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:pattern value="[0-9][0-9][0-9][0-9][0-9]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

# Restrictions on a Series of Values

- The next example defines an element called "gender" with a restriction. The only acceptable value is male OR female:

```
<xs:element name="gender">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="male|female"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

# Restrictions on a Series of Values

- The next example defines an element called "password" with a restriction. There must be exactly eight characters in a row and those characters must be lowercase or uppercase letters from a to z, or a number from 0 to 9:

```
<xs:element name="password">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-zA-Z0-9]{8}" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

# Type declarations

- Any of the types we have used in the element declarations in the examples before can be declared as a data type, and then be used as the value of the type attribute of an attribute or element in the schema.
- This is useful if the type is needed in more than one place, and to keep the complexity of the schema under control.

# Type declaration

- Rather than giving the definition for the type as part of the an element, we can refer for a custom type

```
<xs:element name='priority' type='priorityType'/>
```

- and **at the outer level of the document, just inside the schema element** we can create the type definition

```
<xs:simpleType name='priorityType'>
```

```
  <xs:restriction base='xs:string'>
```

```
    <xs:enumeration value='low'/>
```

```
    <xs:enumeration value='medium'/>
```

```
    <xs:enumeration value='high'/>
```

```
  </xs:restriction>
```

```
</xs:simpleType>
```

# Type declaration

- Or we can restrict any base type right in the definition of the element or attribute without declaring a new type:

```
<xs:element name='rating'>
```

```
  <xs:simpleType>
```

```
    <xs:restriction base='xs:integer'>
```

```
      <xs:minInclusive value='1'/>
```

```
      <xs:maxInclusive value='10'/>
```

```
    </xs:restriction>
```

```
  </xs:simpleType>
```

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

# What is a Complex Element?

- It is an XML element that contains other elements and/or attributes.
- There are four kinds of complex elements:
  - empty elements
  - elements that contain only other elements
  - elements that contain only text
  - elements that contain both other elements and text
- **Any element with an attribute belongs to a complex type.**



# Empty element

- An empty element looks like this:

```
<image src='http://eke-uni.hu/~fcsaba/pics/012.jpg'/>
```

- It would be modeled in the schema like this:

```
<xs:element name='image'>
```

```
  <xs:complexType>
```

```
    <xs:attribute name='src' type='xs:string'/>
```

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

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

# Complex Types Containing Elements Only

An XML element, "person", that contains only other elements:

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

You can define the "person" element in a schema, like this:

```
<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>
```

Notice the `<xs:sequence>` tag. It means that the elements defined ("firstname" and "lastname") must appear in that order inside a "person" element.

# Complex Types Containing Elements Only

Or you can give the complexType element a name, and let the "person" element have a type attribute that refers to the name of the complexType (if you use this method, several elements can refer to the same complex type):

```
<xs:element name="person" type="persontype"/>

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

# Complex Text-Only Elements

- An element with **simple content** (no elements but data) and with any attributes belongs to a **complex type**.
- The attributes are regarded as an **extension** of the string **simpleContent** of the element. For example:

```
<body language='English' wordCount='6'>Here is some text in English</body>
```

- Its schema definition:

```
<xs:element name='body'>
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name='language' type='xs:string'/>
        <xs:attribute name='wordCount' type='xs:integer'/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

- Note that we can type wordCount as an integer, which means that a parser can check that its value is sensible. This illustrates the availability of some simple data types for character values of attributes, or simple content of elements.

# Complex Types with Mixed Content

An XML element, "letter", that contains both text and other elements:

```
<letter>
  Dear Mr.<name>John Smith</name>.
  Your order <orderid>1032</orderid>
  will be shipped on <shipdate>2001-07-13</shipdate>.
</letter>
```

The following schema declares the "letter" element:

```
<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>
```

**Note:** To enable character data to appear between the child-elements of "letter", the mixed attribute must be set to "true". The <xs:sequence> tag means that the elements defined (name, orderid and shipdate) must appear in that order inside a "letter" element.

# Exercise 1

- Create a schema for the email.xml file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- <!DOCTYPE email SYSTEM "email.dtd"> -->
<email
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="email.xsd"
  priority="high">

  <sender>fcsaba.eger@gmail.com</sender>
  <date>2017-02-01T11:11:11</date>
  <recipients>
    <recipient>info@youtube.com</recipient>
    <copy>admin@youtube.com</copy>
  </recipients>
  <subject>Bug report</subject>
  <body>I have found a bug ...</body>
</email>
```

# Exercise 1

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" >
  <xs:element name="email">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="sender" type="xs:string"/>
        <xs:element name="date" type="xs:string"/>
        <xs:element ref="recipients"/>
        <xs:element name="subject" type="xs:string" minOccurs="0"/>
        <xs:element name="body" type="xs:string" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute name="priority" type="xs:string"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="recipients">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="recipient" maxOccurs="unbounded"/>
        <xs:element ref="copy" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="blind-copy" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="recipient" type="xs:string"/>
  <xs:element name="copy" type="xs:string"/>
  <xs:element name="blind-copy" type="xs:string"/>
</xs:schema>
```

# Exercise 2

- Find the problems in the following XML file with the help of the schema file from Exercise 1.

```
<?xml version="1.0" encoding="UTF-8"?>
<email
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="email.xsd"
>
  <sender>fcsaba.eger@gmail.com</sender>
  <date>2017-03-01</date>
  <recipients>
    <recipient>info@youtube.com</recipient>
    <recipient>info2@youtube.com</recipient>
    <copy>admin@youtube.com</copy>
    <blind-copy>admin@mymail.com</blind-copy>
    <copy>admin2@youtube.com</copy>
  </recipient>
  <subject>Bug report</subject>
  <body>I have found a bug ...</body>
</email>
```

<https://goo.gl/RGqUMg>



# Exercise 3

- Create a schema for the match.xml file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- <!DOCTYPE match SYSTEM "match.dtd"> -->
<match
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="match.xsd"
  league="Barclays Premiership" date="2017-03-06">
  <teams>
    <home goals="2">Chelsea</home>
    <away goals="1">Portsmouth</away>
  </teams>
  <scorers>
    <homeScorer>Drogba</homeScorer>
    <homeScorer>Schevchenko</homeScorer>
    <awayScorer>Benjani</awayScorer>
  </scorers>
  <referee>Dermot Gallagher</referee>
  <attendance>45326</attendance>
</match>
```

# Exercise 3

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <xs:element name="match">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="teams"/>
        <xs:element ref="scorers"/>
        <xs:element ref="referee"/>
        <xs:element ref="attendance"/>
      </xs:sequence>
      <xs:attribute name="date" use="required" type="xs:date"/>
      <xs:attribute name="league" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="teams">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="home"/>
        <xs:element ref="away"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="home">
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:NCName">
          <xs:attribute name="goals" use="required" type="xs:integer"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="away">
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:NCName">
          <xs:attribute name="goals" use="required" type="xs:integer"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="scorers">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" ref="homeScorer"/>
        <xs:element ref="awayScorer"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="homeScorer" type="xs:NCName"/>
  <xs:element name="awayScorer" type="xs:NCName"/>
  <xs:element name="referee" type="xs:string"/>
  <xs:element name="attendance" type="xs:integer"/>
</xs:schema>
```