



# Leuville Objects

EXPERTS EN TECHNOLOGIES JAVA, SOA ET UML,  
DU SERVEUR AU MOBILE

## **Java et Web 2.0**

**Rappels JavaScript, DOM et AJAX**  
**Présentation et utilisation de Dojo**  
**Concepts JSF 2.0 et gestions d'événements**  
**Intégration de Dojo à JSF**

**Leuville Objects**  
3 rue de la Porte de Buc  
F-78000 Versailles  
FRANCE

tel : + 33 (0)1 39 50 2000  
fax: + 33 (0)1 39 50 2015

[www.leuville.com](http://www.leuville.com)  
[contact@leuville.com](mailto:contact@leuville.com)



© Leuville Objects, 2000-2013  
29 rue Georges Clémenceau  
F-91310 Leuville sur Orge  
FRANCE

<http://www.leuville.com>

Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects, est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).

Les marques citées sont des marques commerciales déposées par leurs propriétaires respectifs.

CETTE PUBLICATION EST FOURNIE "EN L'ETAT" SANS GARANTIE D'AUCUNE SORTE, NI EXPRESSE NI IMPLICITE, Y COMPRIS, ET SANS QUE CETTE LISTE NE SOIT LIMITATIVE, DES GARANTIES CONCERNANT LA VALEUR MARCHANDE, L'APTITUDE DES PRODUITS A RÉPONDRE A UNE UTILISATION PARTICULIERE, OU LE FAIT QU'ILS NE SOIENT PAS CONTREFAISANTS DE PRODUITS DE TIERS.



# *Table des matières*

<b>JavaScript .....</b>	<b>1</b>
Présentation.....	1-3
Instructions - Conditions - Boucles .....	1-5
Intégration de JavaScript dans une page HTML .....	1-11
Variables .....	1-13
Evènements.....	1-15
Les objets du navigateur .....	1-17
Les forces et faiblesses de JavaScript.....	1-19
DOM en JavaScript.....	1-21
<b>AJAX.....</b>	<b>25</b>
AJAX .....	2-27
Les apport d'AJAX.....	2-29
<b>HTML - bases .....</b>	<b>33</b>
Présentation.....	3-35
Développement d'une page HTML .....	3-39
Structure de base d'une page HTML.....	3-41
Mise en forme : les couleurs .....	3-43
Mise en forme : police et taille .....	3-45
Mise en forme : centré, gras, .....	3-47
Les paragraphes .....	3-49
Les titres.....	3-51
Les listes .....	3-53
Les séparateurs.....	3-55
Les chemins d'accès .....	3-57
Insérer des images.....	3-59
Les lien hypertextes .....	3-61
Créer des tableaux.....	3-65
<b>HTML - Les formulaires .....</b>	<b>69</b>
Le besoin.....	4-71
Création d'un formulaire HTML .....	4-73
Les différents types d'input .....	4-75
Les zones de saisie.....	4-77
Les listes de choix.....	4-79
<b>XHTML.....</b>	<b>81</b>
Présentation.....	5-83
Conversion d'un document HTML en XHTML.....	5-85
<b>Cascading Style Sheets.....</b>	<b>87</b>
Présentation.....	6-89
Syntaxe de base de CSS.....	6-91



Appliquer une CSS à une page HTML .....	6-93
Utilisation de l'attribut style .....	6-95
CSS interne .....	6-97
CSS externe.....	6-99
Quelques propriétés : couleurs et fonds.....	6-101
Quelques propriétés : polices de caractères .....	6-103
Quelques propriétés : mise en forme de texte.....	6-105
Gestion des liens .....	6-107
Identification et regroupement d'éléments .....	6-109
Structuration d'un document HTML : les éléments span et div .....	6-111
Box model.....	6-115
Quelques propriétés : les bordures.....	6-117
Quelques propriétés : largeur et hauteur .....	6-119
Conclusion .....	6-121
<b>Introduction à XML.....</b>	<b>123</b>
Présentation de XML .....	7-125
Applications de XML .....	7-127
Exemple de document XML.....	7-129
Règles syntaxiques XML.....	7-131
Contrôler la structure d'un document XML .....	7-133
Document Type definition .....	7-135
XMLSchema .....	7-137
Exemple XMLSchema.....	7-139
Les espaces de nommage XMLSchema .....	7-141
Les parseurs XML .....	7-143
Principes DOM .....	7-147
Principes SAX.....	7-149
Technologies XML .....	7-151
eXtensible Stylesheet Language (XSL) .....	7-153
Exemple XSLT .....	7-155
<b>Notions de base XML.....</b>	<b>159</b>
Présentation des langages à balises.....	8-161
Le World Wide Web Consortium (W3C).....	8-165
Le langage XML .....	8-167
Le langage XHTML.....	8-171
<b>Présentation de Dojo .....</b>	<b>173</b>
Qu'est-ce que Dojo ? .....	9-175
Les navigateurs supportés .....	9-177
Un premier exemple Dojo.....	9-189
<b>Utilisation de Dojo.....</b>	<b>195</b>
Notions de base de Dojo .....	10-197
Accès à l'arbre DOM.....	10-207
Gestion des événements.....	10-213



AJAX avec Dojo .....	10-221
<b>JSF 2.0 .....</b>	<b>229</b>
Les fonctionnalités apportées par JSF .....	11-231
Comment ça marche .....	11-233
Présentation de JSF 2.0 .....	11-235
Les nouveautés.....	11-237
Les serveurs d'application .....	11-243
L'intégration de JSF au serveur .....	11-245
Exemple d'application .....	11-247
<b>Cycle de vie d'une requête JSF et Evènements .....</b>	<b>251</b>
Cycle de vie d'une requête JSF.....	12-253
Les différents événements rencontrés .....	12-257
L'événement 'action' .....	12-261
<b>La navigation entre pages.....</b>	<b>267</b>
Définition des règles de navigation .....	13-269
Navigation statique .....	13-271
Navigation dynamique.....	13-273
Exemple de navigation.....	13-275
<b>Les Facelets .....</b>	<b>281</b>
Présentation des facelets .....	14-283
Exemple de page de connexion avec Facelet.....	14-285
Utilisation de balises JSF avec Facelets .....	14-287
Composition de pages avec des templates .....	14-289
<b>Les librairies de balises JSF .....</b>	<b>293</b>
La librairie core.....	15-295
La librairie HTML .....	15-299
<b>Composants standards.....</b>	<b>315</b>
Les composants standards.....	16-317
Les composants standards.....	16-322
<b>Composites components.....</b>	<b>323</b>
Présentation.....	17-325
<b>JSF et les événements.....</b>	<b>331</b>
Cycle de vie des événements .....	18-333
L'événement 'action' .....	18-337
<b>Conversion et validation .....</b>	<b>343</b>
Processus de conversion et de validation.....	19-345
Utilisation de la conversion standard.....	19-347
Gestion des erreurs de conversion .....	19-353
Les valideurs standards .....	19-355
Mise en place d'un convertisseur personnalisé.....	19-357



Mise en place d'un valideur personnalisé .....	19-361
<b>Intégration de Dojo à JSF.....</b>	<b>367</b>
Les différentes approches de Dojo avec JSF < 2.0 .....	20-369
Association des composants JSF et des widgets Dojo côté client (deffered binding) .....	20-371
Construction d'un composant JSF personnalisé, combiné avec un widget Dojo .....	20-373
Conversion d'un composant JSF en widget DOJO par injection côté client (lazy injection) .....	20-375
Dojo avec JSF version 2.0 .....	20-377
Dojo avec JSF version 2.0 .....	20-379
<b>Annexes.....</b>	<b>381</b>
<b>Règles d'écriture d'un document XML.....</b>	<b>383</b>
Structure d'un document XML .....	21-385
Les éléments d'un document XML .....	21-387
Règles d'écriture d'un document XML .....	21-389
Documents 'bien formé' et 'valide' .....	21-391
Les sections CDATA .....	21-393
<b>Validation de documents avec des DTDs.....</b>	<b>395</b>
Document Type definition .....	22-397
Types prédéfinis DTD .....	22-399
Règles d'utilisation des éléments de DTD.....	22-401
Les attributs des éléments de DTD .....	22-403
Déclarer des entités dans une DTD.....	22-405
<b>Valider des documents XML avec des schémas XML .....</b>	<b>407</b>
XMLSchema .....	23-409
Exemple XMLSchema.....	23-411
Les éléments d'un schéma XML .....	23-413
Contrôles sur les types XMLSchema.....	23-415
Types XMLSchema dérivés.....	23-417
Les espaces de nommage XMLSchema .....	23-419





# *Java et Web 2.0*

## *JavaScript*

---

*Version 1.0*

- Présentation
- Syntaxe
- Objets du navigateur
- DOM

(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).



## Présentation

- JavaScript est lié à HTML.
  - Langage interprété par les navigateurs Web.
  - Code encapsulé dans des pages HTML et exécuté coté client.
  - Permet de rendre dynamique les pages affichées par le navigateur.
- JavaScript est un langage de programmation orienté objet.
  - chaque objet possède des attributs et des méthodes.
- JavaScript est un langage sensible à la casse.
- Cependant :
  - JavaScript n'est pas implémenté de la même façon sur chaque navigateur.
  - Il est quasi-impossible de produire du code JavaScript compatible avec tous les navigateurs.
  - Tous les navigateurs ne supportent pas JavaScript.

# Présentation

## Notes

## Instructions - Conditions - Boucles

- Les instructions se terminent par ';'.
- Les instructions JavaScript se trouvent généralement au sein de fonctions.

### Définition d'une fonction

- Utilisation du mot clé function

```
function maFonction() {  
    ...  
}
```

# Instructions - Conditions - Boucles

## Notes

# Instructions - Conditions - Boucles

## Conditions

- Egal : ==
- Différent : !=
- Inférieur ou égal : <=
- Supérieur ou égal : >=
- Inférieur strict : <
- Supérieur strict : >
- ET logique : &&
- OU logique : ||
- Identité : ===
- Non identité : !==
- ET bit à bit : &
- OU bit à bit : |

```
if (var1==var2)
{
    .....
}
else
{
    .....
}
```



# Instructions - Conditions - Boucles

## Notes

# Instructions - Conditions - Boucles

## Boucles

```
for (i=0; i<5; i++)  
{  
    .....  
}
```

```
while (a<b)  
{  
    .....  
}  
  
do  
{  
    .....  
}while (a<b)
```

# Instructions - Conditions - Boucles

## Notes

## Intégration de JavaScript dans une page HTML

- Utilisation de la balise `<script>` dans l'en-tête du fichier

```
<html>
  <head>
    <title></title>
    <script language="javascript" type="text/javascript">
      ...
    </script>
  </head>
  ...
</html>
```

- Référence à un fichier JavaScript externe

```
<html>
  <head>
    <title></title>
    <script language="javascript" type="text/javascript" src="script.js"></script>
  </head>
  ...
</html>
```

# **Intégration de JavaScript dans une page HTML**

## **Notes**

## Variables

- Les variables ne sont pas typées.
- Les variables sont déclarées avec le mot clé **var**.

```
var i, chaine, bool; //Déclaration de 3 variables  
i = 2;  
chaine = "bonjour";  
bool = true;
```

# Variables

## Notes

## Evènements

- Permet de réagir à une action coté client.
- Spécifié en ajoutant un attribut onXxx à certains éléments HTML

```
<body onload="maFonction()">  
<input type="button" onclick="maFonction()">
```

- Peut être basé sur l'usage de pseudo-URL

```
<a href="javascript:alert('Coucou !!')">Mon Lien</a>
```

Evènement	Survient	Evènement	Survient
onload	après le chargement de la page	onmouseup	quand on relâche le bouton de la souris
onunload	lors de la fermeture de la page	onkeydown	quand on enfonce une touche du clavier
onbeforeunload	juste avant la fermeture de la fenêtre	onkeyup	quand on relâche la touche
onclick	lors d'un clic	onsubmit	juste avant l'envoi d'un formulaire
ondblclick	lors d'un double clic	onreset	lors de la réinitialisation d'un formulaire
onmousedown	quand on enfonce le bouton de la souris	onselect	quand le contenu d'un élément est sélectionné



# Evènements

## Notes

## Les objets du navigateur

- `window` : créé lors de l'ouverture du navigateur. Cet objet est implicite.
- `window.navigator` : représente le navigateur (type, version, ...)
- `window.document` : représente le document HTML courant

# Les objets du navigateur

## Notes

# Les forces et faiblesses de JavaScript

## Les avantages

- Simplicité et facilité du langage
- Ergonomie d'une page Web
- Peu de place occupée sur une page, donc rapide.

## Les inconvénients

- Javascript peut-être désactivé donc prévoir un moyen d'accéder à l'information.
- Non compatible avec tous les navigateurs (Lynx) mais la plupart le sont.
- JavaScript n'est pas sécurisé.

# Les forces et faiblesses de JavaScript

## Notes

## DOM en JavaScript

- Modification de la structure d'un document HTML suivant l'API W3C DOM
- window.document et chaque élément HTML sont des noeuds (Node)
- Chaque Node possède des propriétés et des méthodes

Propriétés	Commentaires
childNodes	noeuds enfants
firstChild	premier noeud enfant
lastChild	dernier noeud enfant
nextSibling	prochain noeud d'un type (noeud de même niveau)
parentNode	noeud parent
previousSibling	noeud précédent d'un type (noeud de même niveau)
nodeName	nom du noeud
nodeValue	valeur / contenu du noeud
nodeType	type du noeud (cf. ci-dessous)

### Types de noeuds :

1 - Noeud élément  
2 - Noeud attribut  
3 - Noeud texte  
4 - Noeud pour CDATA  
5 - Noeud pour référence d'entité  
6 - Noeud pour entité

7 - Noeud pour instruction de traitement  
8 - Noeud pour commentaire  
9 - Noeud document  
10 - Noeud type de document  
11 - Noeud de fragment de document  
12 - Noeud pour notation

# DOM en JavaScript

## Notes

## DOM en JavaScript

Méthodes	Commentaires
createElement()	Méthode pour créer un nouvel élément HTML dans le document (div, p, span, a, form, input, etc...).
createTextNode()	Méthode pour créer un nœud texte.
appendChild()	Pour ajouter l'élément créé dans le document. L'élément sera ajouté comme étant le dernier nœud enfant d'un élément parent.
insertBefore()	Pour ajouter l'élément créé avant un autre nœud.
removeChild()	Pour supprimer un nœud.



# DOM en JavaScript

## Notes

# *Java et Web 2.0*

## *AJAX*

---

*Version 1.0*

- Qu'est ce que AJAX ?
- Les apports d'AJAX
- Frameworks et outils d'AJAX

(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).



# AJAX

## Définition

- **Asynchronous JavaScript and XML**
- AJAX n'est pas une technologie
- Terme qui évoque l'utilisation conjointe d'un ensemble de technologies libres couramment utilisées sur le Web
  - **HTML** (ou **XHTML**) pour la structure sémantique des informations ;
  - **CSS** pour la présentation des informations ;
  - **DOM** et **JavaScript** pour afficher et interagir dynamiquement avec l'information présentée ;
  - l'objet **XMLHttpRequest** pour échanger et manipuler les données de manière asynchrone avec le serveur Web ;
  - **XML**.
- En alternative au format XML, les applications AJAX peuvent utiliser les fichiers texte ou **JSON**.

# AJAX

## Notes

## Les apports d'AJAX

- Navigation réactive car une page peut être modifiée localement
- Enchaînement des requêtes AJAX en parallèle
- Meilleures performances
- Aucun plugin requis
- Compatibilité avec la plupart des navigateurs couramment utilisés
- Ergonomie améliorée

# Les apports d'AJAX

## Notes

Le risque avec AJAX est de multiplier le code Javascript qui peut-être fastidieux et difficile à maintenir. Ou encore de l'utiliser avec plusieurs toolkits simultanés qui peuvent engendrer des problèmes fonctionnels.

# AJAX

## Frameworks

- **Direct Web Remoting** (DWR) : permet l'appel de méthodes Java coté serveur en JavaScript
- **Echo** : framework Java de développement d'application riches sur internet
- **Google Web Toolkit** : framework Java de développement d'applications riches coté client (JavaScript)
- **AjaxAC** : Framework Ajax en PHP
- **XHRConnection** : classe JavaScript encapsulant diverses fonctionnalités d'appel asynchrône
- **HTML\_AJAX** : Framework Ajax pour PHP
- **xajax** : Framework Ajax pour PHP
- **Dojo toolkit** : Framework JavaScript facilitant le développement d'applications AJAX
- **Yahoo UI** : Bibliothèque javascript permettant de créer des applications AJAX
- **ASP .NET Ajax** : Framework Microsoft pour implémenter AJAX en .NET
- **SAJAX** : Framework Ajax pour PHP
- ...



# AJAX

## Notes

# *Java et Web 2.0*

## *HTML - bases*

---

*Version 1.0*

- Présentation
- Développement d'une page HTML
- Éléments de mise en forme

(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).



## Présentation

### HTML : HyperText Markup Language

- Langage issu du langage SGML.
- Standard du W3C
- Pas un langage de programmation.
- Ensemble de balises permettant de mettre en forme du texte et des images.
- Non extensible.
- Navigation entre pages HTML grace aux liens hypertextes.

### Page HTML

- Composée de balises HTML
- Interprétée par un navigateur
- Juste de l'affichage de données (possibilité de dynamisme avec JavaScript)

# Présentation

## Notes

# Présentation

## Balise

- "instruction" de mise en page comprise entre crochets `<...>`
  - `<html>`, `<table>`, `<font>`, ...
- Possède un nom et parfois des attributs
- Peut posséder une balise de fin (HTML moins strict que XML)
  - `<html>...</html>`
- nom insensible à la casse (HTML moins strict que XML)

# Présentation

## Notes

## Développement d'une page HTML

### Ce qui est nécessaire :

- Un éditeur HTML (simple éditeur de texte, éditeur WYSIWYG, ...)
  - Notepad
  - Dreamweaver
  - ...
- Un navigateur Web
  - Internet Explorer
  - Firefox
  - Opera
  - ...
- Un accès à Internet n'est pas nécessaire

---

L'affichage de la page peut différer d'un navigateur à l'autre.

---

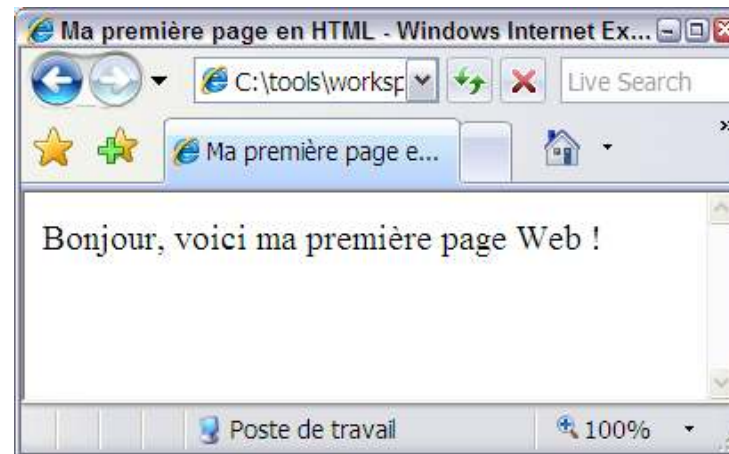


# Développement d'une page HTML

## Notes

## Structure de base d'une page HTML

```
<html>
  <head>
    <title>Ma première page en HTML</title>
  </head>
  <body>
    Bonjour, voici ma première page Web !
  </body>
</html>
```



## Structure de base d'une page HTML

- `<html>` et `</html>` : indique au navigateur le début et la fin de la page HTML.
- `<head>` et `</head>` : entete de la page HTML contenant des meta-information sur la page (titre, liens vers d'autres ressources, ...).
- `<title>` et `</title>` : titre de la page tel qu'il apparait dans la fenêtre du navigateur.
- `<body>` et `</body>` : corps de la page, c'est dans le corps que se trouvent les données à afficher.

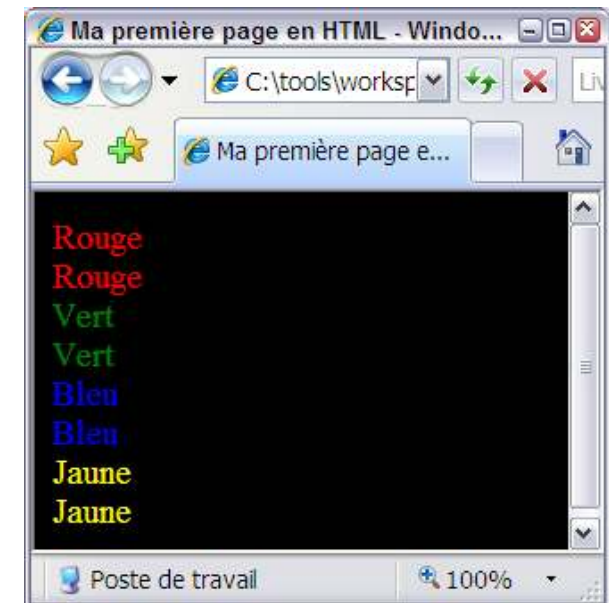
## Mise en forme : les couleurs

```
<html>
  <head>
    <title>Ma première page en HTML</title>
  </head>
  <body bgcolor="#000000">
    <FONT color="red">Rouge</FONT> <BR>
    <FONT color="#FF0000">Rouge</FONT> <BR>
    <FONT color="green">Vert</FONT> <BR>
    <FONT color="#008000">Vert</FONT> <BR>
    <FONT color="blue">Bleu</FONT> <BR>
    <FONT color="#0000FF">Bleu</FONT> <BR>
    <FONT color="yellow">Jaune</FONT> <BR>
    <FONT color="#FFFF00">Jaune</FONT> <BR>
  </body>
</html>
```

---

Donné à titre d'illustration, la balise FONT est dépréciée depuis HTML 4

---



## Mise en forme : les couleurs

### Les nouvelles balises / nouveaux attributs

- `<font>` et `</font>`: indication de police de caractères de mise en forme de texte (dépréciée, utiliser CSS à la place)
- `<br>` ou `<br/>` ou `<br></br>` : saut de ligne
- `bgcolor` : attribut optionnel de `body` permettant de préciser une couleur de fond à la page

### Déclaration d'une couleur

- Deux possibilités :
  - nom de la couleur
  - code RVB exprimé en Hexadécimal

`#FF0000` = rouge (65000 couleurs)

`#F00` = rouge (256 couleurs)

## Mise en forme : police et taille

```
<HTML>
<HEAD><TITLE>Taille et police du textes</TITLE></HEAD>
<BODY>
  <FONT size=7>Taille 7</FONT> <BR>
  <FONT size=6>Taille 6</FONT> <BR>
  <FONT size=5>Taille 5</FONT> <BR>
  <FONT size=4>Taille 4</FONT> <BR>
  <FONT size=3>Taille 3 (par défaut)</FONT> <BR>
  <FONT size=2>Taille 2</FONT> <BR>
  <FONT size=1>Taille 1</FONT> <BR><BR>

  <FONT size="+4">Taille +4</FONT> <BR>
  <FONT size="+3">Taille +3</FONT> <BR>
  <FONT size="+2">Taille +2</FONT> <BR>
  <FONT size="+1">Taille +1</FONT> <BR>
  Taille par défaut ( => 3 ) <BR>
  <FONT size="-1">Taille -1</FONT> <BR>
  <FONT size="-2">Taille -2</FONT> <BR>

  <FONT size=4 face="Verdana">Taille 4 en Verdana</FONT> <BR>
  <FONT size=2 face="Comic sans MS">Taille 3 en Comic sans MS</FONT> <BR>
  <FONT face="Arial, Times New Roman" color="#336699">Taille normal en Arial si la
police existe sinon en Times New Roman en couleur #336699</FONT> <BR>
</BODY>
</HTML>
```



## Mise en forme : police et taille

### Attributs de FONT

- size : taille de la police
  - absolue (1 à 7), par défaut : 3
  - relative à la taille par défaut
- face : choix de la police

---

La police de caractères doit être connue du navigateur du client

---

## Mise en forme : centré, gras, ...

```
...  
<BODY>  
<B>texte en gras</B> <BR>  
<I>texte en italique</I> <BR>  
<U>texte souligné</U> <BR>  
<CENTER>texte centre</CENTER> <BR>  
<B><CENTER>texte centré en gras</CENTER></B><BR>  
<B><CENTER>  
<FONT color="red" size=2>texte en rouge, gras, centré de taille 2 </FONT>  
</CENTER></B> <BR>  
</BODY>  
</HTML>
```





## Mise en forme : centré, gras, ...

### Nouvelles balises

- `<b>` et `</b>` : texte en gras
- `<i>` et `</i>` : texte en italique
- `<u>` et `</u>` : texte souligné
- `<center>` et `</center>` : centre le texte par rapport à l'écran

# Les paragraphes

<BODY>

Paragraphes : <BR>

Vous pouvez former des paragraphes si vous le souhaitez ce qui vous permettra d'aligner du texte soit à gauche (alignement par défaut) soit à droite, au centre ou encore en justifié

`<P align="right">texte aligné à droite</P>`

`<P align="center">texte aligné au centre</P>`

`<P align="left">texte aligné à gauche</P>`

```
<P align="justify">texte justifié : blablablabla-  
blablablablabla blablablablablablablablablabla  
blablablablablablablablablabla blablablablablabla-  
blablablablabla blablablablablablablablablabla bla-  
blablablablablablablablablabla  
blablablablablablablablablabla blablablablablabla-  
blablablablabla</P>
```

```
retrait de texte : <BR>
```

<BLOCKQUOTE>votre texte</BLOCKQUOTE> <BR>

&lt;/BODY&gt;



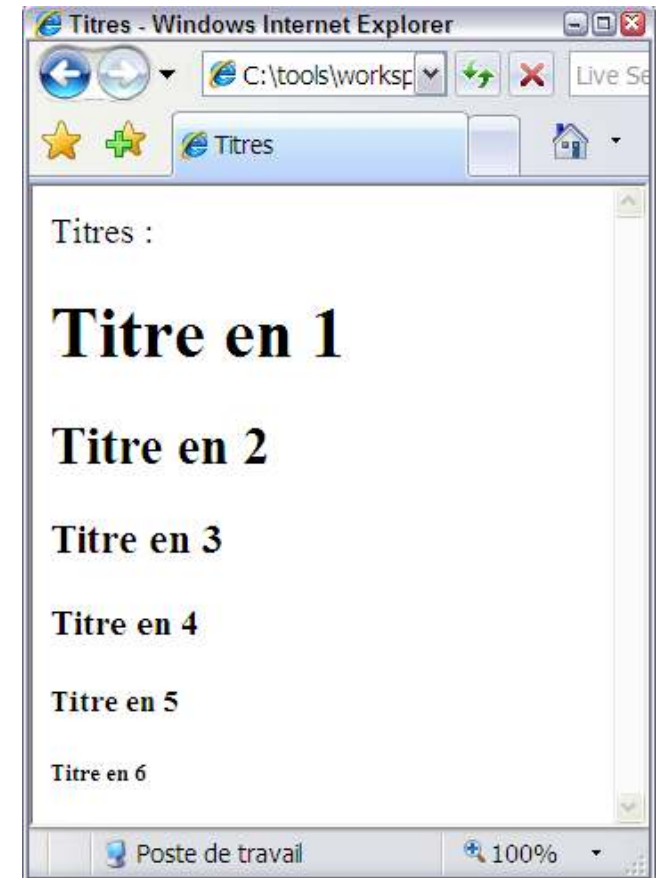
# Les paragraphes

## Nouvelles balises

- `<p>` et `</p>` : permet de délimiter un paragraphe. L'attribut `align`, optionnel, permet de préciser l'alignement du texte
- `<blockquote>` et `</blockquote>` : permet de décaler un bloc de texte par rapport au bord gauche

## Les titres

```
<BODY>
Titres : <BR>
<H1>Titre en 1</H1>
<H2>Titre en 2</H2>
<H3>Titre en 3</H3>
<H4>Titre en 4</H4>
<H5>Titre en 5</H5>
<H6>Titre en 6</H6>
<!-- il n'existe que 6 types de titres -->
</BODY>
```



# Les titres

## Nouvelles balises

- `<Hx>` et `</Hx>` (x de 1 à 6) : titre de niveau x
- `<!-- ... -->` : commentaire HTML

## Les listes

Chapitre 1 : type disque plein (par défaut)

```
<UL type="disc">  
<LI>Page 1</LI>  
<LI>Page 2</LI>  
</UL>
```

Chapitre 2 : type disque vide

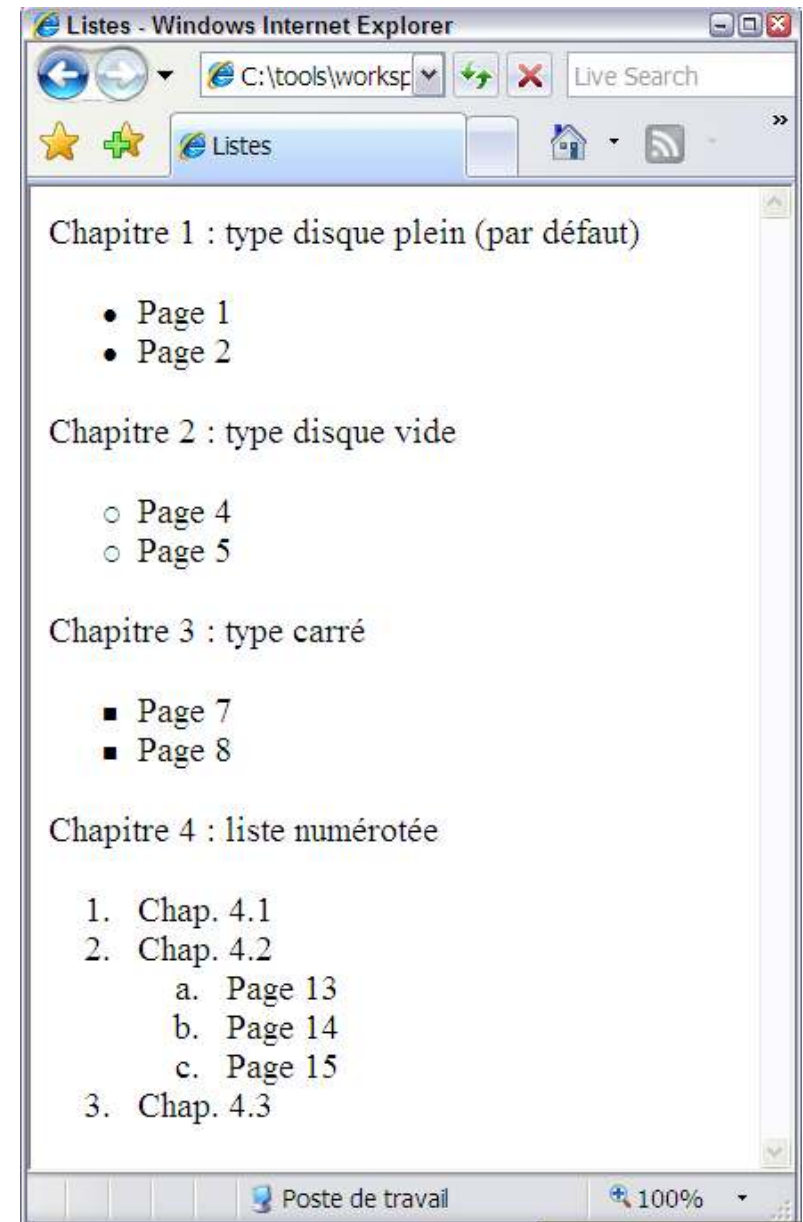
```
<UL type="circle">  
<LI>Page 4</LI>  
<LI>Page 5</LI>  
</UL>
```

Chapitre 3 : type carré

```
<UL type="square">  
<LI>Page 7</LI>  
<LI>Page 8</LI>  
</UL>
```

Chapitre 4 : liste numérotée

```
<OL>  
<LI>Chap. 4.1</LI>  
<LI>Chap. 4.2  
  <OL type="a">  
    <LI>Page 13</LI>  
    <LI>Page 14</LI>  
    <LI>Page 15</LI>  
  </OL>  
</OL>
```



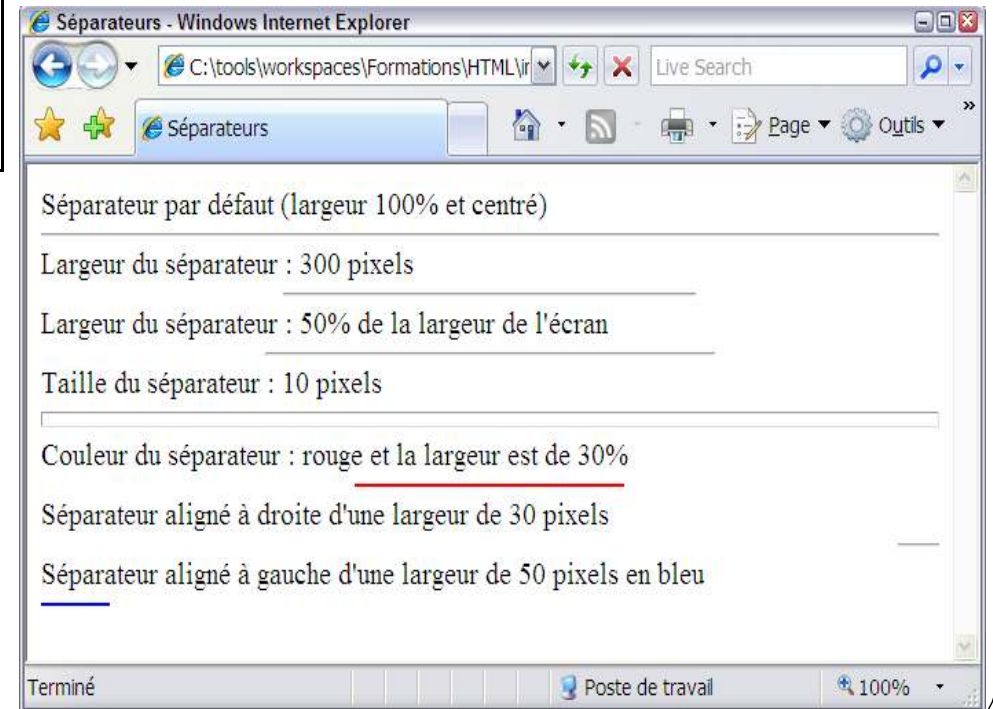
# Les listes

## Nouvelles balises

- `<ul>` et `</ul>` : délimite une liste à puces
- `<ol>` et `</ol>` : délimite une liste numérotée
- `<li>` : élément d'une liste

## Les séparateurs

```
<BODY>
Séparateur par défaut (largeur 100% et centré)
<HR>
Largeur du séparateur : 300 pixels
<HR width=300>
Largeur du séparateur : 50% de la largeur de l'écran
<HR width="50%">
Taille du séparateur : 10 pixels
<HR size=10>
Couleur du séparateur : rouge et la largeur est de 30%
<HR color="#FF0000" width="30%">
Séparateur aligné à droite d'une largeur de 30 pixels
<HR align="right" width=30>
Séparateur aligné à gauche d'une largeur de 50 pixels en bleu
<HR align="left" width=50 color="#0000FF">
</BODY>
```





# Les séparateurs

## Nouvelles balises

- `<hr>` : permet de placer un trait horizontal séparateur

## Les chemins d'accès

- Permettent de référencer une autre ressource (autre page HTML, image, feuille de style, ...)
- Peuvent être de deux types :
  - relatifs : la ressource se trouve sur le même serveur que la page courante. Le chemin indique comment accéder à la ressource en partant de la localisation de la page courante.

**logo.gif** : chemin d'accès à une image se trouvant dans le même répertoire que la page courante.

**images/logo.gif** : chemin d'accès à une image se trouvant dans le répertoire "images", sous-répertoire du répertoire dans lequel se trouve la page courante.

- absolus : la ressource se trouve sur un autre serveur. Le chemin indique l'URL complète de la ressource.

**http://www.leuville.com/images/logo.gif** : URL (chemin d'accès absolu) d'une image

# Les chemins d'accès

## Notes

## Insérer des images

```
<BODY>
```

Une première image :

```
<br><br>
```

Une image avec bordure :

```
<br><br>
```

Une image manquante :

```
<br><br>
```

Une image redimensionnée :

```
<br><br>
</BODY>
```



# Insérer des images

## Nouvelles balises / Nouveaux attributs

- `<img>` : insère une image.
  - `src` : attribut obligatoire, indique le chemin d'accès à l'image
  - `border` : taille de la bordure de l'image
  - `width` et `height` : dimensions de l'image
  - `alt` : texte alternatif à afficher si l'image est introuvable ou si le navigateur ne peut pas afficher les images

## Les lien hypertextes

- Permettent de naviguer sur le site
- 3 types de lien :
  - Interne à un site (chemin d'accès relatif) : permet de passer d'une page HTML à une autre au sein du meme serveur.
  - Interne à une page (ancree) : permet de naviguer au sein d'une page HTML
  - Externe (chemin d'accès absolu) : permet de rediriger vers un autre site
- Peut être associé à :
  - du texte
  - des images

# Les liens hypertextes

## Notes

# Les liens hypertextes

```
<BODY>
  <a name="ancrer1"></a>
  <h2>Liens internes au site</h2>
  <a href="page2.html">Lien vers la page 2</a><br>
  <a href="admin/page.html">Lien vers une autre page</a><br>
  <a href=" ../parent.html">Lien vers une page un niveau au dessus</a>
  <h2>Liens externes</h2>
  <a href="http://www.leuville.com">Leuville Objects</a>
  <a href="http://www.leuville.com">
  <h2>Liens internes à la page</h2>
  <a href="#ancrer1">Retour au sommet</a>
  <h2>Liens vers une ancre d'une autre page</h2>
  <a href="page2.html#uneAncre">Lien vers la page 2</a>
</BODY>
```





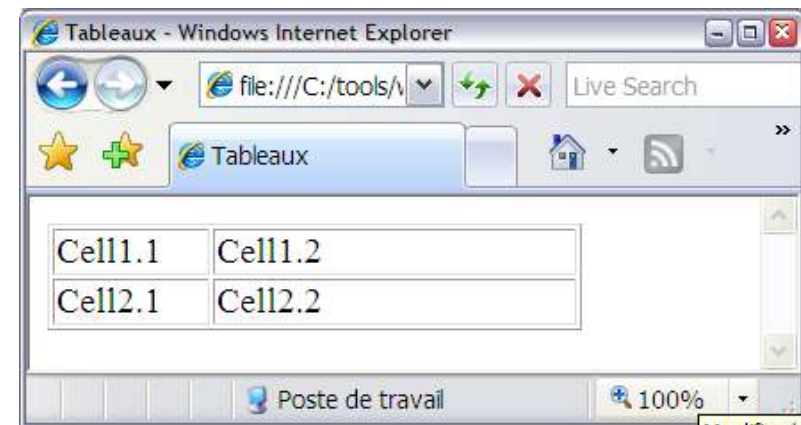
# Les liens hypertextes

## Nouvelles balises / nouveaux attributs

- `<a>` et `</a>` : permet de délimiter un lien hypertexte
  - `href` : attribut indiquant le chemin d'accès de la ressource destination
  - `name` : permet de créer une ancre

## Créer des tableaux

```
<BODY>
  <table width="300px" border="1">
    <tr>
      <td width="30%">Cell1.1</td>
      <td width="70%">Cell1.2</td>
    </tr>
    <tr>
      <td>Cell2.1</td>
      <td>Cell2.2</td>
    </tr>
  </table>
</BODY>
```



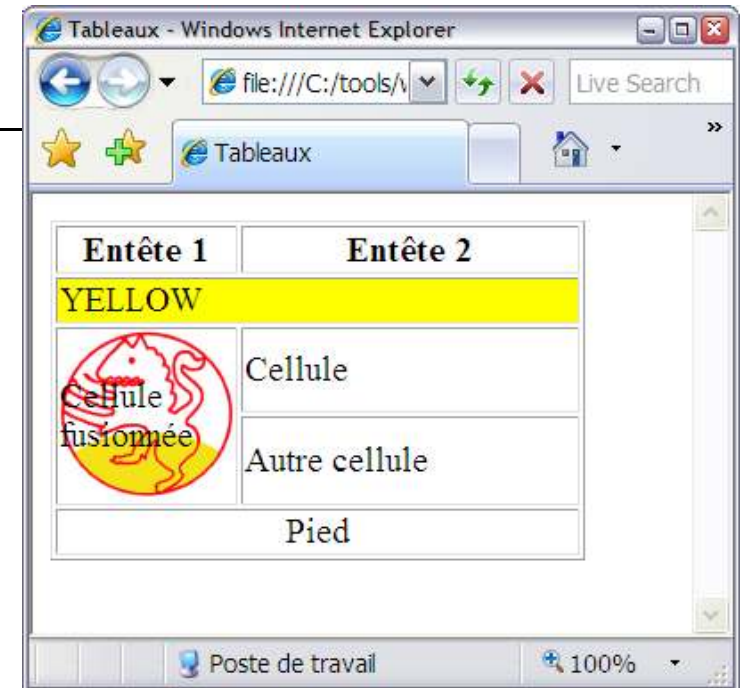
# Créer des tableaux

## Nouvelles balises

- `<table>` et `</table>` : délimite un tableau
  - `width` : largeur du tableau (fixe ou pourcentage)
  - `border` : taille de la bordure (invisible par défaut)
- `<tr>` et `</tr>` : délimite une ligne du tableau
  - `width` : largeur
  - `height` : hauteur
- `<td>` et `</td>` : délimite une cellule

## Créer des tableaux

```
<table width="300px" border="1">
  <thead>
    <tr>
      <th width="35%">Entête 1</th>
      <th>Entête 2</th>
    </tr>
  </thead>
  <tfoot>
    <tr><td colspan="2" align="center">Pied</td></tr>
  </tfoot>
  <tr><td colspan="2" bgcolor="yellow">YELLOW</td></tr>
  <tr>
    <td rowspan="2" background="images/logo-lion.png" height="100px">Cellule fusionnée</td>
    <td>Cellule</td>
  </tr>
  <tr><td>Autre cellule</td></tr>
</table>
```



# Créer des tableaux

## Nouvelles balises / nouveaux attributs

- `<thead>` et `</thead>` : délimite l'en-tête du tableau (sera toujours positionné en haut)
- `<th>` et `</th>` : délimite des cellules d'en-tête
- `<tfoot>` et `</tfoot>` : délimite le pied du tableau (sera toujours positionné en bas)
- `colspan` : fusion de cellules sur une ligne
- `rowspan` : fusion de cellules sur une colonne
- `background` / `bgcolor` : gestion du fond

# *Java et Web 2.0*

## *HTML - Les formulaires*

---

*Version 1.0*

- Besoin
- Les composants d'un formulaire

(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).



## Le besoin

- Créer de l'interactivité avec le visiteur du site
- Proposer diverses façons de saisir des informations
- Envoyer ces informations vers une ressource d'un site web

Les données d'un formulaire HTML ne peuvent être exploitées par une simple page HTML.

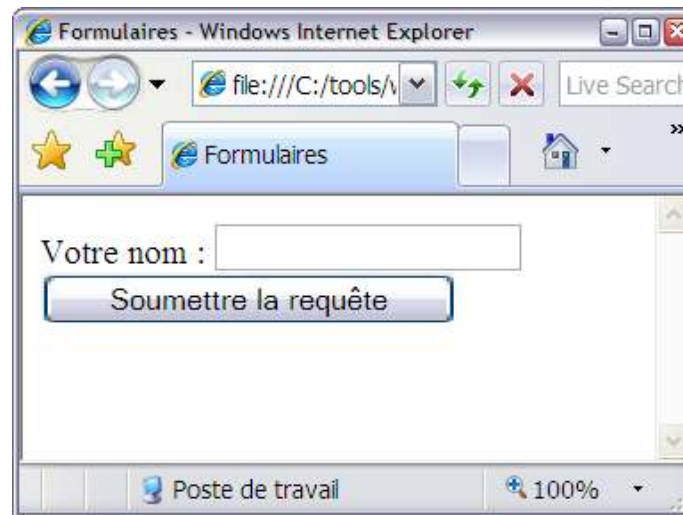


# Le besoin

## Notes

## Création d'un formulaire HTML

```
<BODY>  
  <form name="form1" action="page2.html" method="POST">  
    Votre nom : <input type="text" name="nom"><br>  
    <input type="submit">  
  </form>  
</BODY>
```

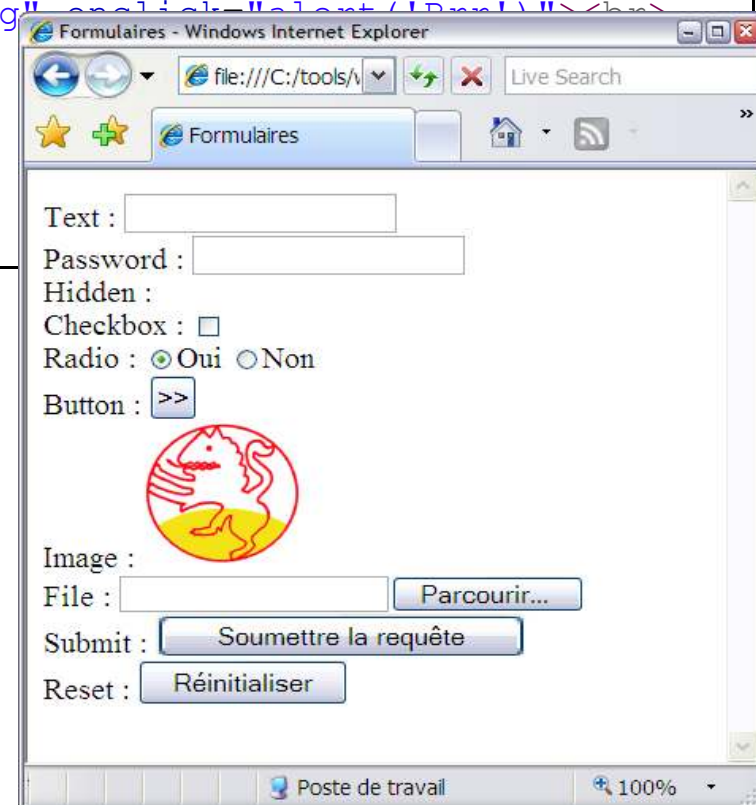


## Création d'un formulaire HTML

- `<form>` et `</form>` : délimite un formulaire
  - `action` : ressource à laquelle transmettre les informations du formulaire
  - `method` : méthode HTTP utilisée pour transmettre les données du formulaire
- 1 GET : les données sont incluses dans l'URL appelée
- 1 POST : les données sont incluses dans le corps de la requête HTTP
- `<input>` : donnée du formulaire
  - `type=text` : champs de type texte
  - `type=submit` : bouton de soumission du formulaire

## Les différents types d'input

```
<form name="form1" action="page2.html" method="POST" enctype="multipart/form-data">
  Text : <input type="text" name="ch1"><br>
  Password : <input type="password" name="passw"><br>
  Hidden : <input type="hidden" value="valeur" name="hid"><br>
  Checkbox : <input type="checkbox" name="check"><br>
  Radio : <input type="radio" name="choix" value = "oui" checked>Oui
          <input type="radio" name="choix" value = "non">Non<br>
  Button : <input type="button" value=">>" onclick="alert('Zzz')"><br>
  Image : <input type="image" src="images/logo-lion.png" onclick="alert('Drool')"><br>
  File : <input type="file" name="fichier"><br>
  Submit : <input type="submit"><br>
  Reset : <input type="reset"><br>
</form>
```

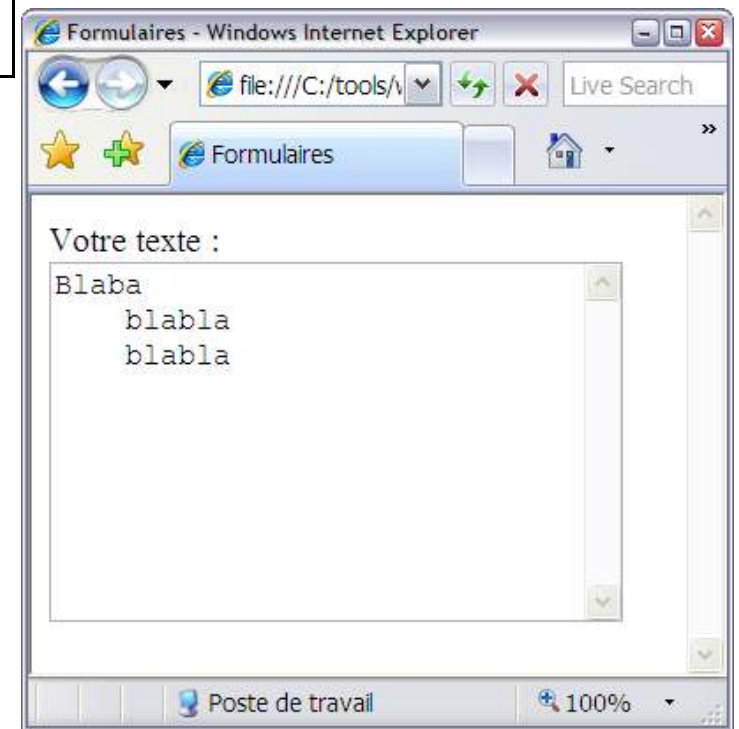


## Les différents type d'Input

- text : champs de saisi de texte
- password : champs de saisi de mot de passe
- hidden : champs caché
- checkbox : case à cocher
- radio : boutons radio
- button : bouton
- image : image
- file : upload de fichier (enctype="multipart/form-data" obligatoire)
- submit : soumission du formulaire
- reset : réinitialisation du formulaire
- Depuis HTML 5 il existe de nouveaux types: color, date, datetime, datetime-local, month, week, time, email, number, range, search, tel, and url

## Les zones de saisie

```
<BODY>
  <form name="form1" action="page2.html" method="POST">
    Votre texte :<br>
    <textarea rows="10" cols="30" name="text">Blaba
    blabla
    blabla</textarea>
  </form>
</BODY>
```

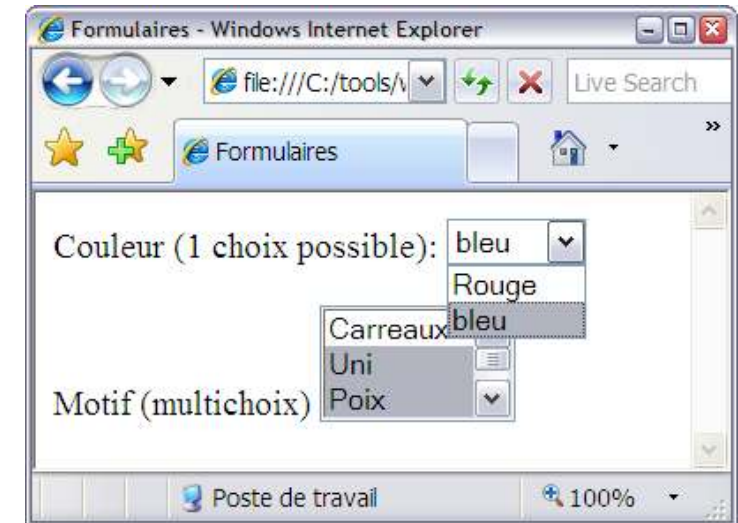


## Les zones de saisie

### Notes

## Les listes de choix

```
<form name="form1" action="page2.html" method="POST">
  Couleur (1 choix possible):
  <select name="couleur">
    <option value="red">Rouge</option>
    <option value="blue" selected>bleu</option>
  </select>
  <br><br>
  Motif (multichoice)
  <select name="motif" multiple size="3">
    <option value="m1">Carreaux</option>
    <option value="m2" selected>Uni</option>
    <option value="m3" selected>Poix</option>
    <option value="m4">Rayures H</option>
    <option value="m5">Rayures V</option>
  </select>
</form>
```





# Les listes de choix

## Notes

# ***Java et Web 2.0***

## ***XHTML***

---

*Version 1.0*

- Présentation
- Conversion de HTML en XHTML

(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).



## Présentation

- Standard W3C.
- XHTML 1.0 évolution de HTML 4 en application de XML 1.0
- Aucune nouvelle fonctionnalité
- Permet de traiter les pages HTML comme des documents XML
- Simplifie la manipulation dynamique des éléments HTML (API DOM en ECMAScript + AJAX)

# Présentation

## Notes

## Conversion d'un document HTML en XHTML

- Les règles syntaxiques du XML doivent être respectées (XML bien formé)
- Le document XHTML doit référencer sa DTD :

```
XHTML 1.0 Strict
    <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
XHTML 1.0 Transitional
    <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
XHTML 1.0 Frameset
    <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
XHTML 1.1
    <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
    "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
```

- Les espaces de noms XML doivent être déclarés :

```
<html xmlns="http://www.w3.org/1999/xhtml">
```

# Conversion d'un document HTML en XHTML

## Notes

# *Java et Web 2.0*

## *Cascading Style Sheets*

---

*Version 1.0*

- Présentation
- Syntaxe
- Appliquer une CSS à une ou plusieurs pages HTML

(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).





## Présentation

- Cascading Style Sheets
- Standard W3C
- HTML permet de structurer des données, CSS permet de formater les données structurées
- Avantages :
  - Séparation données / présentation
  - Centralisation des informations de présentations
  - Facilite la définition et la mise en oeuvre d'une charte graphique
  - Permet d'associer plusieurs look and feel à un même contenu (personnalisation)

# Présentation

## Notes

## Syntaxe de base de CSS

```
sélecteur {propriété : valeur;}
```

- **Sélecteur** : élément(s) HTML sur le(s)quel(s) la propriétés doit s'appliquer.
- **Propriété** : information de présentation.
- **Valeur** : valeur de la propriété

exemple :

```
body {background-color : #FF0000;}
```

# Syntaxe de base de CSS

## Notes

## Appliquer une CSS à une page HTML

- 3 possibilités :
  - Utilisation de l'attribut 'style'
  - CSS interne
  - CSS externe
- Les 3 possibilités sont cumulables, des règles de priorités s'appliquent alors.
- La CSS externe est la seule solution permettant une mise en commun des informations de présentation à l'ensemble des pages d'un site Web.

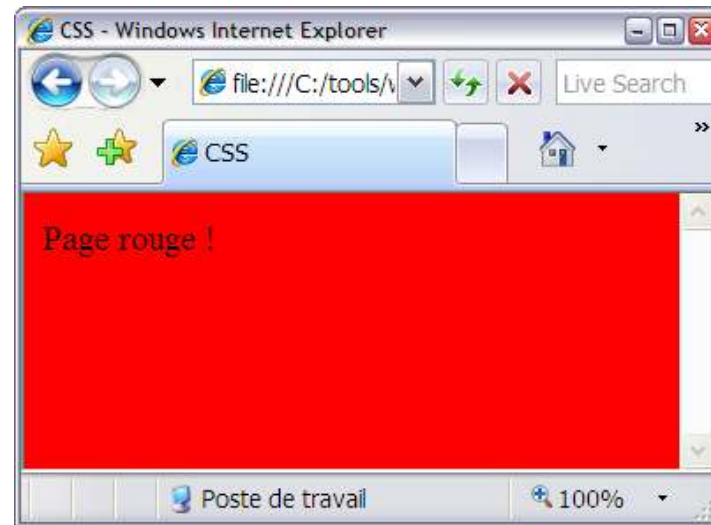
# **Appliquer une CSS à une page HTML**

## **Notes**

## Utilisation de l'attribut style

- Utilisable sur n'importe quel élément HTML.
- Permet de préciser des propriétés de style CSS.

```
<body style="background-color: #FF0000">  
    Page rouge !  
</body>
```





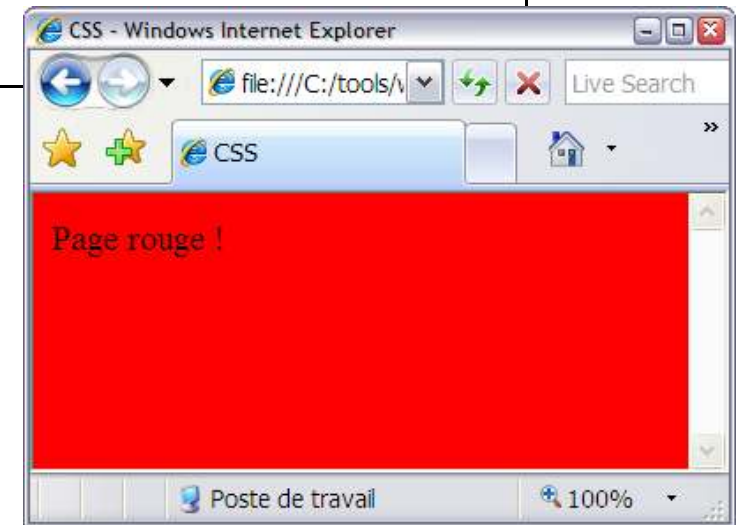
## Utilisation de l'attribut style

### Notes

## CSS interne

- Utilisation de l'élément 'style' dans l'en-tête HTML

```
<html>
  <head>
    <title>CSS</title>
    <style type="text/css">
      body {background-color: #FF0000}
    </style>
  </head>
  <body>
    Page rouge !
  </body>
</html>
```



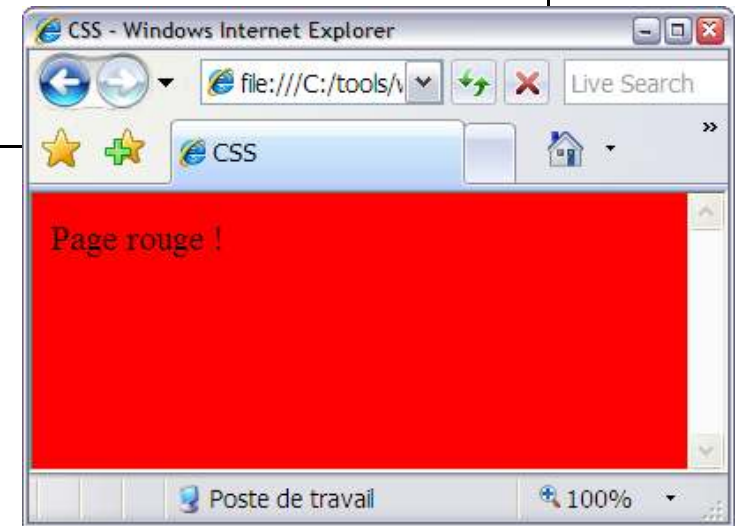
# CSS interne

## Notes

## CSS externe

- Méthode recommandée
- La feuille de style est un fichier à part
- Les documents HTML référencent la feuille de style externe par un élément 'link'

```
<html>
  <head>
    <title>CSS</title>
    <link rel="stylesheet" type="text/css" href="style.css">
  </head>
  <body>
    Page rouge !
  </body>
</html>
```



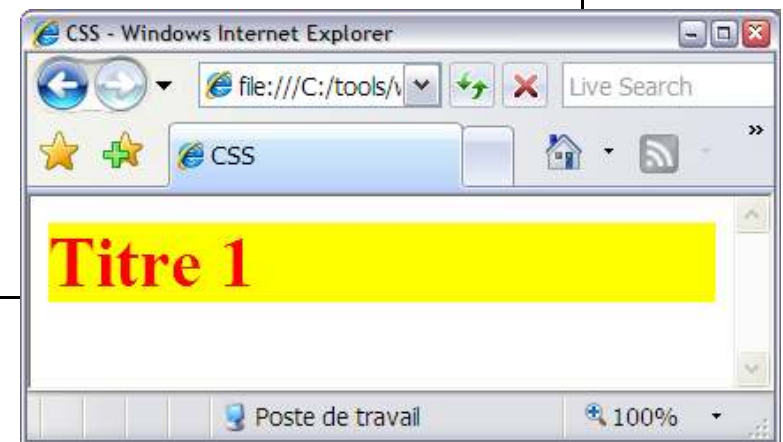
# CSS externe

## Notes

## Quelques propriétés : couleurs et fonds

- color : Couleur du texte
- background-color : Couleur de fond
- background-image : Image de fond
- background-repeat : Répétition de l'image de fond
- background-attachment : Scrolling de l'image de fond
- background-position : Position de l'image de fond
- background : Raccourci permettant de configurer le fond

```
...  
    <style type="text/css">  
        h1 {color:red; background-color:yellow}  
    </style>  
</head>  
<body>  
    <h1>Titre 1</h1>  
</body>  
</html>
```



## **Quelques propriétés : couleurs et fonds**

### **Notes**

## Quelques propriétés : polices de caractères

- font-family : Police de caractères
- font-style : Style (italic, ...)
- font-variant : Normal ou small-caps
- font-weight : Mise en gras
- font-size : Taille de la police
- font : Raccourci permettant de configurer la font

```
...  
<style type="text/css">  
  h1 {  
    font-family: vivaldi;  
    font-weight: bold;  
    font-style: italic;  
    font-size: 5em  
  }  
</style>  
</head>  
<body>  
  <h1>Titre 1</h1>  
</body>
```





## Quelques propriétés : polices de caractères

### Notes

## Quelques propriétés : mise en forme de texte

- text-indent : Indentation
- text-align : Alignement du texte
- text-decoration : Souligné, barré,...
- letter-spacing : Espacement des lettres
- text-transform : Application de transformation

```
...  
<style type="text/css">  
  h1 {  
    text-align: center;  
    text-decoration: underline;  
    text-transform: uppercase;  
  }  
</style>  
</head>  
<body>  
  <h1>Titre 1</h1>  
</body>  
</html>
```



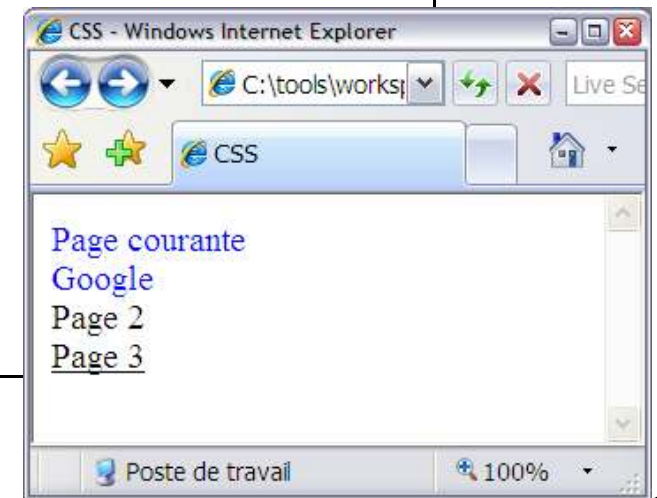
## **Quelques propriétés : mise en forme de texte**

### **Notes**

## Gestion des liens

- Utilisation de pseudo-classes associées à l'élément 'a' :
  - link : lien standard
  - visited : lien dont la cible a déjà été visitée
  - active : lien cliqué
  - hover : lien survolé par la souris

```
...  
<style type="text/css">  
  a:link {color:black; text-decoration: none}  
  a:visited {color:blue; text-decoration: none}  
  a:active {color:red; text-decoration: none}  
  a:hover {text-decoration:underline}  
</style>  
</head>  
<body>  
  <a href="index.html">Page courante</a><br />  
  <a href="http://www.google.fr">Google</a><br />  
  <a href="page2.html">Page 2</a><br />  
  <a href="page3.html">Page 3</a>
```



# Gestion des liens

## Notes

## Identification et regroupement d'éléments

- Utilisation de deux attributs s'appliquant à tout élément HTML :
  - id** : attribut permettant d'associer un identifiant unique à un élément
  - class** : attribut permettant d'associer une catégorie à un élément

```
...
<style type="text/css">
    #tableau {border: solid black 1px}
    #tableau th {background-color: silver;}
    #tableau tr.ligne-paire {background-color: yellow}
    #tableau tr.ligne-impair {font-style: italic}
</style>
</head>
<body>
    <table id="tableau">
        <tr><th>Titre</th><th>Auteur</th></tr>
        <tr class="ligne-impair">
            <td>Les Misérables</td><td>Victor Hugo</td>
        </tr>
        <tr class="ligne-paire">
            <td>Germinal</td><td>Emile Zola</td>
        </tr>
    </table>
```



## Identification et regroupement d'éléments

- La valeur d'un id est unique dans le document HTML
- Plusieurs éléments peuvent partager la même classe
- Le sélecteur d'id est '#'
- Le sélecteur de classe est '.'

## Structuration d'un document HTML : les éléments span et div

- `<span>...</span>` : permet de déclarer du contenu inline
- `<div>...</div>` : permet de déclarer un bloc d'élément
- Utilisés conjointement avec 'id' et 'class', ils forment la base de la définition de chartes graphiques.

```
<body>
  <div id="entete">
    <h1>Mon application</h1>
  </div>
  <div id="menu">
    <h2>Menu</h2>
    <ul>
      <li>Choix 1</li>
      <li>Choix 2</li>
    </ul>
  </div>
  <div id="contenu">
    <h2>Titre de la page</h2>
    blablabla
  </div>
  <div id="pied-de-page">
    &copy;Leuville Objects - 2008
  </div>
```



# Structuration d'un document HTML : les éléments span et div

## Notes

## Structuration d'un document HTML : les éléments span et div

- extrait de la CSS :

```
#entete {position:absolute; top:0px; left:0px; right:0px; height:
#entete h1 {font-family: vivaldi}

#menu {position:relative; top:55px; left:-10px; height:250px; wi
#menu h2 {background-color: black; color:white;width: 100%; text-
#menu ul{padding:10; margin:10}
#menu li {color:blue}

#contenu {position:absolute; left:150px; top:70px; border:solid b
#contenu h2 {width:100%; color:red; text-align: center}

#pied-de-page {position:absolute; left:0px; top:320px; width:500p
```

- Page obtenue à l'écran :

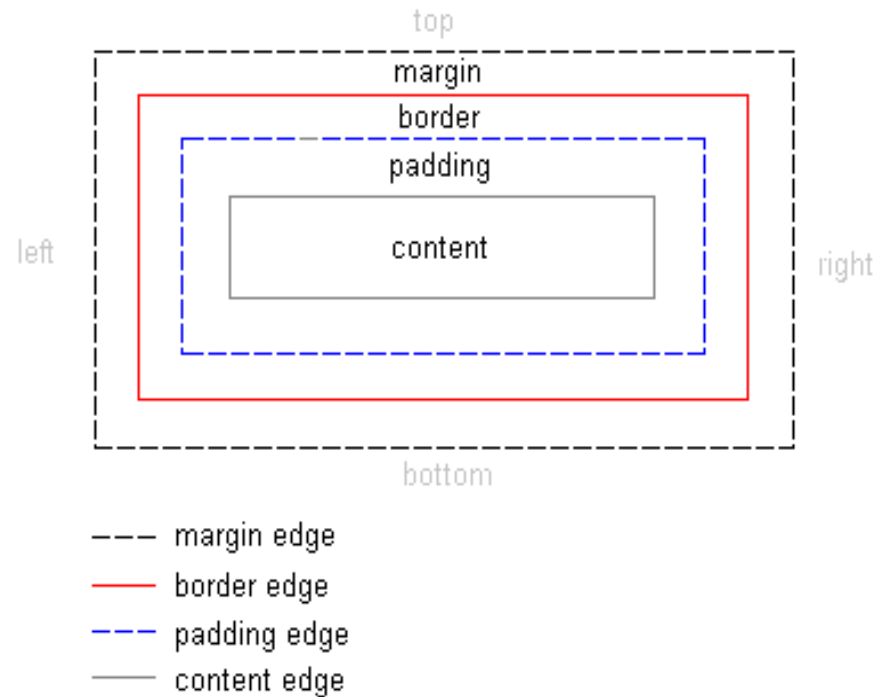


# Structuration d'un document HTML : les éléments span et div

## Notes

## Box model

- Modèle indiquant les différents niveaux d'imbrication de padding / margin / border



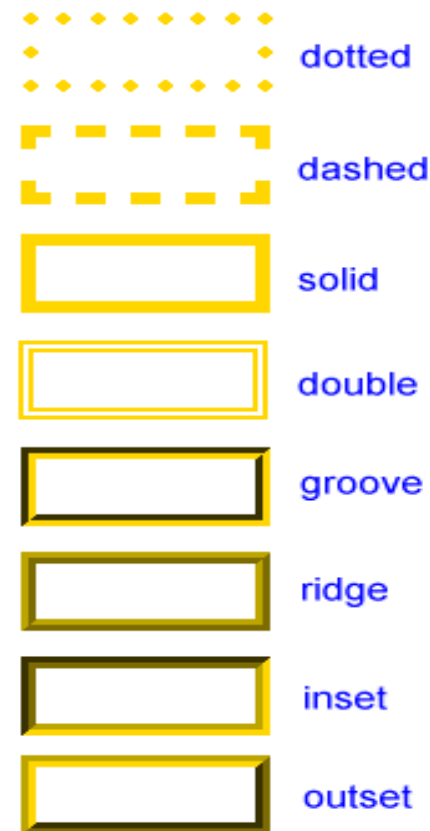
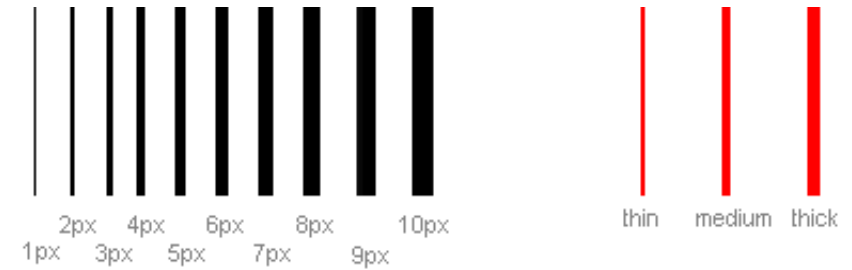
- Ce modèle pourtant standard est plus ou moins bien implémenté suivant les navigateurs.

## Box model

- Margin : distance séparant un élément des éléments l'entourant
- Padding : distance séparant le contenu de l'élément des bords de l'élément

## Quelques propriétés : les bordures

- border-width : Epaisseur de la bordure
- border-color : Couleur de la bordure
- border-style : Style de la bordure
- border : raccourci



## Quelques propriété : les bordures

### Notes

## Quelques propriétés : largeur et hauteur

- width : Largeur (relative ou absolue)
- height : Hauteur (relative ou absolue)

---

Seuls les éléments de type bloc supportent ces deux propriétés

---

- L'affichage (bloc ou inline) d'un élément peut être modifié avec la propriété 'display'.



## Quelques propriétés : largeur et hauteur

### Notes

## Conclusion

### Avantages de CSS

- Il s'agit d'un standard
- Il permet de définir et réutiliser des chartes graphiques très simplement.
- Il existe de nombreuses propriétés CSS (gestion du page flow, z-index, positionnement relatif ou absolu, ...).

### Inconvénients de CSS

- Toutes les propriétés ne sont pas implémentées de la même façon par tous les navigateurs.
- Il est quasi-impossible d'écrire une CSS avancée compatible avec tous les navigateurs.

# Conclusion

## Notes

# *Java et Web 2.0*

## *Introduction à XML*

---

*Version 1.0*

- Concepts XML
- Document Type Definition
- XML Schema
- Parseurs XML SAX et DOM
- Transformations XML et autres technologies XML

(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).



## Présentation de XML

### eXtensible Markup Language

- Permet de modéliser et de stocker des données de façon portable
- Extensible car il permet de définir de nouvelles balises : c'est un métalangage
- Ne permet pas à lui seul d'afficher les données qu'il contient.
- Dérivé de SGML (Standard Generalized Markup Language)
- Ne fait pas d'hypothèse en matière de système d'exploitation et de langage de programmation

### Document XML = données structurées

- Les données peuvent faire référence à la structure à laquelle elles sont conformes
- Ce lien permet d'effectuer une validation de conformité

# Présentation de XML

## eXtensible Markup Language

Le format HTML est utilisé pour formater et afficher les données qu'il contient : il est destiné à structurer, formater et échanger des documents d'une façon la plus standard possible..

XML est utilisé pour modéliser et stocker des données. Il ne permet pas à lui seul d'afficher les données qu'il contient.

Pourtant, XML et HTML sont tous les deux des dérivés d'une langage nommé SGML (Standard Generalized Markup Language). La création d'XML est liée à la complexité de SGML. D'ailleurs, un fichier XML avec sa DTD correspondante peut être traité par un processeur SGML.

## Document XML = données structurées

Les données sont la plupart du temps stockée sous la forme de textes et la structure peut en première approximation être assimilée à la grammaire d'un langage.

# Applications de XML

## Exemples

- Distribution d'information
- Gestion de données d'entreprise
- Transactions métier et transport de données
- Flux métier
- Gestion des connaissances
- Intégration d'applications
- Descripteurs de déploiement J2EE
- Format externe de persistance de données
- Web Services



# Applications de XML

## Notes

## Exemple de document XML

### Données

```
<!DOCTYPE web-app PUBLIC
"-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>

  <servlet>
    <servlet-name>titi</servlet-name>
    <servlet-class>Servlet1</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>titi</servlet-name>
    <url-pattern>s1</url-pattern>
  </servlet-mapping>

</web-app>
```

### Structure

```
<!ELEMENT web-app (icon?, display-name?, description?, distri-
butable?, context-param*, servlet*, servlet-mapping*, session-
config?, mime-mapping*, welcome-file-list?, error-page*, taglib*,
resource-ref*, security-constraint*, login-config?, security-role*,
env-entry*, ejb-ref*)>

<!ELEMENT servlet (icon?, servlet-name, display-name?, des-
cription?, (servlet-class|jsp-file), init-param*, load-on-startup?, se-
curity-role-ref*)>

...
```

- Deux organisations possibles
  - Données et structure dans deux fichiers séparés
  - Structure en entête des données, dans un seul fichier
- Alternative possible : XML Schema

## Exemple de document XML

### Notes

Dans cet exemple, le fichier de données fait référence à un descripteur appelé Document Type Definition ou DTD. Cette définition est stockée ici dans un fichier séparé.

# Règles syntaxiques XML

## Principales

- Le document doit contenir au moins une balise
- Chaque balise d'ouverture (exemple `<tag>`) doit posséder une balise de fermeture (exemple `</tag>`). Si le tag est vide, c'est à dire qu'il ne possède aucune données (exemple `<tag></tag>`), un tag abrégé peut être utilisé (exemple correspondant : `<tag/>`)
- Les balises ne peuvent pas être intercalées (exemple `<liste><element></liste></element>` n'est pas autorisé)
- Toutes les balises du document doivent obligatoirement être contenues entre une balise d'ouverture et de fermeture unique dans le document nommée élément racine
- Les valeurs des attributs doivent obligatoirement être encadrées avec des quotes simples ou doubles
- Les balises sont sensibles à la casse
- Les balises peuvent contenir des attributs même les balises vides
- Les données incluses entre les balises ne doivent pas contenir de caractères `<` et `&` : il faut utiliser respectivement `&lt;` ; et `&amp;` ;
- La première ligne du document doit normalement correspondre à la déclaration de document XML : le prologue.

# Règles syntaxiques XML

## Notes

## Contrôler la structure d'un document XML

### Deux possibilités

- DTD : Document Type Définition
  - Structure du document représentée à la façon d'une grammaire d'un langage
  - Une DTD n'est pas un document XML
  - Pas d'information précise sur les types des données
  - Simple à utiliser
- XML Schema
  - Nouvelle recommandation W3C
  - Est un document XML
  - Permet à XML de devenir un vrai langage de définition de données métier
  - Associe des types aux données d'un document XML
  - Plus complexe à utiliser

## Contrôler la structure d'un document XML

### Deux possibilités

Les professionnels chargés d'intégrer des données XML dans un système d'information considèrent aujourd'hui que la simplicité d'expression des DTD ne correspond plus aux besoins des applications modernes.

Parmi les langages de définition de types récemment proposés pour succéder aux DTD, XML Schéma du W3C est le plus utilisé, mais aussi l'un des plus difficile à maîtriser.

## Document Type definition

- Une DTD XML est analogue à une DTD SGML
- Une DTD XML est analogue à une grammaire (format EBNF : Extended Backus-Naur Form)

### Exemple

- Une anthologie comporte au moins un poème
- Un poème comporte un titre optionnel et au moins une strophe
- Un titre est un élément terminal (chaîne de caractères)
- Une strophe contient au moins une ligne
- Une ligne est un élément terminal (chaîne de caractères)

```
<!ELEMENT anthologie (poeme+)>
<!ELEMENT poeme (titre?, strophe+)>
<!ELEMENT titre (#PCDATA)>
<!ELEMENT strophe (ligne+)>
<!ELEMENT ligne (#PCDATA)>
```

### Contenu d'une DTD

- Types prédéfinis : ANY, EMPTY, PCDATA
- Règles d'utilisation : ?, +, ...
- Eléments avec attributs
- Entités



# Document Type Definition

## Notes

Les balises d'un document XML sont libres. Pour pouvoir valider si le document est correct, il faut définir un document nommé DTD qui est optionnel. Sans sa présence, le document ne peut être validé : on peut simplement vérifier que la syntaxe du document est correcte.

Une DTD est un document qui contient la grammaire définissant le document XML. Elle précise notamment les balises autorisées et comment elles s'imbriquent.

La DTD peut être incluse dans l'en tête du document XML ou être mise dans un fichier indépendant. Dans ce cas, la directive `< !DOCTYPE>` dans le document XML permet de préciser le fichier qui contient la DTD.

Il est possible d'utiliser une DTD publique ou de définir sa propre DTD si aucune ne correspond à ces besoins.

Pour être valide, un document XML doit avoir une syntaxe correcte et correspondre à la DTD, dans le cas où l'usage de cette dernière est défini.

## XMLSchema

### Tentative d'amélioration des DTD

- Typage fort et types normalisés par le W3C
- Support des cardinalités autres que 0, 1, plusieurs
- Mécanisme de dérivation
- Espaces de nommage
- XSD exprimé en XML

### Mais

- Plus difficiles à construire que des DTD
- Plus difficiles à lire

# XMLSchema

## Tentative d'amélioration des DTD

Pour pallier les inconvénients des Document Type Définitions, les schémas ont été créés.

Ce sont de vrais fichiers XML dont le but est de valider les documents.

Un schéma est un fichier de type .XSD.

## Exemple XMLSchema

### Données

```
<?xml version="1.0" ?>
<films xsi:noNamespaceSchemaLocation="cine.xsd"
  xmlns:xsi="http://www.w3.org/...">

  <film>
    <titre>le silence des agneaux</titre>
    <acteur>jodie foster</acteur>
    <acteur>anthony hopkins</acteur>
  </film>

  <film>
    <titre>conan le barbare</titre>
    <acteur>arnold schwarzenegger</acteur>
  </film>

</films>
```

### Structure en XSD

```
<?xml version="1.0" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">

  <xsd:element name="acteur" type="xsd:string"/>
  <xsd:element name="titre" type="xsd:string"/>

  <xsd:element name="film">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="titre"/>
        <xsd:element ref="acteur" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="films">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="film" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

## Exemple XMLSchema

### Notes

La balise `<films xsi:noNamespaceSchemaLocation="cine.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">` inséré dans le document XML permet d'associer le document à un XMLSchema.

## Les espaces de nommage XMLSchema

### Utilité

- Définir une zone dans laquelle il y a unicité des noms
- Permet d'éviter les collisions de noms d'éléments avec d'autres éléments importés

### Utiliser un espace de nommage

```
<wsdl:definitions targetNamespace="http://localhost:8080/axis/WSTest1.jws"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:apachesoap="http://xml.apache.org/xml-soap"
  xmlns:impl="http://localhost:8080/axis/WSTest1.jws"
  xmlns:intf="http://localhost:8080/axis/WSTest1.jws"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <wsdl:types>
    <schema targetNamespace="TP9" xmlns="http://www.w3.org/2001/XMLSchema"
      <import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
      <complexType name="unbean">
        <sequence>
          <element name="x" type="xsd:int"/>
          <element name="y" nillable="true" type="xsd:string"/>
        </sequence>
      </complexType>
    </schema>
  </wsdl:types>
```

# Les espaces de nommage XMLSchema

## Notes

## Les parseurs XML

### Besoin

- Analyser le contenu de documents XML
- Emplois typiques: travail sur données XML, web services, architectures SOA

### Techniques

- Construire une représentation en mémoire : DOM ou Document Object Model
- Déclencher des traitements lors de la rencontre d'éléments particuliers : SAX ou Simple API for XML

### Avantages et inconvénients

	Avantages	Inconvénients
DOM	parcours libre de l'arbre possibilité de modifier la structure et le contenu de l'arbre	gourmand en mémoire doit lire tout le document avant d'exploiter les résultats
SAX	peu gourmand en ressources mémoire rapide assez simple à mettre en oeuvre permet de ne traiter que les données utiles	traite les données séquentiellement un peu plus difficile à programmer, il est souvent nécessaire de sauvegarder des informations pour les traiter



# Les parseurs XML

## Techniques

Il existe plusieurs types de parseur. Les deux plus répandus sont ceux qui utilisent un arbre pour représenter et exploiter le document et ceux qui utilisent des événements. Le parseur peut en plus permettre de valider le document XML.

Ceux qui utilisent un arbre permettent de le parcourir pour obtenir les données et modifier le document.

Ceux qui utilisent des événements associent à des événements particuliers des méthodes pour traiter le document.

SAX (Simple API for XML) est une API libre créée par David Megginson qui utilisent les événements pour analyser et exploiter les documents au format XML.

Les parseurs qui produisent des objets composant une arborescence pour représenter le document XML utilisent le modèle DOM (Document Object Model) défini par les recommandations du W3C.

## Les parseurs XML

### Implémentations DOM

- Apache Xerces 2 : implémentation expérimentale DOM level 3
- Crimson
- Oracle XML Parser
- Microsoft XML Parser : DOM level 1 & 2

### Implémentations SAX

- Apache Xerces 2 : implémentation SAX 2.0.1 et SAX2 Extensions 1.0
- J2SE version 1.4.2 (package org.xml.sax)
- Microsoft XML Parser : SAX 1.0 & 2.0

# Les parseurs XML

## Notes

SAX et DOM ne fournissent que des définitions : ils ne fournissent pas d'implémentation utilisable. L'implémentation est laissée aux différents éditeurs qui fournissent un parseur compatible avec SAX et/ou DOM. L'avantage d'utiliser l'un des deux est que le code utilisé sera compatible avec les autres : le code nécessaire à l'instanciation du parseur est cependant spécifique à chaque fournisseur.

IBM fournit gratuitement un parseur XML : xml4j. Il est téléchargeable à l'adresse suivante : <http://www.alphaworks.ibm.com/tech/xml4j>

Le groupe Apache développe Xerces à partir de xml4j : il est possible de télécharger la dernière version à l'URL <http://xml.apache.org>

Sun a développé un projet dénommé Project X. Ce projet a été repris par le groupe Apache sous le nom de Crimson.

Ces trois projets apportent pour la plupart les mêmes fonctionnalités : ils se distinguent sur des points mineurs : performance, rapidité, facilité d'utilisation etc. ... Ces fonctionnalités évoluent très vite avec les versions de ces parseurs qui se succèdent très rapidement.

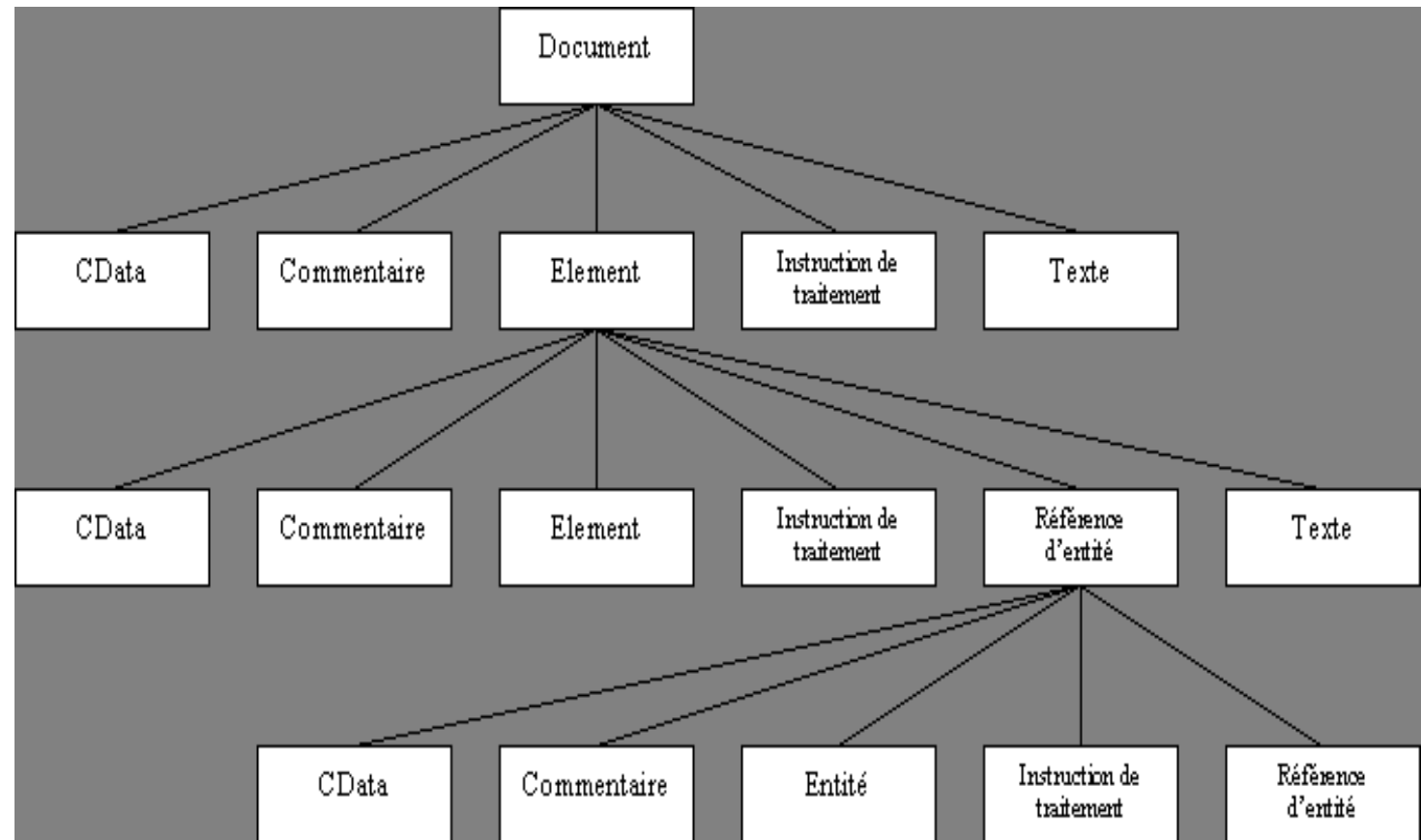
Pour les utiliser, il suffit de dézipper le fichier et d'ajouter les fichiers .jar dans la variable définissant le CLASSPATH.

Il existe plusieurs autres parseurs que l'on peut télécharger sur le web.

## Principes DOM

### Spécification W3C

- Construire une représentation mémoire arborescente d'un document XML
- Indépendance vis-à-vis des langages de programmation
- Permettre la manipulation du document et notamment sa modification
- Implémentation Java regroupée dans le package `org.w3c.dom`



# Principes DOM

## Spécification W3C

Il existe deux versions de DOM nommées «niveau» :

- DOM Core Level 1 : cette spécification contient les bases pour manipuler un document XML (document, élément et noeud)
- DOM Core Level 2 :

Le niveau 3 est en cours de développement.

# Principes SAX

---

[www.megginson.com/SAX/index.html](http://www.megginson.com/SAX/index.html)

---

## Programmation événementielle

- Handlers implémentant des interfaces d'écoute d'éléments

## Deux versions

- SAX 1
- SAX 2
  - Apporte le support des espaces de nommages
  - Principes de fonctionnement très proches de SAX 1
  - Evolutions d'API

# Principes SAX

## Notes

## Technologies XML

### Grand nombre de spécifications

- DTD, Schémas XML : Définition de structures XML
- DOM, SAX : Parsing de documents XML
- XSL, XSLT, XSL-FO : Transformation de documents XML
- XPath : Système de sélection de noeuds
- XQuery : Requêtes sur un document XML
- XPointer : Référence à un fragment de document XML
- ...



# Technologies XML

## Notes

## eXtensible Stylesheet Language (XSL)

- Transformer un document XML en un autre document
- Véritable langage de programmation
- Interprété directement par certains navigateurs

### Deux constituants

- Transformation des données avec XSLT (XSL Transformation)
  - Sélection de noeuds
  - Application de règles de gabarits (template rules) pour transformer un arbre source en arbre destination
- Formattage des données avec XSL-FO (XSL Formatting Objects)
  - Application de mises en formes
  - Processeurs prêts à l'emploi, comme FOP pour produire des documents PDF

# eXtensible Stylesheet Language (XSL)

## Deux constituants

Un document XML peut être représenté comme une structure arborescente. Ainsi XSLT permet de transformer les documents XML à l'aide de feuilles de style contenant des règles appelées template rules (ou règles de gabarit en français).

Le processeur XSLT (composant logiciel chargé de la transformation) crée une structure logique arborescente (on parle d'arbre source) à partir du document XML et lui fait subir des transformations selon les template rules contenues dans la feuille XSL pour produire un arbre résultat représentant, par exemple, la structure d'un document HTML. Les composants de l'arbre résultat sont appelés objets de flux.

Chaque template rule définit des traitements à effectuer sur un élément (noeud ou feuille) de l'arbre source. On appelle "patterns" (en français motifs, parfois "éléments cibles") les éléments de l'arbre source.

L'arbre source peut être entièrement remodelé et filtré ainsi qu'ajouter du contenu à l'arbre résultat, si bien que l'arbre résultat peut être radicalement différent de l'arbre source.

le langage de formatage des données (XSL/FO), c'est-à-dire un langage permettant de définir la mise en page (affichage de texte ou de graphiques) de ce qui a été créé par XSLT.

Une fois l'arbre source créé, XSL/FO permet de formater le résultat, c'est-à-dire d'interpréter l'arbre résultat, ou plus exactement les objets de flux le composant en leur appliquant des objets de mise en forme afin d'en faire une représentation visuelle (papier, écran, ...)

## Exemple XSLT

### Production d'une page HTML à partir d'un document XML

```
<?xml version="1.0"?>
<BIB>
  <LIVRE>
    <TITRE>titre livre 1</TITRE>
    <AUTEUR>auteur 1</AUTEUR>
    <EDITEUR>editeur 1</EDITEUR>
  </LIVRE>
  <LIVRE>
    <TITRE>titre livre 2</TITRE>
    <AUTEUR>auteur 2</AUTEUR>
    <EDITEUR>editeur 2</EDITEUR>
  </LIVRE>
  <LIVRE>
    <TITRE>titre livre 3</TITRE>
    <AUTEUR>auteur 3</AUTEUR>
    <EDITEUR>editeur 3</EDITEUR>
  </LIVRE>
</BIB>
```

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
  <xsl:template match="/">
    <html>
      <body>
        <table border="1" cellspacing="0" cellpadding="3">
          <tr bgcolor="#FFFF00">
            <td>Titre</td> <td>Auteur</td> <td>Editeur</td>
          </tr>
          <xsl:for-each select="BIB/LIVRE"><tr>
            <td><xsl:value-of select="TITRE"/></td>
            <td><xsl:value-of select="AUTEUR"/></td>
            <td><xsl:value-of select="EDITEUR"/></td>
          </tr></xsl:for-each>
        </table>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

## Exemple XSLT

### Production d'une page HTML à partir d'un document XML

Après les balises de départ d'un fichier XSL, on aborde un tableau tout à fait classique en Html.

Pour chaque élément de type "BIB/LIVRE", on remplit la cellule correspondante au titre par la balise `xsl:value-of` avec l'attribut `select="TITRE"` qui indique comme chemin d'accès la balise racine BIB, la balise LIVRE et la balise TITRE. Et bien entendu de même pour les balises AUTEUR et EDITEUR.

## Exemple XSLT

### Production d'une page HTML à partir d'un document XML

- Lien entre XML et XSL

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="bib.xsl"?>
<BIB>
  <LIVRE>
    <TITRE>titre livre 1</TITRE>
    <AUTEUR>auteur 1</AUTEUR>
    <EDITEUR>editeur 1</EDITEUR>
  </LIVRE>
  <LIVRE>
    <TITRE>titre livre 2</TITRE>
    <AUTEUR>auteur 2</AUTEUR>
    <EDITEUR>editeur 2</EDITEUR>
  </LIVRE>
  <LIVRE>
    <TITRE>titre livre 3</TITRE>
    <AUTEUR>auteur 3</AUTEUR>
    <EDITEUR>editeur 3</EDITEUR>
  </LIVRE>
</BIB>
```

## **Exemple XSLT**

### **Production d'une page HTML à partir d'un document XML**

On ajoute au fichier XML le lien vers la feuille de style XSL.

# *Java et Web 2.0*

## *Notions de base XML*

---

*Version 1.0*

- Présentation des langages à balises
- Le langage XML
- Le langage XHTML

(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects, est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).





## Présentation des langages à balises

### Le langage SGML à la base des autres langages à balise

- Origine de SGML en 1960 avec le langage GML (Generalized Markup Language)
- Norme ISO
- Utilisation de '<' et '>' pour délimiter la déclaration des balises

### Le langage HTML

- Langage utilisant les règles d'écriture du langage SGML
- Utilisé pour la publication de pages Web
- Version courante 5.0

---

Non supporté sur tous les navigateurs contrairement à la 4.1

---

### Le langage XML

- Sous ensemble du langage SGML
- Version simplifiée (analyse et manipulation plus simple)

# Présentation des langages à balises

**Notes :**

## Présentation des langages à balises

### Le langage XML (suite)

- Version courante 1.1
- De nombreuses utilisations du langage XML
  - Le langage XHTML (eXtensible HyperText Markup Language)
  - Les flux RSS (Really Simple Syndication)
  - ...
- Utilisation d'espaces de nommage

# Présentation des langages à balises

## Notes

## **Le World Wide Web Consortium (W3C)**

### **Consortium à l'origine des spécifications**

- Pour des langages du Web
- Pour des protocoles liés au réseaux
- Pour de la mise en forme de pages Web
- ...



**Met à disposition des outils de validation (pour HTML, CSS, ...)**

**Dirigé par Tim Berners-Lee (qui est à l'origine du 'Web')**

**Constitué des principaux acteurs du Web (Microsoft, IBM,...)**

**Accessible à l'URL : <http://www.w3.org>**

# **Le World Wide Web Consortium (W3C)**

**Notes :**

# Le langage XML

## Est un meta langage (comme SGML)

- est un langage à balise
- génère un sous-ensemble de SGML
- peut-être validé à l'aide de fichiers de description (DTD ou schéma XML)

## Définit les règles de constructions, mais pas la sémantique

## Un document XML peut-être défini comme bien formé et/ou valide :

- Bien formé : Le document respecte les règles d'écriture de XML
- Valide : Le document est valide est respecte la structure défini dans une DTD ou un schéma XML

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE Test SYSTEM "test.dtd">

<Test>
  <nom>Leuville</nom>
  <adresse>
    <rue>3 rue de la porte de Buc</rue>
    <code-postal>78000</code-postal>
    <localite>Versailles</localite>
  </adresse>
</Test>
```

Exemple de document XML valide



# Le langage XML

**Notes :**

## **Le langage XML**

### **Utilisé dans de nombreux cas différents**

- Définition de langages
- Fichiers de configurations (pour des serveurs ...)
- Définition de protocoles (SOAP, WSDL ...)
- etc.

# Le langage XML

## Notes

## Le langage XHTML

### Peut-être vu comme une évolution du langage HTML

- Plus structuré
- Les règles d'écriture peuvent-être plus strictes
- Version courante 2.0

---

Version abandonnée et non recommandée contrairement à la version 1.1 standardisée.

---

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
...
</html>
```

### Plusieurs niveaux d'écriture des pages

- standard : permet d'utiliser une page écrite en HTML 4
- transitional : règle d'écriture plus strict (<html>...</HTML> impossible)
- strict : impose de respecter l'ensemble des règles d'écriture au formalisme XML

# Le langage XHTML

**Notes :**

# *Java et Web 2.0*

## *Présentation de Dojo*

---

*Version 1.0*

- Qu'est ce que Dojo ?
- Les navigateurs supportés par Dojo
- Les composants Dojo
- Un premier exemple

(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).



## Qu'est-ce que Dojo ?

### Définition

- Framework open source en JavaScript
- Développement d'applications Web (côté client) en JavaScript
- Facilité l'implémentation d'AJAX pour les sites Web
  - Des templates et bibliothèques sont proposés au niveau de la communication client-serveur
  - Des widgets UI (Dijit) de haut niveau sont fournis
  - APIs de gestion d'accès aux navigateurs et d'encapsulation des implémentations JavaScript
- Dernière version 1.9



# Qu'est-ce que Dojo ?

## Notes

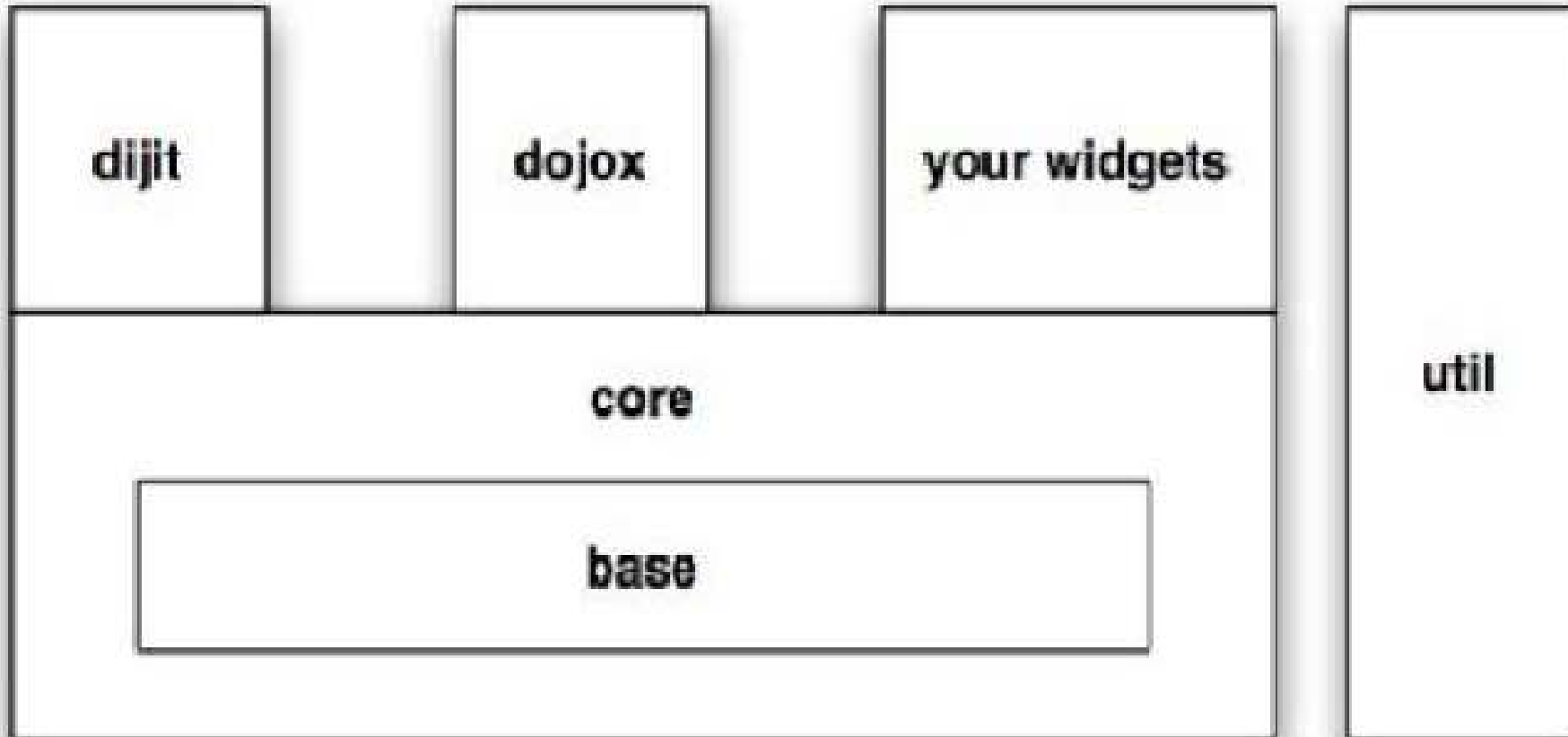
## Les navigateurs supportés

- Dojo 1.9 est compatible avec les navigateurs suivants :
  - Chrome : 13 - 26
  - Firefox : 3.6.x - 20
  - Internet Explorer : 8 - 10
  - Opera : 10.50 - 12
  - Safari : 5.0.x et 5.1.x
  
- Et pour les mobiles utilisant :
  - iOS 4.x et 5.x, et 6.x
  - Android 2.x, 3.x, et 4.x
  - Blackberry 6-7 et 10

# Les navigateurs supportés

## Notes

## Les composants Dojo



# Les composants Dojo

## Notes

# Les composants Dojo

## Dojo Base

- Petit noyau de taille 26Kb
- De nombreuses fonctionnalités telles que:
  - Détection de navigateur
  - Système de chargement de package
  - Gestion événementielle
  - Support des animations
  - Programmation asynchrone AJAX
  - Moteur de requêtage CSS3 très performant
  - Fonctions de traitement du CSS et du positionnement
  - Programmation orientée objet
  - Protection contre les fuites mémoire
  - Structure DOM
  - ...

# Les composants Dojo

## Notes

# Les composants Dojo

## Dojo Core

- Contient les fonctionnalités dites optionnelles:
  - Accès aux données unifié (dojo.data)
  - Outils de debug multi-navigateur (Firebug Lite)
  - Drag and drop (dojo.dnd)
  - Animations avancées (dojo.fx)
  - Support multilingue
  - Les fonctions utiles: formatage des dates, formatage des nombres, gestion de chaîne, ...
  - Couche de transport avancée (IFRAME, JSON-P)
  - Gestion des cookies
  - Appel aux procédures distantes (RPC), y-compris JSON-P
  - Gestion du bouton de retour du navigateur
  - ...



# Les composants Dojo

## Notes

Dojo Base + Core fournit près de 50 scripts JavaScript.

## Les composants Dojo

### Dijit (Dojo widgets)

- Contient la bibliothèque de widgets :
  - Widgets orientés objet par le biais de la représentation DOM
  - De nombreux types de widgets
  - Possibilité d'attribuer des thèmes aux widgets

# Les composants Dojo

## Notes

## Les composants Dojo

### Dojox (Dojo extensions)

- Contient les projets expérimentaux de Dojo et Dijit.
- De nombreuses fonctionnalités et widgets proposées mais instables.
- Un fois testé et approuvé sous dojox ceux-ci passent sous Dojo ou Dijit.

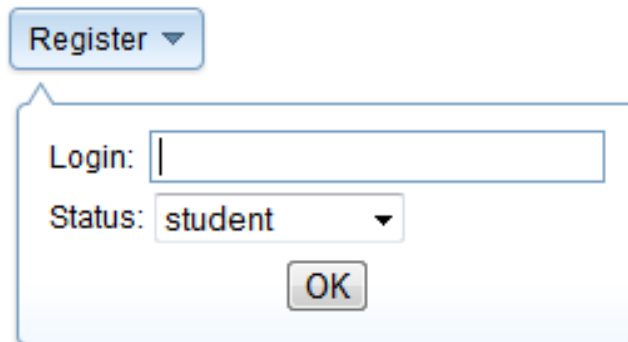
# Les composants Dojo

## Notes

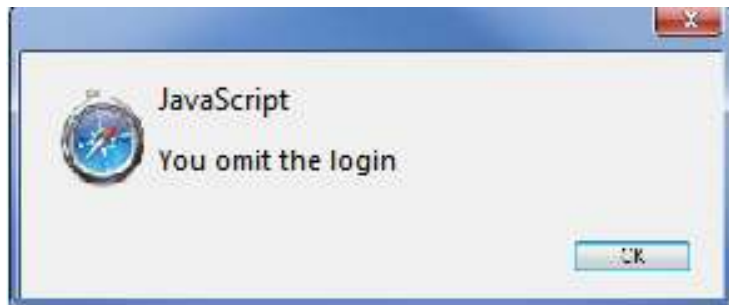
## Un premier exemple Dojo

- Création d'une première page HTML proposant la possibilité de s'enregistrer par exemple.
- Utilisation d'une boîte de dialogue Dojo affichant le formulaire et d'une liste de sélection.
- La validation affiche une fenêtre d'information.

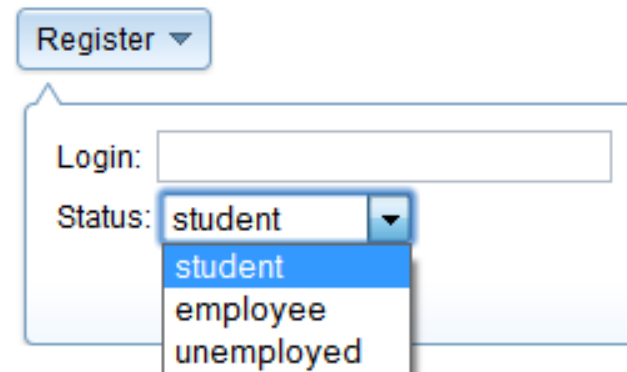
### Mon premier test avec Dojo



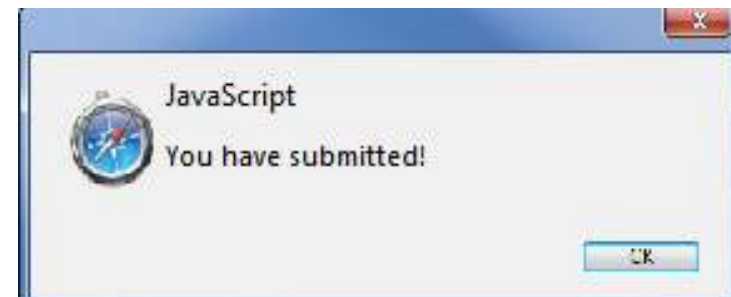
A Dojo dialog box titled "Register" with a dropdown arrow. Inside the dialog, there is a "Login:" label followed by an empty text input field. Below it is a "Status:" label followed by a dropdown menu showing "student". At the bottom right of the dialog is an "OK" button.



### Mon premier test avec Dojo



A Dojo dialog box titled "Register" with a dropdown arrow. Inside the dialog, there is a "Login:" label followed by an empty text input field. Below it is a "Status:" label followed by a dropdown menu. The dropdown menu is open, showing three options: "student", "employee", and "unemployed". At the bottom right of the dialog is an "OK" button.



# Un premier exemple Dojo

## Notes

## Un premier exemple Dojo

- Il faut déclarer la librairie Dojo, dojo.js, dans le header
- Possibilité de définir et rajouter les modules Dojo requis.

```
<!DOCTYPE0HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>TooltipDialog Widget Tests</title>
<style type="text/css">
@import "../WEB_DOJO/dojo/dojo/resources/dojo.css";
</style>

<!-- required: a default dijit theme: -->
<link id="themeStyles" rel="stylesheet"
href="../WEB_DOJO/dojo/dojo/dijit/themes/claro/claro.css">

<!-- required: dojo.js -->
<script type="text/javascript"
src="../WEB_DOJO/dojo/dojo/dojo.js"
data-dojo-config= "has:{'dojo-firebug': true}, parseOnLoad:
false, async: false">
</script>
<script type="text/javascript">
    dojo.require("dijit.dijit");
    dojo.require("dijit.TooltipDialog");
    dojo.require("dijit.form.Button");
```

```
    dojo.require("dijit.form.TextBox");
    dojo.require("dijit.form.Select");
    dojo.require("dojo.parser");
    dojo.addOnLoad(function() {
        dojo.parser.parse();
    });
</script>
<script type="text/javascript">
    // click treatment
    require([ "dojo/on", "dojo/dom", "dojo/domReady!" ],
function(on, dom) {
    var myButton = dom.byId("submit");
    on(myButton, "click", function(evt) {
        var myText = dom.byId("text").value;
        if (myText === "" || myText === null)
            alert("You omit the login ");
        else
            alert("You have submitted!");
    });
});
</script>
</head>
```



# Un premier exemple Dojo

## Notes

## Un premier exemple Dojo

```
<!-- HTML BODY -->
<body class="claro">
  <h1 class="testTitle">Mon premier test avec Dojo</h1>
  <div dojoType="dijit.form.DropDownButton" id="tooltipDlgButton">
    <span>Register</span>
    <div dojoType="dijit.TooltipDialog" id="tooltipDlg" title="Enter Login information">
      <table>
        <tr>
          <td><label for="text">Login:</label></td>
          <td><span dojoType="dijit.form.TextBox" name="text" id="text"></span></td>
        </tr>
        <tr>
          <td><label for="combo">Status:</label></td>
          <td><select name="combo" id="combo" hasDownArrow="true">
              <option value="Student">student</option>
              <option value="Employee">employee</option>
              <option value="Unemployed">unemployed</option>
            </select></td>
        </tr>
        <tr>
          <td colspan="2" align="center">
            <button type="submit" id="submit">OK</button>
          </td>
        </tr>
      </table>
    </div>
  </div>
</body>
</html>
```

# Un premier exemple Dojo

## Notes

# *Java et Web 2.0*

## *Utilisation de Dojo*

---

*Version 1.0*

- Notions de base
- Accès à l'arbre DOM
- Gestion des événements
- AJAX avec Dojo

(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).



## Notions de base de Dojo

### L'objet dojoConfig

- Anciennement appelé djConfig dans la version 1.6 (déprécié depuis la version 1.7).
- Définition des propriétés de configuration.
- Déclaré dans le script avant même l'appel à dojo.js. Si non respecté, les propriétés de configuration ne seront pas prises en compte.
- Deux exemples équivalents pour déclarer dojoConfig:

```
<!-- set Dojo configuration, load Dojo -->
<script>
    dojoConfig = {
        has: {
            "dojo-firebug": true
        },
        parseOnLoad: false,
        foo: "bar",
        async: true
    };
</script>
<script
    src="http://ajax.googlea-
pis.com/ajax/libs/dojo/1.7.1/dojo/dojo.js">
</script>
```

```
<script
src="http://ajax.googlea-
pis.com/ajax/libs/dojo/1.7.1/dojo/dojo.js"
data-dojo-config= "has:{'dojo-firebug': true}, parseOnLoad:
false, foo: 'bar', async: true">
</script>
```

## Notions de base de Dojo

### Notes

dojoConfig doit être déclaré avant dojo.js car celui-ci possède également une configuration par défaut qui sera écrasée par celle décrite avant.

## Notions de base de Dojo

### Les attributs de dojoConfig

- **has()** : permet d'activer ou de désactiver certaines fonctions supportées par Dojo (ex d'attributs: dojo-debug:true, dojo-debug-messages:false, dojo-amd-factory-scan:false)
- **parseOnLoad**: permet de parser tout une page avec toutes les dépendances initialisées. Par défaut, il est à "false", il est d'ailleurs recommandé par Dojo de ne pas l'activer et de déclarer explicitement la méthode *parser.parse()* de dojo/parser.
- **async**: défini si Dojo core doit être généré de façon asynchrone (valeur possible: true, false, legacyAsync).
- **waitSeconds**: temps d'attente nécessaire avant de signaler le délai de charge des modules.
- **packages**: un tableau d'objets fournissant le nom et la localisation d'un paquet.
- **paths**: permet de créer des chemins d'accès personnalisés.
- **deps**: un tableau contenant les chemins d'accès aux ressources générées juste après le module dojo.
- **callback**: ce rappel est exécuté une fois que toutes les ressources ont été générées.
- ...



## Notions de base de Dojo

### Notes

Il est également possible d'ajouter des propriétés personnalisées.

## Notions de base de Dojo

### Les tableaux avec Dojo

- Les méthodes disponibles avec `dojo/_base/array`:
  - **dojo.indexOf**: retourne le premier indice contenant l'élément recherché dans le tableau. S'il n'existe pas, -1 est retourné.
  - **dojo.lastIndexOf**: comme précédemment mais avec le dernier indice du tableau.
  - **dojo.every**: teste si oui ou non tous les éléments du tableau correspondent à la valeur indiquée.
  - **dojo.some**: teste si un des éléments donné est contenu dans le tableau.
  - **dojo.filter**: crée un nouveau tableau contenant les éléments qui ont passé la fonction de filtre.
  - **dojo.map**: crée une map à partir d'un tableau donné, via une fonction filtrant chaque élément du tableau.
  - **dojo.foreach**: On parcourt chaque élément du tableau à l'aide d'une fonction. Il est également possible de passer par un **for** basic.
  - ...

```
require(["dojo/_base/array"], function(baseArray) {
    var arr1 = [1,2,3,4,3,2,1,2,3,4,3,2,1];
    baseArray.indexOf(arr1, 2); // returns 1
    baseArray.forEach(arr1, function(item, index){
        if(index == 3){
            arr1[0]=item;
            arr1[index] = 7;
        }
    });
    alert(arr1); // [4,2,3,7,3,2,1,2,3,4,3,2,1]
});
```

# Notions de base

## Notes

## Notions de base de Dojo

### Les objets avec Dojo

- **dojo.declare**: méthode créant des objets Dojo via le module `dojo/_base/declare`. Il comprend 3 arguments:
  - **className**: le nom de la classe à donner. Ce nom sera globalement disponible dans toute l'application. Toutefois Dojo recommande d'utiliser ce paramètre uniquement pour les classes utilisant le parser. Sinon de l'omettre, afin d'avoir une classe anonyme accessible dans son champ d'application uniquement.
  - **superClass**: définit la/les classe(s) mère(s) héritée(s). Elle peut être *null* si elle n'hérite de personne, ou un tableau de classes dans le cas de l'héritage multiple. Si plusieurs méthodes ou attributs sont définis identiquement dans plusieurs classes mères alors c'est la dernière du tableau qui est prise en compte.
  - **properties**: définit les attributs et méthodes de la classe.
    - n On peut y définir un constructeur avec la propriété "constructor:"
    - n Ou redéfinir une méthode héritée avec "this.inherited".
    - n Le mot-clé "**this**" fait référence à l'instance et non à la classe d'origine.
- Pour instancier l'objet crée, on fait un **new** de l'objet.

# Notions de base

## Notes

## Notions de base de Dojo

### Exemple avec les objets Dojo

```
// Define class A
var A = declare(null, {
    // The constructor
    constructor: function(args){
        console.log("Construct");
    },

    myMethod: function(){
        console.log("Hello!");
    }
});

// Define class B
var B = declare(A, {
    myMethod: function(){
        // Call A's myMethod
        this.inherited(arguments); // arguments provided to A's myMethod
        console.log("World!");
    }
});
```

```
// Create an instance of B
var myB = new B();
myB.myMethod();
```

# Notions de base

## Notes

## Accès à l'arbre DOM

- **dom.byId** permet de récupérer un noeud de l'arbre DOM avec l'identifiant donné. L'inconvénient avec cette méthode est qu'elle ne permet pas d'accéder à plusieurs noeuds.
- **dojo.query** offre une solution à cet inconvénient en passant par des sélecteurs comme en CSS3. Elle retourne donc une liste de noeud *NodeList* et prend en argument le *pattern* recherché. Ce *pattern* peut avoir une syntaxe particulière si l'on souhaite effectuer des requêtes avancées ou plus restrictives:

**Table 1: Les sélecteurs supportés par dojo.query**

Sélecteur	Signification
*	Tous les éléments de l'arbre
E	Tous les éléments de type E (par exemple la balise li)
E[foo]	Tous les éléments de type E avec un attribut "foo"
E[foo="bar"]	Les éléments E avec un attribut foo égal à bar
E:nth-child(n)	Les éléments E le n-ième noeud enfant de son parent (ex: div.division1-1:nth-child(3)). La valeur de n peut être un nombre ou "even" ou "odd" pour les valeurs paires et impaires.
E:not(s)	Les éléments de E ne correspondant pas au simple sélecteur s.
E:empty	Les éléments de E ne possédant pas de noeuds enfants.
#myId	Les éléments avec l'identifiant myId.
.myClass	Les éléments avec la class="myClass"
E > F	Les éléments de F étant enfant des éléments de E
s1 s2	L'ensemble des éléments du sélecteur s2 qui sont descendants de l'ensemble des éléments de s1.
s1, s2	L'ensemble des éléments de s1, union avec l'ensemble des éléments de s2.



## Accès à l'arbre DOM

### Notes

La méthode query de dojo peut également prendre un second paramètre afin de restreindre la recherche du premier paramètre dans la limite indiquée par ce second argument.

## Accès à l'arbre DOM

### La liste de noeuds NodeList

- La liste de noeuds se parcourt comme un tableau sous Dojo.
- Cette objet contient des méthodes utiles afin de manipuler la liste de noeuds:
  - **attr**: récupère ou modifie l'attribut HTML de chaque noeud.
  - **style**: comme la méthode attr.
  - **addClass**: ajoute une classe CSS dans chaque noeud et retourne la racine.
  - **connect**: connecte un gestionnaire à un événement (déprécié depuis la version 1.7).
  - **on**: gère les événements en retournant un tableau des gestionnaires disponibles (cf. la gestion d'événements).
  - ...

# Accès à l'arbre DOM

## Notes

## Accès à l'arbre DOM

### Exemple

```
<script type="text/javascript">
//Wait for the DOM to be ready before working with
it
require(["dojo/query", "dojo/NodeList-dom",
"dojo/domReady!"], function(query) {
    // Add "red" to the className of each node
    // matching the selector ".odd"
    query(".odd").addClass("red");
    // Add "blue" to the className of each node
    // matching the selector ".even"
    query(".even").addClass("blue");
});
</script>
```

```
<body>
    <div id="List">
        <div class="odd">One</div>
        <div class="even">Two</div>
        <div class="odd">Three</div>
        <div class="even">Four</div>
        <div class="odd">Five</div>
        <div class="even">Six</div>
    </div>
</body>
```

# Accès à l'arbre DOM

## Notes

## Gestion des événements

### Les événements DOM

- Tous les navigateurs ne respectent pas la spécification W3C DOM, il peut donc y avoir des problèmes d'ordonnancement des gestionnaires d'écoutes ou de fuites mémoire.
- Dojo a normalisé ces différentes APIs pour prévenir ces problèmes et met à disposition une API appelée `dojo/on` pour la gestion d'événements.
- La méthode **`dojo.on(element, eventName, handler)`** s'applique à tout élément de type window, document, node, form, mouse, keyboard...
  - Cette méthode renvoie un gestionnaire d'événements qui peut alors être détruit avec **`remove()`**.
  - Il est possible d'appliquer la méthode **`on.once()`** afin d'appeler le gestionnaire qu'une seule fois.
  - Une particularité de cette méthode est que le gestionnaire d'événements s'exécute dans le contexte du noeud donné dans le premier argument. Par ailleurs, il est possible de définir ce contexte via la méthode **`lang.hitch`** du module `dojo/_base/lang`, pratique pour l'appel à une méthode d'un objet.

### La délégation d'événements

- Attacher un contrôleur à un noeud de niveau supérieur au lieu de plusieurs noeuds au même niveau, pour qu'il récupère les événements cibles obtenus et suivre la logique du gestionnaire.
  - Sa syntaxe: **`dojo.on(elementParent, selecteur: nomEvenemnt, gesitonnaire)`**

# Gestion des événements

## Notes

## Gestion des événements

### Publication et souscription

- Pouvoir enregistrer/ souscrire un gestionnaire d'événement par la publication d'un sujet "topic", sans avoir à créer un objet traitant l'événement ou avoir un gestionnaire rattaché à un noeud existant.
- Le module dojo/topic permet donc de souscrire et publier des événements types.
  - **topic.subscribe(topic, context, handler)**: souscrit un gestionnaire d'événements nommé par le topic dans un certain contexte, le handler peut-être omis si par exemple le traitement n'est que le renvoi d'une chaîne de caractères. Cette méthode renvoie un gestionnaire ayant la possibilité de le détruire avec la méthode **remove()**.
  - **topic.publish(topic, args...)**: publie le traitement par appel au nom du topic souscrit, suivant les arguments nécessaires.



## Gestion des événements

### Notes

Dans les versions précédant la 1.7, ces méthodes sont dans le module dojo. D'ailleurs, il fallait que les arguments de `dojo.publish` soient placés dans un tableau.

## Gestion des événements

### Exemple de gestion d'événement et de délégation.

```
require(["dojo/on", "dojo/dom", "dojo/query", "dojo/dom-Ready!"], function(on, dom){
    var myObject = {
        id: "myObject",
        onClick: function(evt){
            alert("The scope of this handler is " + this.id);
        }
    };
    var div = dom.byId("parentDiv");
    on(div, ".clickMe:click", myObject.onClick);
});
```

<body>

```
<div id="parentDiv">
  <button id="button1" class="clickMe">Click me
</button>
  <button id="button2" class="clickMe">Click me also
</button>
  <button id="button3" class="clickMe">Click me too
</button>
  <button id="button4" class="clickMe">Please click me
</button>
</div>
</body>
```

# Gestion des événements

## Notes

## Gestion des événements

### Exemple de souscription et de publication.

```
<script>
require(["dojo/on", "dojo/topic", "dojo/dom", "dojo/dom-construct", "dojo/domReady!"],
function(on, topic, dom, domConstruct) {
var alertButton = dom.byId("alertButton"),
createAlert = dom.byId("createAlert");
on(alertButton, "click", function() {
// When this button is clicked,
// publish to the "alertUser" topic
    topic.publish("alertUser", "I am alerting you.");
});
on(createAlert, "click", function(evt){
    // Create another button
    var anotherButton = domConstruct.create("button", {
        innerHTML: "Another alert button"
    }, createAlert, "after");
    // When the other button is clicked,
    // publish to the "alertUser" topic
    on(anotherButton, "click", function(evt){
        topic.publish("alertUser", "I am also alerting you.");
    });
});
});
```

```
// Register the alerting routine with the "alertUser"
topic.subscribe("alertUser", function(text){
    alert(text);
});
</script>
```

```
<body>
<h1>Demo: Publish and Subscribe with dojo/topic</h1>
<button id="alertButton">Alert the user</button>
<button id="createAlert">Create another alert button</button>
</body>
```

# Gestion des événements

## Notes

## AJAX avec Dojo

### AJAX I/O

- Possibilité de personnaliser les requêtes AJAX avec les méthodes **xhrPost** et **xhrGet** du module de base de dojo, avec plusieurs propriétés offertes:
  - **url**: URL effectuant le traitement
  - **handleAs**: Une chaîne de caractères représentant le format des données retournées (ex: json, text, xml, javascript...)
  - **timeout**: Le temps limite en milliseconde considérant que la requête est un echec.
  - **content**: Une map contenant les informations envoyées vers le serveur.
  - **form**: Option remplissant la map du formulaire précédemment décrite. Si aucune URL est donnée, ce sera celle décrite dans l'attribut "action" de la balise form qui sera prise en compte. D'ailleurs si la propriété "content" précédente est utilisée alors celle-ci écrasera la valeur de form. Il ne faut donc pas utiliser ces 2 propriétés ensembles.
  - **load(response, ioArgs)**: Chargement du résultat retourné, "response" est sous format indiqué par "handleAs".
  - **error(errorMessage)**: Chargement du message d'erreur retourné lorsque la requête a échoué.
  - **handle(response, ioArgs)**: Chargement du résultat obtenu quelque soit ça réussite ou non.

# AJAX avec Dojo

## Notes

## AJAX avec Dojo

### AJAX xhrGet Exemple

```
// The "xhrGet" method executing an HTTP GET
dojo.xhrGet({
    // The URL to request
    url: "get-message.php",
    // The method that handles the request's successful result
    // Handle the response any way you'd like!
    load: function(result) {
        alert("The message is: " + result);
    }
});
```



# **AJAX avec Dojo**

## **Notes**

## AJAX avec Dojo

### JSON / JSONP

- Format de données utilisable avec AJAX permettant de décrire des objets ou structures de données complexes.
- Dojo peut parser très facilement les formats JSON depuis un serveur (la propriété "handleAs" le permet).
- Une limite des technologies AJAX est qu'il se restreint au domaine courant (ex: pas possible de récupérer du contenu de dojotoolkit.org à partir de son site web avec xhrGet).
- Une solution à cette limite est JSONP (JSON with Padding):
  - Un noeud **script** est créé, celui-ci définit les différents paramètres et la fonction de retour qu'on lui fournit avec pour valeur de l'attribut **src** contenant l'URL auquel on souhaite accéder.
  - Ce noeud est ajouté à l'arbre DOM afin qu'il accède au domaine requis tout au début.
  - Le serveur envoie un JSON, enveloppé par la fonction de rappel comme spécifié dans la requête.
  - Le navigateur exécute le code renvoyé en remettant la réponse du serveur à la fonction de rappel.
  - Le module et la méthode Dojo sont **dojo.io.script**.
  - Les paramètres sont identiques à celle de **xhrGet/xhrPost**, avec une option supplémentaire qui est **callBackParamName**, contenant le nom de la fonction de rappel.

# **AJAX avec Dojo**

## **Notes**

## AJAX avec Dojo

### Exemple JSON

```
// Using dojo.xhrGet, as we simply want to retrieve information
dojo.xhrGet({
    // The URL of the request
    url: "get-news.php",
    // Handle the result as JSON data
    handleAs: "json",
    // The success handler
    load: function(jsonData) {
        // Create a local var to append content to
        var content = "";
        // For every news item we received...
        dojo.forEach(jsonData.newsItems, function(newsItem) {
            // Build data from the JSON
            content += "<h2>" + newsItem.title + "</h2>";
            content += "<p>" + newsItem.summary + "</p>";
        });
        // Set the content of the news node
        dojo.byId("newsContainerNode").innerHTML = content;
    },
    // The error handler
    error: function() {
        dojo.byId("newsContainerNode").innerHTML = "News is not available at this time."
    }
});
```

# AJAX avec Dojo

## Notes

# ***Java et Web 2.0***

## ***JSF 2.0***

---

*Version 1.0*

- Présentation
- Les serveurs d'applications
- Exemple d'utilisation
- ...

(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects, est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).



## **Les fonctionnalités apportées par JSF**

- Des composants d'interface graphique (UI Components) extensibles.
- Une gestion de la navigation.
- Un système de conversion et de validation des données saisies.
- Une gestion automatisée du cycle de vie de certains JavaBeans.
- Un gestionnaire d'événements.
- Une gestion aisée des erreurs
- Une gestion de l'internationalisation (i18n)



# **Les fonctionnalités apportées par JSF**

## **Notes**

## Comment ça marche

### Une Servlet à déclarer dans le web.xml

```
<servlet>
  <display-name>FacesServlet</display-name>
  <servlet-name>FacesServlet</servlet-name>
  <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>FacesServlet</servlet-name>
  <url-pattern>/faces/*</url-pattern>
</servlet-mapping>
```

Lors de la soumission d'un formulaire dans l'interface :

- la servlet FacesServlet intercepte la requête et crée un FacesContext contenant, entre autre, les contextes de session et de requête ainsi que l'arbre de l'UI.
- FacesServlet délègue alors le traitement de la requête à un objet LifeCycle
- LifeCycle traite la requête en modifiant les données de FacesContext et en appelant des gestionnaires d'événements
- La réponse produite par LifeCycle est retransmise au client.

## Comment ça marche

**Notes :**

## Présentation de JSF 2.0

### Version mise à disposition avec Java EE 6

- Disponible depuis décembre 2009
- Fournir un standard JEE, spécifié dans la JSR 314, pour le développement d'IHM riches
- Maximiser la productivité des développeurs
  - Fournir les fonctionnalités récurrentes et avancées :
    - n Validation
    - n Conversions
    - n Ajax
    - n etc...
- Masquer la complexité

# Présentation de JSF 2.0

**Notes :**

## Les nouveautés

### Le remplacement de configuration xml par des annotations

- Les principales annotations de JSF2 :
  - Définition des Managed Bean : `@ManagedBean(name=" ", eager=true)`
  - `@RequestScoped`
  - `@SessionScoped`
  - `@ApplicationScoped`
  - `@ViewScoped` : La même instance du bean est utilisée tant que le même utilisateur est sur la même page (ex : avec AJAX).
  - `@CustomScoped (value=« #{someMap} »)` : Le bean est stocké dans un tableau, et le programmeur peut contrôler son cycle de vie.
  - `@NoneScoped` : N'est pas stocké dans un scope, utile s'il est référencé par d'autres beans.
  - `@FacesComponent`
  - `@FacesRenderer`
  - `@FacesConverter`
  - `@FacesValidator`

## Les nouveautés

### Notes :

Le fichier faces-config.xml est toujours obligatoire, cependant il peut être vide.

## Les nouveautés

### Utilisation des facelets comme technologie de présentation coté client

- Les pages jsp ne sont plus autorisées
- Toutes les pages portent l'extension .xhtml
- Notez que l'url est .JSF alors que l'extension des fichiers est .XHTML



## **Les nouveautés**

**Notes :**

## Les nouveautés

### Intégration native d'Ajax : plus besoin de librairies RichFaces, IceFaces,... pour faire de l'Ajax

- JSF2 apporte le support Ajax par le tag `<f:ajax render=" " execute=" ">` :

**render peut valoir un élément ou une liste d'éléments séparés par un espace, et quatre valeurs spéciales sont autorisées :**

- `@this` : n'affecte que ce composant (qui a déclenché la requête Ajax)
- `@form` : affecte tous les composants de la feuille qui a déclenché la requête
- `@none` : n'affecte aucun composant (par défaut si render pas présent)
- `@all` : affecte tous les composants de la vue
- EL : une Expression langage qui renvoie une String

**execute peut valoir un élément ou une liste d'éléments séparés par un espace qui seront envoyés au serveur :**

- `@this` : L'élément contenant `f:ajax`, valeur par défaut
- `@form` : Le formulaire contenant `f:ajax`, permet d'envoyer tous les éléments de la feuille, très utile
- `@none` : Rien n'est envoyé
- `@all` : Envoi de tous les éléments de la page
- Les versions supérieures des librairies RichFaces, IceFaces, PrimeFaces,... sont supportées

## Les nouveautés

### Notes :

- Amélioration des performances par l'utilisation du partial processing
- L'attribut onevent, permet d'exécuter une fonction JavaScript

## Les serveurs d'application

### Les serveurs d'application compatibles

- Les serveurs certifiés JEE 6 :
  - Glassfish 3
  - Jboss 6 +
  - Oracle WebLogic Server 12c
  - IBM WebSphere Application Server 8.0
  - ...
- Les serveurs supportant les servlet 2.5 ou supérieur
  - Tomcat 6+
  - Jetty 6+
  - Google App Engine
  - JBoss 5+
  - WebSphere 6+
  - WebLogic 9+

## Les serveurs d'application

### Notes :

Dans un serveur d'application JEE6, il n'est pas nécessaire d'ajouter les jars JSF2 dans le classpath de l'application web, en revanche, il est obligatoire d'ajouter ces jars dans le répertoire WEB-INF/lib de l'application web pour les autres serveurs.

## **L'intégration de JSF au serveur**

### **Pour les serveurs supportant les Servlet 2.5**

- Ajouter les librairies et fichiers suivants :
  - jsf-api.jar, jsf-impl.jar
  - les librairie JSTL 1.2 téléchargeables depuis Oracle Mojarra ou Apache MyFaces
  - web.xml
  - faces-config.xml

### **Pour tous les serveurs certifiés JEE 6**

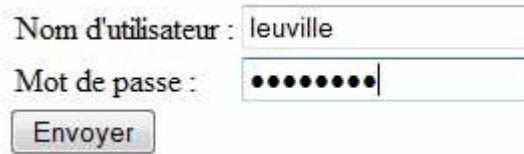
- Contiennent les librairies requises
- Ajouter dans WEB-INF
  - web.xml
  - faces-config.xml, ce fichier peut être vide

## **Les nouveautés**

**Notes :**

## Exemple d'application

### Exemple de page de login



Nom d'utilisateur : leuville  
Mot de passe : .....  
Envoyer

- Déclaration de l'utilisation de JSF

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core">
```

```
</html>
```

- Définition de la page de login

```
<h:body>
<h:form>
  <h:panelGrid cellpadding="2" columns="2">
    Nom d'utilisateur : <h:inputText value="#{user.utilisateur}" />
    Mot de passe : <h:inputSecret value="#{user.password}" />
  </h:panelGrid>
  <h:commandButton value="Envoyer" action="reponse.xhtml" />
</h:form>
</h:body>
```



## Exemple d'application

**Notes :**

## Exemple d'application

### Définition du Managed Bean

```
package com.leuville;

import java.io.Serializable;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;

@ManagedBean(name="user")
@SessionScoped
public class ExempleBean implements Serializable {
    private String utilisateur;

    private String password;

    public String getPassword() { return password; }

    public void setPassword(String password) { this.password = password; }

    public String getUtilisateur() { return utilisateur; }

    public void setUtilisateur(String utilisateur) { this.utilisateur = utilisateur; }
}
```

## **Exemple d'application**

**Notes :**

# *Java EE - JavaServer Faces*

## *Cycle de vie d'une requête JSF et Evènements*

---

*Version 1.0*

- Cycle de vie d'une requête JSF
- Les différents événements rencontrés

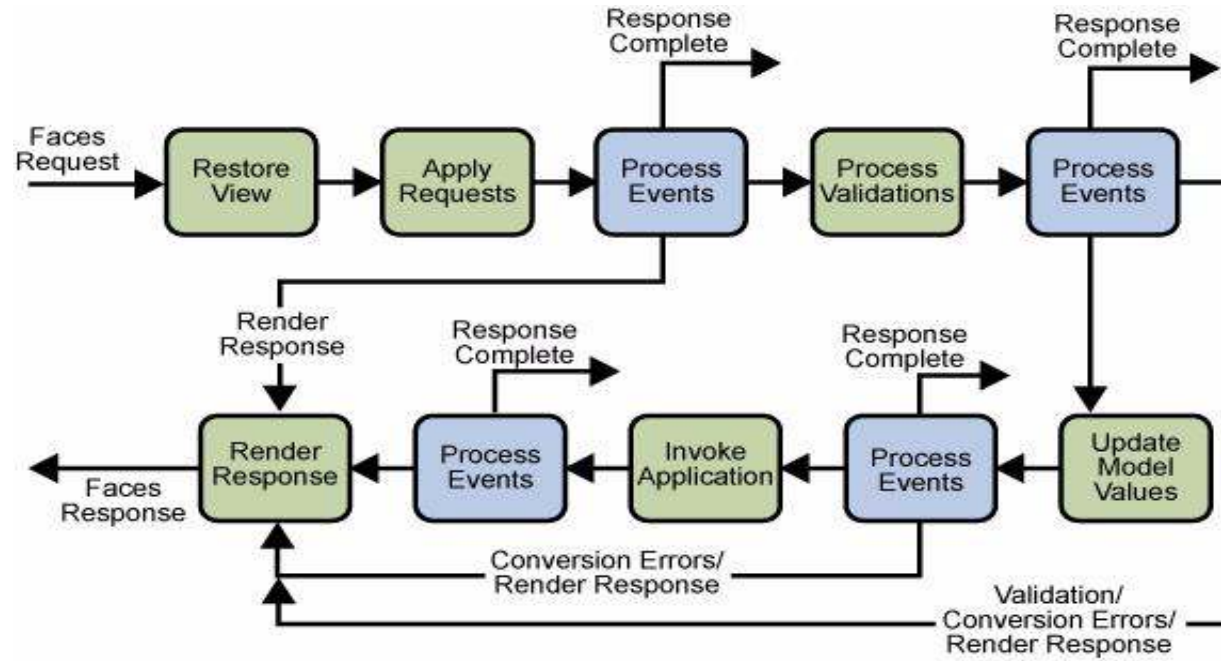
(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects, est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).



## Cycle de vie d'une requête JSF

- Le traitement de la requête est effectuée par une instance de "javax.faces.lifecycle.Lifecycle"



- Le traitement comporte 6 phases
- Chaque phase peut être suivie d'une sous-phase de gestion d'événements

# Cycle de vie d'une requête JSF

## Notes

- **Restore View** : Phase de reconstruction de l'arbre des composants UI. Une JSP dans une application JSF est un ensemble de composants organisés de façon hiérarchique. L'arbre constitué est stocké dans le FacesContext.
- **Apply Request Values** : Mise à jour des données contenues dans les composants à partir des données contenues dans la requête (paramètres, en-têtes, cookies, ...).
- **Process Validations** : Validation des données contenues dans les composants. Toutes les validations des données traitées à l'étape précédentes sont exécutées. Les données sont converties dans le bon type Java, si une validation échoue, la main est donnée à la phase de rendu de la réponse.
  - Chaque composant valide et convertit les données qu'il contient. Si un composant détecte une valeur non valable, il met sa propriété « valid » à false et il met un message d'erreur dans la file d'attente des messages (ils seront affichés lors de la phase de rendu « render response ») et les phases suivantes sont sautées
- **Update Model Values** : Dans le cas où la phase de validation s'est correctement déroulée, cette phase permet de mettre à jour les données de l'application (JavaBeans de la couche modèle).
- **Invoke Application** : Cette phase consiste à invoquer le code applicatif traitant la logique métier. Les actions associées aux boutons ou aux liens sont exécutées.

**Render Response** : La réponse est générée et envoyée au client;

## Cycle de vie d'une requête JSF

### Bésoin de sauter des phases dans le cycle de vie

- Présence d'un bouton Annuler dans le formulaire
- Génération d'un fichier PDF par exemple

### Utilisation de l'attribut `immediate=true` sur un `UICommand`

- Ajouté à un bouton ou un lien : `<h:commandButton>`, `<h:commandLink>` permet de passer directement de la phase **Apply Request Values** à la phase **Invoke Application** en sautant les phases de validation et de mise à jour du modèle

### Utilisation de l'attribut `immediate=true` sur un `EditableValueHolder`

- Ajouté sur une liste déroulante ou des cases à cocher, permet de déclencher immédiatement la validation et la conversion de la valeur qu'il contient, avant la validation et la conversion des autres composants de la page



## Cycle de vie d'une requête JSF

### Notes :

Il est quelquefois indispensable de sauter des phases du cycle de vie.

Par exemple, si l'utilisateur clique sur un bouton d'annulation, on ne veut pas que la validation des champs de saisie soit effectuée, ni que les valeurs actuelles soient mises dans le modèle

Autre exemple : si on génère un fichier PDF à renvoyer à l'utilisateur et qu'on l'envoie à l'utilisateur directement sur le flot de sortie de la réponse HTTP, on ne veut pas que la phase de rendu habituelle soit exécutée

## Les différents événements rencontrés

### Trois types d'événements disponibles

- Value change event : Associés à des composants de type input (inputText, etc)
- Action event : Associés à des composants de commande (commandButton, etc)
- Phase event : Evènements déclenchés régulièrement dans le cycle de vie de JSF

## Les différents événements rencontrés

**Notes :**

## L'événement 'value change'

Permet de surveiller la modification d'un composant

Peut permettre de rendre dépendant les autres composants

Please fill in your address

Adress   
 City   
 Province/Region   
 Country

Submit address

Veuillez saisir votre adresse

Adresse   
 Ville   
 Province/Région   
 Pays

Envoyer l'adresse

```
public void paysChanged(ValueChangeEvent event){
    FacesContext context = FacesContext.getCurrentInstance();
    if(US.equals((String)event.getNewValue()))
        context.getViewRoot().setLocale(Locale.US);
    else if(CANADA.equals((String)event.getNewValue()))
        context.getViewRoot().setLocale(Locale.CANADA);
    else
        context.getViewRoot().setLocale(Locale.FRANCE);
}
```

```
<h:selectOneMenu value="#{form.pays}" onchange="submit()"
    valueChangeListener="#{form.paysChanged}">
```

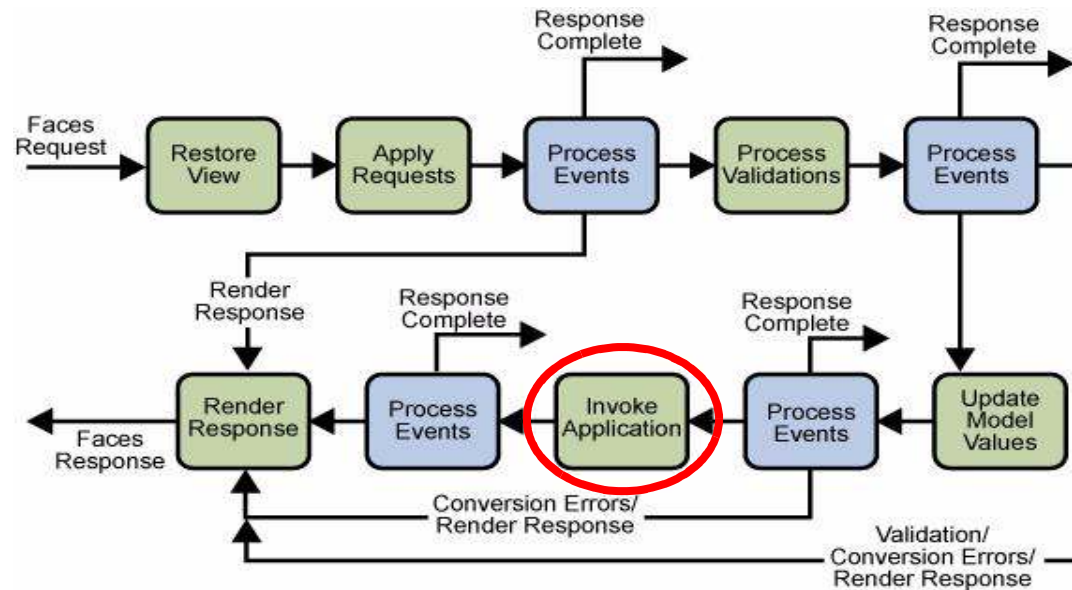
## **L'événement 'value change'**

**Notes :**

## L'événement 'action'

## Déclenché par des composants de type bouton, liens, etc

## Appelés lors de la phase d'invocation de l'application



## **L'événement 'action'**

**Notes :**

## Les balises d'écoute d'événements

### Solution alternative à la définition d'événements dans d'autres balises

#### Deux balises existantes (de la librairie core)

- `actionListener`
- `valueChangeListener`

#### Placées dans le corps des balises

```
<h:selectOneMenu value="#{form.pays}" onchange="submit()"
                  valueChangeListener="#{form.paysChanged}">
    Devient
    <h:selectOneMenu value="#{form.pays}" onchange="submit()" >
        <f:valueChangeListener type="#{form.paysChanged}">
        ...
    </h:selectOneMenu>
```

#### Permet d'attacher plusieurs listener à un composant



## Les balises d'écoute d'événements

**Notes :**

## Les écouteurs de phase

### Evénements déclenchés avant et après chaque phase du cycle de vie

#### Définit dans le fichier faces-config.xml

- Utilisation de la balise *phase-listener* balise fille de *lifecycle*
- La classe doit implémenter l'interface `javax.event.PhaseListener` qui définit trois méthodes :
  - `getPhaseId()` : renvoie un objet de type `PhaseId` qui permet de préciser à quelle phase ce listener correspond
  - `beforePhase()` : traitements à exécuter avant l'exécution de la phase
  - `afterPhase()` : traitements à exécuter après l'exécution de la phase

## Les écouteurs de phase

### Notes :

La classe `PhaseId` définit des constantes permettant d'identifier chacune des phases :  
`PhaseId.RESTORE_VIEW`, `PhaseId.APPLY_REQUEST_VALUES`, `PhaseId.PROCESS_VALIDATIONS`,  
`PhaseId.UPDATE_MODEL_VALUES`, `PhaseId.INVOKE_APPLICATION` et  
`PhaseId.RENDER_RESPONSE`

Elle définit aussi la constante `PhaseId.ANY_PHASE` qui permet de demander l'application du listener à toutes les phases. Cela peut être très utile lors du débogage.

# *Java EE - JavaServer Faces*

## *La navigation entre pages*

---

*Version 1.0*

- Définition des règles de navigation
- Aspects avancés de la navigation

(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects, est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).



## Définition des règles de navigation

### Deux types de navigation possible

- Navigation statique
- Navigation dynamique

### Utilisation du fichier *faces-config.xml* pour effectuer cela

- Utilisation de la balise navigation-rule

### Utilisation d'un littéral

## Définition des règles de navigation

**Notes :**

## Navigation statique

### Utile dans certaines situations

### Navigation entre les pages prédéfinie

```
<h:commandButton value="Login" action="login" />
<navigation-rule>
  <from-view-id>/welcomeJSF.xhtml</from-view-id>
  <navigation-case>
    <from-outcome>login</from-outcome>
    <to-view-id>/welcome.xhtml</to-view-id>
  </navigation-case>
</navigation-rule>
```

- Définition du comportement dans le cas d'une action (ici login) : balise *from-outcome*
- Définition du nom de la page qui doit être à l'origine de l'action (ici welcomeJSF.xhtml) : balise *from-view-id*
- Définition de la page qui doit-être affichée (ici welcome.xhtml) : balise *to-view-id*

### Utilisation d'un littéral page-b

- la navigation sera vers la page page-b.xhtml
- `<h:commandButton value="Go to Page B" action="page-b" />`



# Navigation statique

**Notes :**

## Navigation dynamique

**Permet plus de souplesse dans l'enchaînement des pages**

**Utilisation de *method expression* pour définir dynamiquement les valeurs des actions**

```
<h:commandButton value="Réponse" action="#{quiz.reponseAction}" /
```

- responseAction est une méthode retournant la valeur de l'action depuis le managed bean "quiz"

**L'ensemble des cas possibles (actions) doivent-être définis dans faces-config.xml**

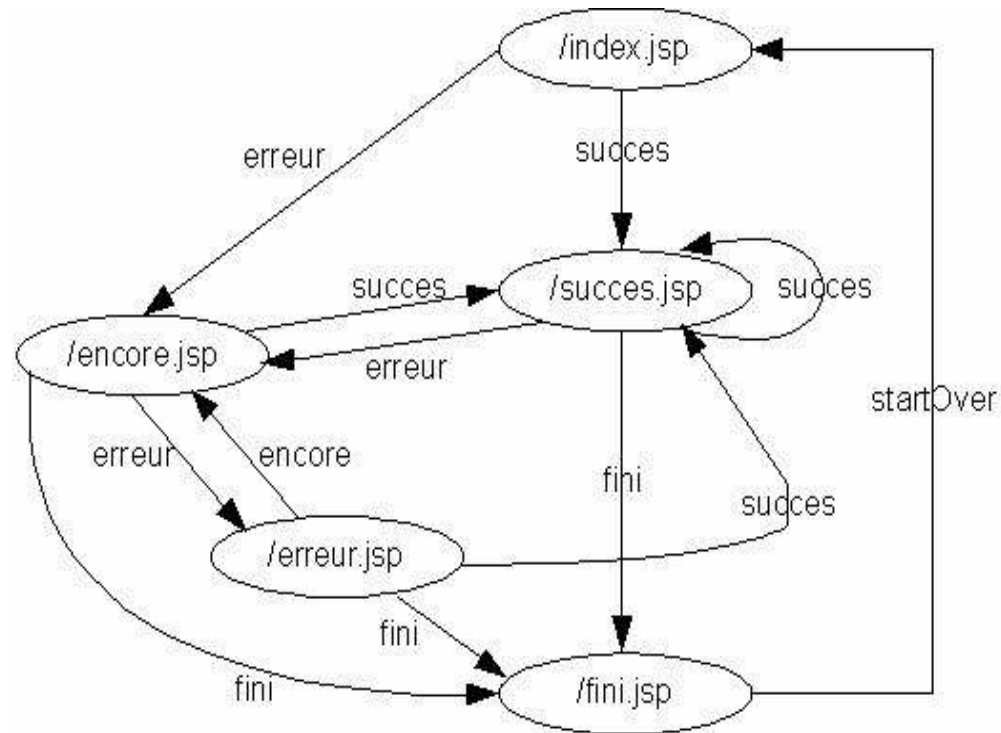
- Utilisation de balise *navigation-case*

# Navigation dynamique

**Notes :**

## Exemple de navigation

### Mise au point du diagramme de navigation



**Permet de déterminer comment vont pouvoir être appelées les pages**

**Permet de définir les éléments à déclarer dans le fichier `faces-config.xml`**

## Exemple de navigation

**Notes :**

## Exemple de navigation

**Pour la mise au point du fichier faces-config.xml on peut constater que l'on doit définir les actions**

- succes : La réponse est valide
- encore : Pour recommencer
- erreur : La réponse est fausse
- fini : Toutes les réponse ont été validées
- startOver : On souhaite recommencer le quiz

```
<navigation-rule>
  <navigation-case>
    <from-outcome>succes</from-outcome>
    <to-view-id>/succes.jsp</to-view-id>
    <redirect />
  </navigation-case>
  <navigation-case>
    <from-outcome>encore</from-outcome>
    <to-view-id>/encore.jsp</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-outcome>erreur</from-outcome>
```

```
    <to-view-id>/erreur.jsp</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-outcome>fini</from-outcome>
    <to-view-id>/fini.jsp</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-outcome>startOver</from-outcome>
    <to-view-id>/welcome.jsp</to-view-id>
  </navigation-case>
</navigation-rule>
```

## Exemple de navigation

**Notes :**

## Définition de la page index.xhtml

### Va définir comment va être appelée la prochaine page

`<h:commandButton value="#{msg.reponseBouton}" action="#{quiz.reponseAction}" /`

- Utilisation de l'attribut action (utilisation d'une *method expression*)
- La valeur de l'attribut dépend de la justesse de la réponse ou non

```
public String reponseAction(){  
    essais++;  
    if(problemes[indexCourant].isCorrect(reponse)){  
        score++;  
  
        if(indexCourant == problemes.length - 1){  
            return "fini";  
        } else {  
            nextProblem();  
            return "succes";  
        }  
    }  
    } else if (essais == 1){  
        return "encore";  
    } else {  
        if(indexCourant == problemes.length - 1){  
            return "fini";  
        } else {  
            nextProblem();  
            return "erreur";  
        }  
    }  
}
```



## Définition de la page index.xhtml

**Notes :**

# *Java et Web 2.0*

## *Les Facelets*

---

*Version 1.0*

- Présentation des Facelets
- Utilisation de Facelets dans une page JSF
- Les templates

(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).



## Présentation des facelets

**Solution de remplacement à l'utilisation des pages JSP**

**Permet d'implémenter les vues au format XHTML (eXtensible HTML)**

**Offre la possibilité de créer des composants personnalisés sans écrire de code**

**Offre la possibilité d'utiliser un système de template**

**Il faut spécifier que l'on utilise Facelet comme view handler dans faces-config.xml**

```
<application>
  <view-handler>com.sun.facelets.FaceletViewHandler</view-handle
</application>
```

**Nécessite d'ajouter des paramètres d'initialisation dans le fichier web.xml**

```
<context-param>
  <param-name>javax.faces.DEFAULT_SUFFIX</param-name>
  <param-value>.xhtml</param-value>
</context-param>
```

---

Il faut avoir placé les deux fichiers XML dans le répertoire

---

---

Cette configuration n'est pas nécessaire dans JSF2

---

# Présentation des facelets

## Notes

## Exemple de page de connexion avec Facelet

### Mise en place d'une page de login

Nom Leuville

Mot de passe \*\*\*\*

Connexion

Bienvenu, Leuville

[Retour...](#)

### Définition de namespaces à utiliser au niveau de la balise html

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:h="http://java.sun.com/jsf/html">
  ...
</html>
```

### Chargement d'un bundle effectué directement dans la page

```
<f:loadBundle basename="com.leuville.messages" var="msgs"
```

### Remplacement de la balise par le composant JSF en utilisant l'attribut *jsfc*

- Permet de définir le composant JSF qui va être utilisé

## Exemple de page de connexion avec Facelet

### Notes

## Utilisation de balises JSF avec Facelets

### Pourquoi utiliser des balises JSF avec Facelets

- Facelets est plus orienté *designer*
- XHTML est verbeux et moins maintenable
- Plus proche de l'écriture d'une page JSP pour un développeur

### L'utilisation de balises JSF revient à renoncer à l'approche offerte par jsfc



# Utilisation de balises JSF avec Facelets

## Notes

## Composition de pages avec des templates

### Permet de définir des zones dans une page

- Utilisation de balises insert

```
...
xmlns:ui="http://java.sun.com/jsf/facelets"
...
<div class="header">
    <ui:insert name="header"/>
</div>

<div class="menu">
    <ui:insert name="menu"/>
</div>

<div class="content">
    <ui:insert name="content"/>
</div>
```

- Vont être remplacées lors de l'utilisation du layout

```
<ui:composition template="/layout.xhtml">
```

# Composition de pages avec des templates

## Notes

## Composition de pages avec des templates

**Il faut ensuite remplacer les éléments représentés par la balise `insert`**

- Utilisation de la balise *define*

```
<ui:define name="header">  
    
</ui:define>
```

# Composition de pages avec des templates

## Notes

# ***Java et Web 2.0***

## ***Les librairies de balises JSF***

---

*Version 1.0*

- Les librairies core et HTML

(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).



## La librairie core

### 20 balises différentes

- La balise *view* représente le niveau de base de la vue dans lequel on va placer l'ensemble des éléments JSF
- La balise *subview* est une sous vue qui se trouve dans la vue
- La balise *attribute* définit un ensemble clé/valeur dans sa balise parente
- La balise *param* permet d'ajouter un élément paramètre à une balise
- La balise *facet* ajoute une facette à un composant
- La balise *actionListener* ajoute un "écouteur d'événement" à un composant
- La balise *valueChangeListener* ajoute un "écouteur de changement de valeur" à un composant
- La balise *phaseListener* ajoute un écouteur de phase (du cycle de vie)
- La balise *converter* ajoute un convertisseur temporaire à un composant



## La librairie core

**Notes :**

## La librairie core

### 20 balises différentes (suite)

- La balise *convertDateTime* ajoute un convertisseur de date à un composant
- La balise *convertNumber* ajoute un convertisseur numérique à un composant
- La balise *validator* ajoute un valideur à un composant
- La balise *validateDoubleRange* valide une donnée de type double
- La balise *validateLength* valide la longueur d'une chaîne de caractères
- La balise *validateLongRange* valide la longueur d'une variable de type long
- La balise *loadBundle* charge un *resource bundle* et range les propriétés comme une *Map*
- La balise *selectitems* définit les *items* pour un selecteur simple ou multiple d'un composant
- La balise *selectitem* définit un *item* pour un selecteur simple ou multiple d'un composant
- La balise *verbatim* transforme un texte contenant des balises en composant

## La librairie core

**Notes :**

## La librairie HTML

**25 balises différentes qui représentent les types de composants suivants :**

- Entrées
- Sorties
- Commandes
- Sélection
- Autres cas (messages, etc)

# La librairie HTML

**Notes :**

## La librairie HTML

La balise *form* correspond à la ré-écriture de la balise form HTML

### La balise inputText

```
<h:inputText value="1234567" readonly="true" />
```

```
<h:inputText value="inputText" style="background: Teal;color: white" />
```

```
<h:inputText value="1234567" maxlength="6" size="10" />
```

### La balise inputTextArea

```
<h:inputTextarea rows="5"/>
```

```
<h:inputTextarea rows="3" cols="10" value="01234567890123456"/>
```

### La balise inputSecret

```
<h:inputSecret value="password" redisplay="true" />
```

```
<h:inputSecret value="password" redisplay="false" />
```

# La librairie HTML

**Notes :**

## La librairie HTML

La balise *inputHidden* correspond à un champs caché

```
<h:inputHidden value="hidden" />
```

La balise *outputLabel* permet d'afficher un texte

```
<h:outputLabel value="output label" />
```

La balise *outputLink* correspond à un lien hypertexte HTML



```
<h:outputLink value="http://www.leuville.com/">  
  <h:graphicImage value="/logo.png" />  
</h:outputLink>
```

La balise *outputFormat* est une balise identique à *outputText*

- Permet de construire un texte composé
- Associé à la balise *param*

La balise *outputText*

```
Un outputText  <h:outputText value="Un outputText" />
```



# La librairie HTML

**Notes :**

## La librairie HTML

La balise *commandButton* est utilisée pour représenter les boutons HTML **submit** et **reset**



```
<h:commandButton value="submit" type="submit" />
```



```
<h:commandButton value="reset" type="reset" />
```



```
<h:commandButton value="Clicker sur le bouton"
type="button" onclick="alert('button clicked')" />
```

La balise *commandLink* est un lien hypertexte qui se comporte comme un *pushButton*

[un lien avec commandLink](#)

```
<h:commandLink>
```

```
<h:outputText value="un lien avec commandLink" />
```

```
</h:commandLink>
```

```
<h:commandLink>
```

```
<h:outputText value="un lien avec commandLink" />
```

```
<h:graphicImage value="/logo.png" />
```

```
</h:commandLink>
```



[un lien avec commandLink](#)

La balise *message* affiche le dernier message pour un composant

La balise *messages* affiche l'ensemble des messages

# La librairie HTML

**Notes :**

## La librairie HTML

### La balise *graphicImage* affiche une image

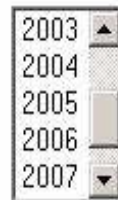


```
<h:graphicImage value="/logo.png" />
```



```
<h:graphicImage value="/logo.png" style="border: thin solid black" />
```

### La balise *selectOneListBox*



```
<h:selectOneListbox value="année" size="5">  
  <f:selectItem itemValue="2000" itemLabel="2000" />  
  <f:selectItem itemValue="2001" itemLabel="2001" />  
  ...  
  <f:selectItem itemValue="2007" itemLabel="2007" />  
  <f:selectItem itemValue="2008" itemLabel="2008" />  
</h:selectOneListbox>
```

# La librairie HTML

**Notes :**

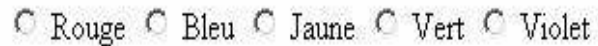
## La librairie HTML

### La balise *selectOneMenu*



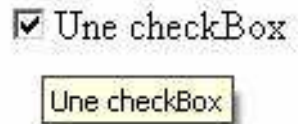
```
<h:selectOneMenu value="Mardi">
  <f:selectItem itemValue="Lundi" itemLabel="Lundi" />
  <f:selectItem itemValue="Mardi" itemLabel="Mardi" />
  ...
  <f:selectItem itemValue="Samedi" itemLabel="Samedi" />
  <f:selectItem itemValue="Dimanche" itemLabel="Dimanche" />
</h:selectOneMenu>
```

### La balise *selectOneRadio*



```
<h:selectOneRadio value="couleur">
  <f:selectItem itemValue="Rouge" itemLabel="Rouge" />
  <f:selectItem itemValue="Bleu" itemLabel="Bleu" />
  <f:selectItem itemValue="Jaune" itemLabel="Jaune" />
  <f:selectItem itemValue="Vert" itemLabel="Vert" />
  <f:selectItem itemValue="Violet" itemLabel="Violet" />
</h:selectOneRadio>
```

### La balise *selectBooleanCheckbox* représente une simple checkbox



```
<h:selectBooleanCheckbox value="true" title="Une checkBox" />
<h:outputText value="Une checkBox" />
```

# La librairie HTML

**Notes :**

## La librairie HTML

### La balise *selectManyCheckbox* représente un ensemble de checkboxes



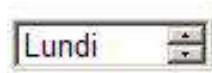
```
<h:selectManyCheckbox value="couleur">
  <f:selectItem itemValue="Rouge" itemLabel="Rouge" />
  ...
  <f:selectItem itemValue="Violet" itemLabel="Violet" />
</h:selectManyCheckbox>
```

### La balise *selectManyListbox* représente une liste à sélection multiple



```
<h:selectManyListbox value="">
  <f:selectItem itemValue="Angleterre" itemLabel="Angleterre" />
  <f:selectItem itemValue="France" itemLabel="France" />
  ...
  <f:selectItem itemValue="Russie" itemLabel="Russie" />
</h:selectManyListbox>
```

### La balise *selectManyMenu* représente un menu à sélection multiple



```
<h:selectManyMenu value="Mardi">
  <f:selectItem itemValue="Lundi" itemLabel="Lundi" />
  <f:selectItem itemValue="Mardi" itemLabel="Mardi" />
  ...
  <f:selectItem itemValue="Samedi" itemLabel="Samedi" />
  <f:selectItem itemValue="Dimanche" itemLabel="Dimanche" />
</h:selectManyMenu>
```

---

Attention, l'affichage obtenu dépend du navigateur

---



# La librairie HTML

## Notes

## La librairie HTML

La balise *panelGrid* représente une table HTML

La balise *panelGroup* permet de regrouper des composants

La balise *dataTable* permet de mettre au point un tableau élaboré

La balise *column* représente une colonne dans une *dataTable*

# La librairie HTML

**Notes :**

# ***Java EE - JavaServer Faces Composants standards***

---

*Version 1.0*

- Les composants standards

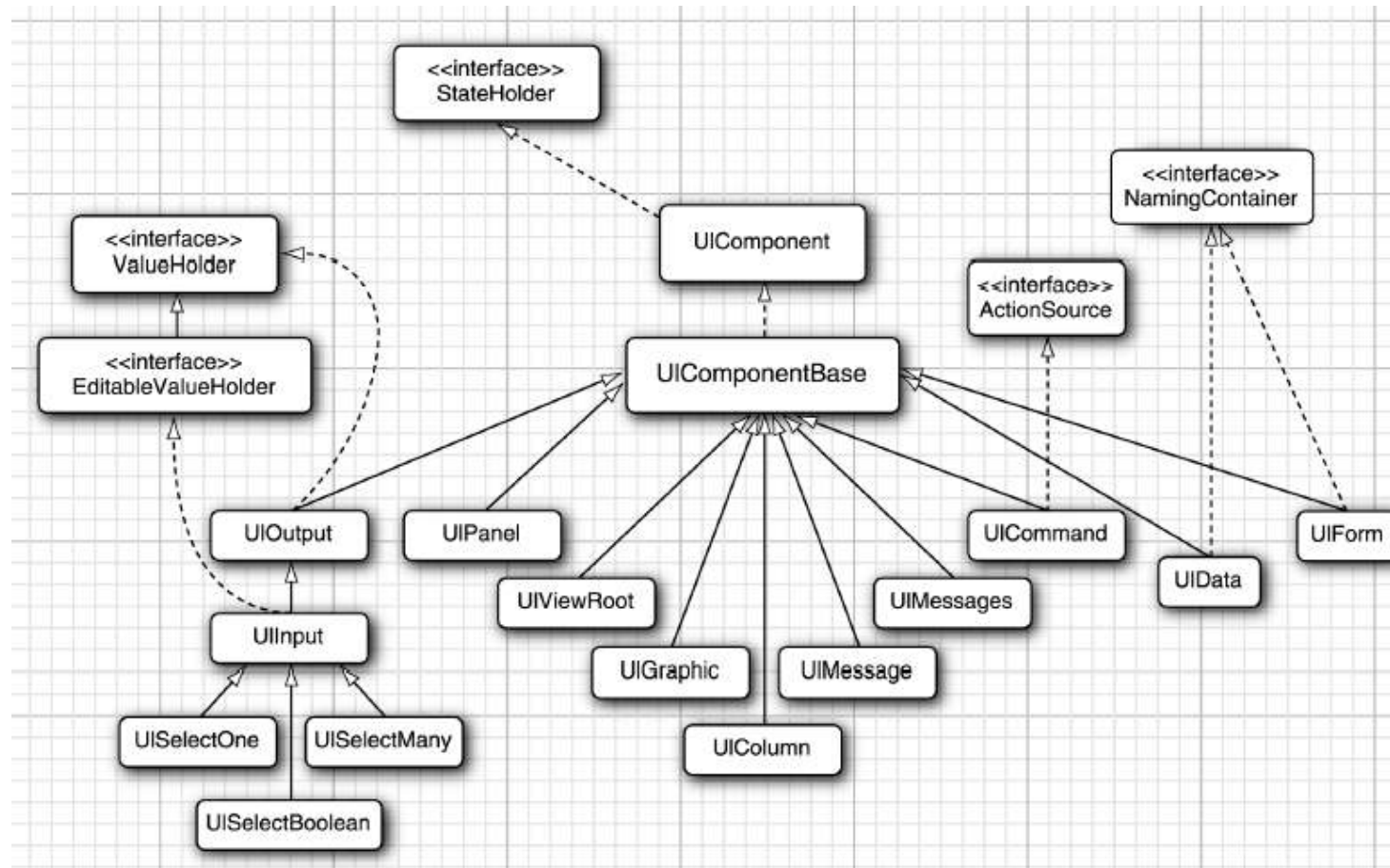
(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects, est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).



## Les composants standards

- Le modèle de composants UI
  - Un composant JSF réfère la classe `UIComponent` qui définit le comportement des classes d'interfaces graphiques



# Les composants standards

**Notes :**

## Les composants standards

- Ces composants sont des javabeans

**Table 2: Composants standards**

Composant	Description
UICommand	Composant permettant de réaliser une action
UIForm	Composant qui contient d'autres composants dont l'état sera envoyé au serveur
UIGraphic	Représente une image
UIInput	Pour la saisie des données
UIPanel	Composant qui regroupe d'autres composants sous forme de tableau
UIParameter	Composant permettant de passer des paramètres à un composant
UISelectItem	Composant représentant l'élément sélectionné parmi un ensemble d'items
UISelectItems	Composant représentant un ensemble d'éléments Item
UISelectBoolean	Composant permettant de sélectionner parmi deux états possibles
UISelectMany	Composant permettant de sélectionner plusieurs éléments parmi un ensemble
UISelectOne	Composant permettant de sélectionner un seul élément parmi un ensemble
UIOutput	Composant permettant d'afficher des données



## Les composants standards

### Notes :

Les composants UI sont gérés par un ViewHandler entre chaque requête, et le StateManager gère l'état de chaque composant. L'état de chaque composant peut être géré sur le client. Ce choix est à faire dans le fichier WEB.XML, en modifiant la valeur de `javax.faces.STATE_SAVING_METHOD`

Par défaut, l'état est géré sur le serveur.

## Les composants standards

- Le composant UI est associé à d'autres composants.
- Pas de code de rendu spécifique pour un client
- Associé à une classe `Renderer` qui encapsule les spécificités du client, et est utilisée autant pour encoder ou décoder
- `UIComponent` et `Renderer` sont donc indépendants.
- Un TLD sert de lien entre `UIComponent` et le `Renderer`
- Hormis ces classes, des classes telles que `Converters`, `Validators`, `ActionListeners` peuvent être liées.

# Les composants standards

**Notes :**

# *Java EE - JavaServer Faces*

## *Composites components*

---

*Version 1.0*

- Présentation
- Création de composants composés

(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects, est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).



## Présentation

### JSF 2 ajoute les *composites components* :

- Réutilisables, constitués uniquement de code facelet, pas de code Java
- Ils utilisent des balises JSF et le langage d'expression EL de JSF
- Des réels composants

### Nécessite de placer le fichier du composant (par exemple `monComposant.xhtml`) :

- Dans un sous répertoire de ressources, par exemple : `resources/leuville-cc`
- Utiliser le namespace : `<html...xmlns:composite=« http://java.sun.com/jsf/composite »>`
- Déclarer les éventuels attributs dans la balise `composite:interface`
- Composer la sortie dans la balise `composite:implementation`

### Utilisation de ce composant dans une page facelet

- Utiliser le namespace : `<html ... xmlns:leuville-cc="http://java.sun.com/jsf/composite/leuville-cc">`
- Utiliser le composant : `<leuville-cc:monComposant.../>`

# Présentation

**Notes :**

## Création de composant composé

### monComposant.xhtml : Le composant peut utiliser des attributs

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:composite="http://java.sun.com/jsf/composite"
      xmlns:ui="http://java.sun.com/jsf/facelets">
<head><title>My composite component</title></head>
<body>

<composite:interface>
  <composite:attribute name="msg1" required="true"/>
  <composite:attribute name="msg2" default="Hi from Leuville"/>
</composite:interface>

<composite:implementation>
  <h2>Message 1: "#{cc.attrs.msg1}"</h2>
  <h2>Message 2: "#{cc.attrs.msg2}"</h2>
</composite:implementation>
</body>
</html>
```

**Ce fichier doit être placé dans un sous répertoire de ressources, par exemple: leuville-cc**

**cc.attrs permet d'accéder aux attributs définis dans la balise *composite:interface***



## Création de composant composé

### Notes :

Il est possible, dans la balise `composite:attribute` de faire référence à un managed bean, et d'accéder à ses propriétés dans la balise `composite:implementation`

## Création de composant composé

### La page Facelet utilisant ce composant

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:orsys="http://java.sun.com/jsf/composite/leuville-cc">
<h:head><title>Composite Components</title>
</h:head>
<h:body>
<orsys:monComposant msg1="Hello, world"/>
</body>
</html>
```

### Possibilité de faire de l'imbrication de composants

Pour accéder à un attribut du parent, utiliser cc.parent.attrs

Mojarra 2.0.3 ou supérieur est obligatoire

# Création de composant composé

**Notes:**

# *Java EE - JavaServer Faces*

## *JSF et les événements*

---

*Version 1.0*

- Cycle de vie des événements
- Les balises HTML
- Les templates
- Les balises d'écoute des événements

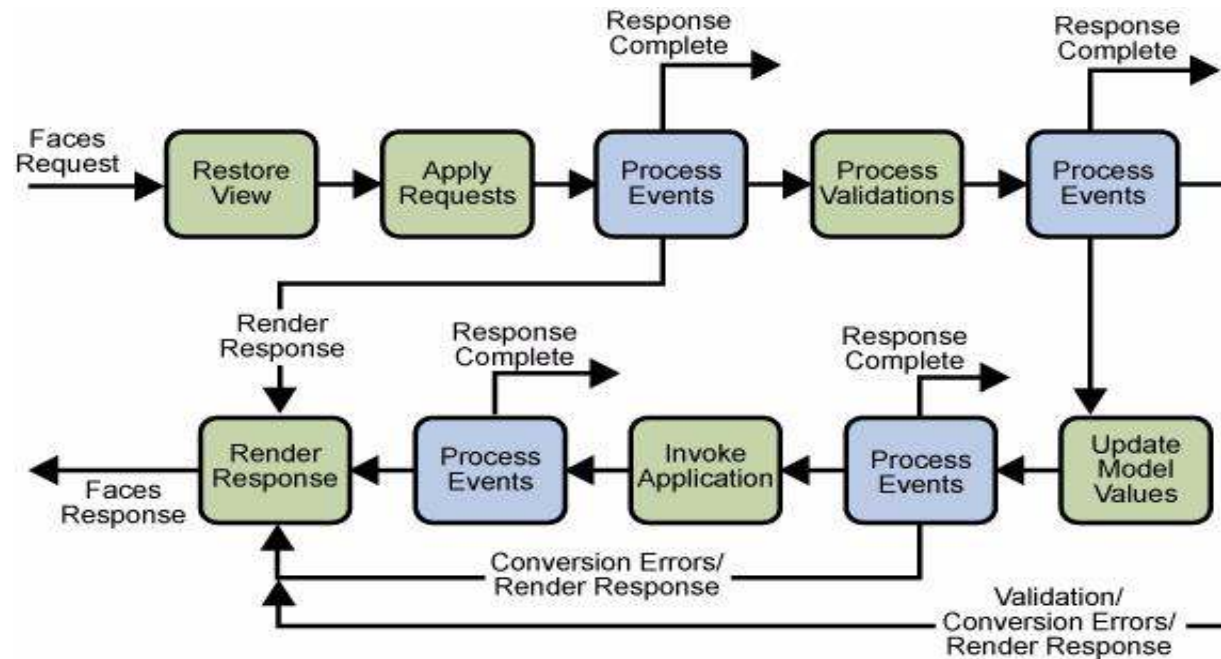
(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects, est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).



## Cycle de vie des événements

Sont associés à plusieurs étapes du cycle de vie



### Trois types d'événements disponibles

- Value change event : Associés à des composants de type input (inputText, etc)
- Action event : Associés à des composants de commande (commandButton, etc)
- Phase event : Événement déclenchés régulièrement dans le cycle de vie de JSF

# Cycle de vie des événements

## Notes

## L'événement 'value change'

**Permet de surveiller la modification d'un composant**

**Peut permettre de rendre dépendant les autres composants**

Please fill in your address		Veuillez saisir votre adresse	
Adress	<input type="text"/>	Adresse	<input type="text"/>
City	<input type="text"/>	Ville	<input type="text"/>
Province/Region	<input type="text"/>	Province/Région	<input type="text"/>
Country	<input type="text" value="United States"/>	Pays	<input type="text" value="France"/>
<input type="button" value="Submit adress"/>		<input type="button" value="Envoyer l'adresse"/>	

```
public void paysChanged(ValueChangeEvent event){
    FacesContext context= FacesContext.getCurrentInstance();
    if(US.equals((String)event.getNewValue()))
        context.getViewRoot().setLocale(Locale.US);
    else if(CANADA.equals((String)event.getNewValue()))
        context.getViewRoot().setLocale(Locale.CANADA);
    else
        context.getViewRoot().setLocale(Locale.FRANCE);
}
```

```
<h:selectOneMenu value="#{form.pays}" onchange="submit()"
    valueChangeListener="#{form.paysChanged}">
```



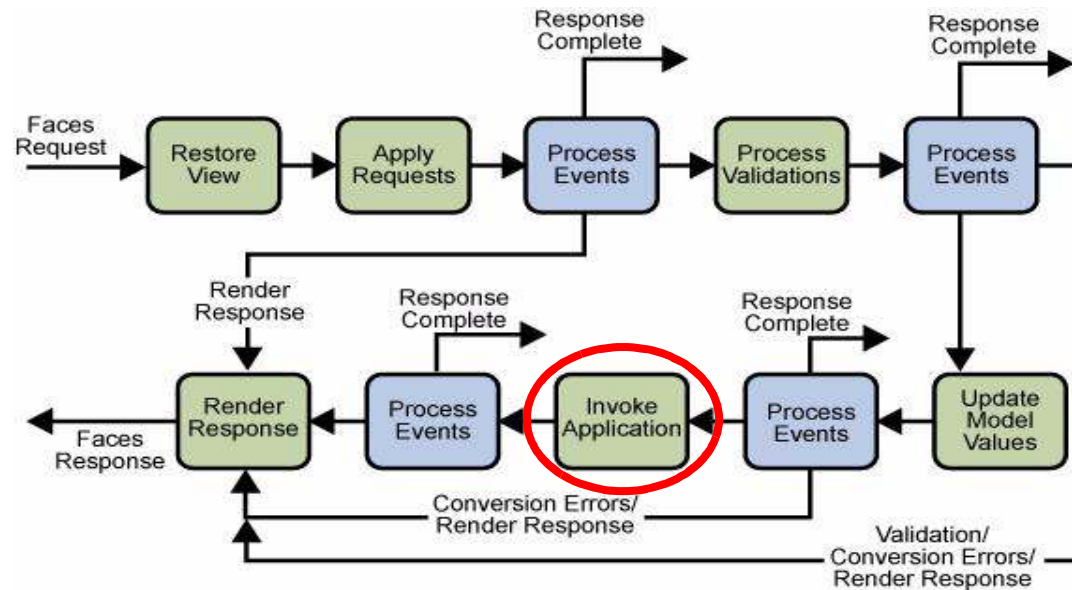
## L'événement 'value change'

**Notes :**

## L'événement 'action'

Déclenché par des composants de type bouton, liens, etc

Appelés lors de la phase d'invocation de l'application



## **L'événement 'action'**

**Notes :**

## Les balises d'écoute d'événements

### Solution alternative à la définition d'événements dans d'autres balises

#### Deux balises existantes (de la librairie core)

- `actionListener`
- `valueChangeListener`

#### Placées dans le corps des balises

```
<h:selectOneMenu value="#{form.pays}" onchange="submit()"
                 valueChangeListener="#{form.paysChanged}">
    Devient
    <h:selectOneMenu value="#{form.pays}" onchange="submit()" >
        <f:valueChangeListener type="#{form.paysChangedType}">
        ...
    </h:selectOneMenu>
```

#### Permet d'attacher plusieurs listener à un composant

# Les balises d'écoute d'événements

**Notes :**

## Les écouteurs de phase

**Événements déclenchés avant et après chaque phase du cycle de vie**

**Définit dans le fichier faces-config.xml**

- Utilisation de la balise *lifecycle*

# Les écouteurs de phase

**Notes :**

# ***Java EE - JavaServer Faces***

## ***Conversion et validation***

---

*Version 1.0*

- Présentation du processus de conversion et de validation
- Utilisation de la conversion standard
- Utilisation de la validation standard
- Utilisation de validations et conversions personnalisées

(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

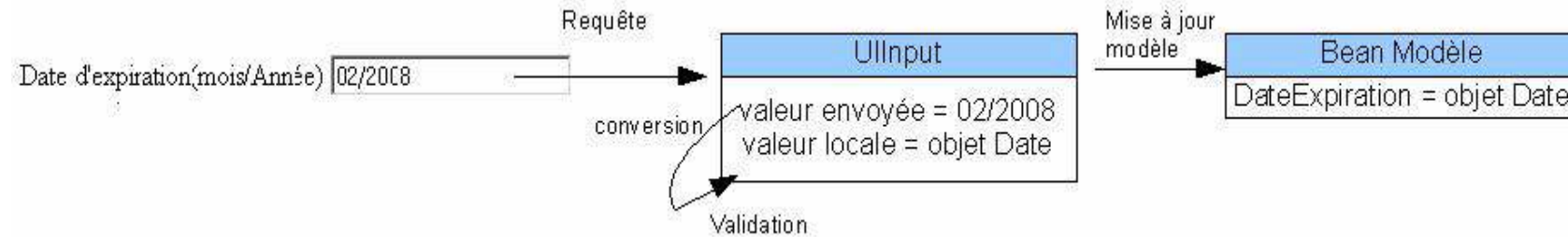
Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects, est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).





## Processus de conversion et de validation

### Plusieurs phases entre le navigateur et le model (bean)



- La conversion de la donnée saisie dans un type attendu
- La validation de la valeur dans le type locale (si besoin)
- La mise à jour du bean (modèle)

## **Processus de conversion et de validation**

**Notes :**

## Utilisation de la conversion standard

### Permet de transformer une valeur dans un format spécifique

- Conversion de nombres : `convertNumber`
- Conversion de date : `convertDateTime`

### Prise en compte des données locales

**veuillez saisir les données de paiement**

Montant :	<input type="text" value="100,00 €"/>
Carte de crédit :	<input type="text" value="12154216512"/>
Date d'expiration(mois/Année)	<input type="text" value="08/2008"/>

Annotations de code :

- Pointeur vers le champ Montant :  
`<h:inputText id="montant" value="#{payement.montant}">`  
`<f:convertNumber type="currency"/>`  
`</h:inputText>`
- Pointeur vers le champ Date d'expiration :  
`<h:inputText id="date" value="#{payement.date}">`  
`<f:convertDateTime pattern="MM/yyyy"/>`  
`</h:inputText>`

- Utilisation de l'attribut *type* pour déterminer le type de donnée à convertir
- Utilisation de l'attribut *pattern* pour définir le formatage à appliquer

### Possibilité d'utiliser l'attribut *converter* des balises (`inputText`, `outputText` ...)

## Utilisation de la conversion standard

**Notes :**

## La balise `convertNumber`

**Permet de déterminer comment afficher des valeurs numériques**

**Peut être paramétrée par l'intermédiaire des attributs**

Attribut	Type	Valeur
type	String	number(défaut), currency ou percent
pattern	String	Définition en fonction d'un pattern (défini dans <code>java.text.DecimalFormat</code> )
maxFractionDigits	int	Nombre maximum de chiffres derrière la virgule
minFractionDigits	int	Nombre minimum de chiffres derrière la virgule
maxIntegerDigits	int	Nombre maximum de chiffres dans la partie entière
minIntegerDigits	int	Nombre minimum de chiffres dans la partie entière
integerOnly	boolean	Vrai si on ne prend que la partie entière (défaut: faux)
groupingUsed	boolean	Vrai si l'on utilise un groupement des chiffres (défaut:vrai)
locale	<code>java.util.Locale</code>	Définition de la donnée Locale à utiliser
currencyCode	String	Code monétaire à utiliser (ISO 4217)
currencySymbols	String	Symbole monétaire à utiliser

**Table 3: Les attributs de `convertNumber`**

## La balise `convertNumber`

**Notes :**

## La balise `convertDateTime`

**Permet de déterminer comment afficher une date**

**Peut-être paramétrée par l'intermédiaire des attributs**

Attribut	Type	Valeur
type	String	date (défaut), time ou both
dateStyle	String	default, short, medium, long ou full
timeStyle	String	default, short, medium, long ou full
pattern	String	Définition en fonction d'un pattern (défini dans <code>java.text.SimpleDateFormat</code> )
locale	<code>java.util.Locale</code>	Définition de la donnée Locale à utiliser
timeZone	<code>java.util.TimeZone</code>	Fuseau horraire à utiliser

**Table 4: Les attributs de la balise `convertDateTime`**



## La balise `convertDateTime`

**Notes :**

## Gestion des erreurs de conversion

### Utilisation d'un mécanisme d'interception des erreurs

- Les tags `<message>` et `<messages>` permettent l'affichage des messages
- Envois de message (par le composant où l'erreur de conversion a lieu)
- Ré-affichage de la page après la fin du processus de validation (toutes les valeurs saisies sont ré-affichées)

### Possibilité de mettre au point un *gestionnaire de conversion* personnalisé

### Utilisation de *message* pour afficher un message d'erreur

The diagram illustrates a validation error in a web form. At the top, a text input field contains the value '10000000,00'. To its right is the HTML tag `<h:message for="montant" />`, with an arrow pointing from the tag to the input field. Below the input field, the error message is displayed: `j_id_id17:montant: '10000000,00' could not be understood as a currency value. Example: 99,99 €`. An arrow points from the input field to this message.

## Gestion des erreurs de conversion

### Notes :

Les messages d'erreurs issus de ces conversions peuvent être affichés en utilisant les tag `<message>` ou `<messages>`.

Par défaut, ils contiennent une description : « Conversion error occurred ».

Pour modifier ce message par défaut ou l'internationaliser, il faut définir une clé `javax.faces.component.UIInput.CONVERSION` dans le fichier properties de définition des chaînes de caractères.

`javax.faces.component.UIInput.CONVERSION`=La valeur saisie n'est pas correctement formatée.

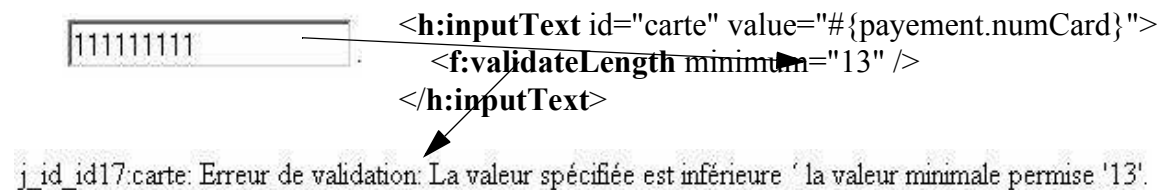
## Les valideurs standards

**Permet de s'assurer de la cohérence des données par rapport aux valeurs transmises au modèle**

### Plusieurs types de valideurs déjà existants

- `validateDoubleRange` utilisant la classe `DoubleRangeValidator` : Une valeur de type *double* avec une taille optionnelle
- `validateLongRange` utilisant la classe `LongRangeValidator` : Une valeur de type *long* avec une taille optionnelle
- `validateLength` utilisant la classe `LengthValidator` : Nombre de caractères d'une chaîne
- Validation implicite `<h:inputText required="true" .../>`

### Possibilité de mettre au point son propre valideur



## Les valideurs standards

### Notes :

L'ajout d'une validation sur un contrôle peut se faire de plusieurs manières :

ajout d'une ou plusieurs validations directement dans la vue

ajout par programmation d'une validation en utilisant la méthode `addValidator()`.

certaines implémentations de composants peuvent contenir des validations implicites comme `<h:inputText>`.

## Mise en place d'un convertisseur personnalisé

### Doit implémenter l'interface javax.faces.convert.Converter

### Permet de décrire les comportements dans les différents cas rencontrés

- `public Object getAsObject(FacesContext context, UIComponent component, String value)`
- `public String getAsString(FacesContext context, UIComponent component, Object value)`

### Déclaration de l'utilisation

- dans le fichier faces-config.xml (balise converter)

```
<converter>
  <converter-id>com.leuville.Card</converter-id>
  <converter-class>com.leuville.CardConverter</converter-class>
</converter>
```

- OU avec l'annotation `@FacesConverter`

```
import javax.faces.component.UIComponent;
import javax.faces.context.FacesContext;
import javax.faces.convert.Converter;
import javax.faces.convert.FacesConverter;
@FacesConverter("com.leuville.Card")
public class CardConverter implements Converter{
    //...
}
```

## Mise en place d'un convertisseur personnalisé

### Notes :

- **context** est l'instance FacesContext de la requête
- **component** est le composant dont la valeur doit être convertie
  - Il n'est pas de la responsabilité de ces méthodes de modifier les caractéristiques du composant
- **value** est la valeur à convertir

Si une exception de type ConverterException est montée lors de ces méthodes, le composant est marqué *invalid* et le message issu de l'exception est ajouté à la file du FacesContext courant.

## Mise en place d'un convertisseur personnalisé

- Utilisation dans la page xhtml

```
<h:inputText id="carte" value="#{payement.card}"  
  <f:converter converterId="com.leuville.Card" />  
</h:inputText>
```



# Mise en place d'un convertisseur personnalisé

## Notes

## Mise en place d'un validateur personnalisé

### Doit implémenter l'interface Validator

- `public void validate(FacesContext context, UIComponent component, Object object)`

### Déclaration de l'utilisation

- dans le fichier faces-config.xml (balise validator)

```
<validator>
  <validator-id>com.leuville.Card</validator-id>
  <validator-class>com.leuville.CardValidator</validator-clas
</validator>
```

- OU avec l'annotation `@FacesValidator`

```
import javax.faces.component.UIComponent;
import javax.faces.context.FacesContext;
import javax.faces.validator.FacesValidator;
import javax.faces.validator.Validator;
import javax.faces.validator.ValidatorException;

@FacesValidator("com.leuville.Card")
public class CardValidator implements Validator{
//...
}
```

# Mise en place d'un valideur personnalisé

## Notes

## Mise en place d'un valideur personnalisé

- Utilisation dans la page JSP

```
<h:inputText id="carte" value="#{payement.card}"  
  <f:converter converterId="com.leuville.Card" />  
  <f:validator validatorId="com.leuville.Card" />  
</h:inputText>
```

### Possibilité d'utiliser le même nom d'identifiant pour un convertisseur et un valideur

- Utilisation d'espace de nommage différent entre validator et converter

## Mise en place d'un valideur personnalisé

**Notes :**

## Mise en place d'un valideur personnalisé

### Possibilité d'utiliser une méthode d'une classe comme valideur

```
public class Beanpayement {  
    ...  
    public void methodValidator(FacesContext context, UIComponent component, Object object)  
    ...  
}  
}
```

- Même signature que la méthode *validate* de l'interface Validator

```
<h:inputText id="carte" value="#{payement.numCard}" required="true" validator="#{payement.methodValidat
```

## Mise en place d'un valideur personnalisé

**Notes :**

# *Java et Web 2.0*

## *Intégration de Dojo à JSF*

---

*Version 1.0*

- Les différentes approches avec la version de JSF < 2.0
  - Association des composants JSF avec des widgets Dojo
  - Construction d'un composant JSF personnalisé, combiné avec un widget Dojo
  - Conversion d'un composant JSF en un widget Dojo
- Dojo avec JSF version 2.0

(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).





## Les différentes approches de Dojo avec JSF < 2.0

- DojoFaces est l'archive permettant de connecter les widgets Dojo aux modèles de balises standards pour JSF 1.1 jusqu'à la 2.0.
- Associer JSF et Dojo permet d'obtenir les avantages des 2 technologies, plusieurs approches sont possibles:
  - L'association des composants JSF et des widgets Dojo du côté client.
  - Construction d'un composant JSF personnalisé, combiné avec un widget Dojo
  - Conversion d'un composant JSF en widget Dojo par injection côté client.
- Chacune de ces approches possèdent des avantages et des inconvénients, classifiés selon 3 critères:
  - La complexité: s'il est facile à implémenter.
  - L'applicabilité: si cela est possible avec tous les composants JSF, s'il y a des limites.
  - La réutilisation: s'il est facile de le réutiliser, de le réadapter sur un autre projet.
- Le tableau ci-dessous indique le niveau pour chaque aspect (1 étant le plus faible et 3 le plus fort).

**Table 5: Comparaison**

Les approches	Complexité	Applicabilité	Réutilisation
<b>Approche 1 (Association)</b>	Niveau 1	Niveau 2	Niveau 2
<b>Approche 2 (Construction)</b>	Niveau 3	Niveau 3	Niveau 3
<b>Approche 3 (Conversion)</b>	Niveau 2	Niveau 3	Niveau 2

# Les différentes approches

## Notes

## Association des composants JSF et des widgets Dojo côté client (deffered binding)

- Il s'agit d'utiliser JavaScript pour relier le composant JSF et le widget Dojo, plusieurs étapes sont nécessaires:
  - Sur la page d'un projet JSF, il faut cacher, à savoir rendre invisible, le composant JSF et ajouter la librairie Dojo.
  - Puis durant la phase de rendu, copier les données du composant JSF au widget Dojo.
  - Puis durant la phase d'envoi/mise à jour, copier les données du widget Dojo au composant JSF.
- D'après le tableau de comparaison:
  - Cette approche est facile à implémenter.
  - Sa limite est de ne pas être applicable pour tous les composants JSF (par exemple *Checkbox*).

## **Association des composants JSF et des widgets Dojo côté client (deffered binding)**

### **Notes**

## Construction d'un composant JSF personnalisé, combiné avec un widget Dojo

- JSF est un framework puissant avec une API flexible permettant de créer des composants personnalisés.
- Cette approche est la plus complexe des trois.
- Les composants JSF sont composés de différentes classes:
  - **UIComponent**: est la classe de base pour tous les composants JSF.
  - **Render**: est la classe de rendu pour les composants JSF, il récupère le rendu du code HTML, il s'agit alors ici d'utiliser le rendu de style Dojo.
  - **UIComponentTag**: classe gestionnaire de balise de JSP permettant à la classe UIComponent d'être utilisée sur JSP.
  - **Tag Library Descriptor (TLD)**: le descripteur de fichier des balises est un standard JSP de JavaEE qui associe la classe de gestionnaire de balises avec une balise dans une page JSP.
- Les étapes à suivre:
  - Utiliser une balise JSF personnalisée.
  - Soit créer un nouveau composant JSF avec UIComponent, soit réutiliser un déjà existant.
  - Réimplémenter les classes de rendu et de gestion de balises.
  - Définir le descripteur de fichiers des balises pour le composant JSF.
  - Enregistrer ce composant JSF dans faces-config.xml

# **Construction d'un composant JSF personnalisé, combiné avec un widget Dojo**

## **Notes**

## Conversion d'un composant JSF en widget DOJO par injection côté client (lazy injection)

- Cette approche est proche de la première avec l'usage de JavaScript côté client, sauf que celle-ci effectuent des injections d'informations. Il y a donc plusieurs étapes à suivre:
  - Injecter les informations de widget Dojo dans le composant JSF. Cela est possible car JSF traduit ses composants en balises HTML qui seront parser avec Dojo. Pour les différencier, un attribut spécial est ajouté "jsf2dojo='true'".
  - Parser les composants JSF (précédemment injectés) dans les widgets Dojo. Il faudra pour cela ajouter le script 'jsf2dojo.js' qui nous permettra de récupérer la liste de noeuds correspondant aux composants JSF.
  - Vérifier visuellement la page, si celle-ci contient bien le style des widgets Dojo.



# **Conversion d'un composant JSF en widget DOJO par injection côté client (lazy injection)**

## **Notes**

## Dojo avec JSF version 2.0

- DojoFaces est l'archive permettant de connecter les widgets Dojo aux modèles de balises standards pour JSF 1.1 jusqu'à la 2.0.
- Se reporter aux balises de l'archive avec **xmlns:dojo="http://j4fry.org/dojo"**
- A ajouter dans le fichier *faces-config.xml*

```
<lifecycle>
  <phase-listener>
    org.j4fry.dojo.listener.SetContentTypeListener
  </phase-listener>
</lifecycle>
```
- Sans omettre d'ajouter le framework Dojo dans la partie *resources* pour JSF 2.0.
- L'écriture s'effectue directement sur des pages XHTML avec JSF 2.0.
- Cependant dans la version 1.7 de Dojo, certaines syntaxes de code ne sont pas encore reconnues (par exemple *'on()'* ne fonctionne pas il faut passer par *'connect()'*).

# **Dojo avec JSF 2.0**

## **Notes**

## Dojo avec JSF version 2.0

### Exemple d'une page XHTML personnalisable avec Dojo et JSF définis

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:dojo="http://j4fry.org/dojo">
<h:head>
  <ui:insert name="meta"></ui:insert>
  <title><ui:insert name="title">
    No title defined!
  </ui:insert></title>
  <style type="text/css">
    <ui:insert name="styles">
      @import "${facesContext.externalContext.requestContext-
tPath}/resources/dojo/dojo/resources/dojo.css"
    </ui:insert>
  </style>
  <link rel="stylesheet" href="${facesContext.external-
Context.requestContext-
Path}/resources/dojo/dojo/dijit/themes/claro/claro.css" />
  <ui:insert name="djConfig">
    <script type="text/javascript" src="${facesCon-
text.externalContext.requestContext-
Path}/resources/dojo/dojo/dojo.js"
      data-dojo-config="isDebug: false, parseOnLoad: true,
extraLocale: ['en-us']">
    </script>
  </ui:insert>
</h:head>
<h:body class="claro">
  <div id="header">
    <ui:insert name="header">
      <ui:include src="./header.xhtml" />
    </ui:insert>
  </div>
  <div id="content">
    <h:outputLabel value="Welcome #{registerBean.name}" />
    <ui:insert name="content">No content
defined!</ui:insert>
  </div>
  <div id="footer">
    <ui:insert name="footer">
      <ui:include src="./footer.xhtml" />
    </ui:insert>
  </div>
</h:body></html>
```

## Dojo avec JSF 2.0

### Notes

L'inconvénient de cette pratique est qu'il existe très peu de documentations sur ce sujet. Sachant que certains modules Dojo ne fonctionnent pas avec les éléments de DojoFaces.

Sur le navigateur Chrome, il existe des outils pour analyser le code généré, utile pour débbugger.

# *Java et Web 2.0*

## *Annexes*

---

*Version 1.0*

- Annexe 1: Règles d'écriture d'un document XML
- Annexe 2: Validation d'un document XML avec des DTD
- Annexe 3: Validation d'un document XML avec des schémas XML

(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects, est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).



***Règles d'écriture d'un document XML***

---

- Structure d'un document XML
- Composition d'un document XML
- Document Bien formé et valide

(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects, est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).





## Structure d'un document XML

### Composé d'éléments obligatoires et optionnels

- Un prologue (optionnel) de déclaration XML

- La version XML utilisée
- Le jeu de codage utilisé
- La dépendance à une DTD

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
```

- Un prologue (optionnel) d'instruction de traitement (interprété par l'application qui utilise le document)

- Le type (MIME) du style utilisé , text/css pour les fichiers CSS et text/xsl pour les fichiers XSL
- L'URL du fichier contenant les styles à appliquer

```
<?xml-stylesheet type="text/xsl" href="myStyle.xsl" ?>
```

- La déclaration (optionnel) de l'utilisation d'une DTD

```
<!DOCTYPE Test SYSTEM "test.dtd">
```

- Un élément racine (obligatoire) qui va contenir les autres éléments (fils)
- Des commentaires (optionnel) identiques aux commentaires HTML

```
<!-- Un commentaire -->
```

# Structure d'un document XML

## Notes

## Les éléments d'un document XML

### Un document XML peut-être représenté sous forme d'arbre

- Une racine unique
- Des branches
- Des feuilles

```
<?xml version="1.0" encoding="ISO-8859-1" standal-  
one="no"?>  
  
<!DOCTYPE Test SYSTEM "test.dtd">  
<!-- Un commentaire -->  
<Test>  
  <nom>Leuville</nom>  
  <adresse>  
    <rue>3 rue de la porte de Buc</rue>  
    <code-postal>78000</code-postal>  
    <localite>Versailles</localite>  
  </adresse>  
</Test>
```

### Un élément

- Possède un nom
- Peut contenir des attributs
- Peut contenir des balises ou du texte

# Les balises d'un document XML

## Notes

## Règles d'écriture d'un document XML

### 8 règles à respecter pour qu'un document soit bien formé

- L'élément racine doit être unique
- Le nom ne peut pas commencer par un chiffre
- Si le nom ne comporte qu'un caractère, ce caractère est forcément une lettre
- Toute balise non vide (qui contient des balises ou du texte) doit posséder une balise fermante
- Une balise sans contenu doit s'écrire : `<balise />`
- Les noms d'attributs sont en minuscule
- Les valeurs des attributs sont entre guillemets

# Règles d'écriture d'un document XML

## Notes

## **Documents 'bien formé' et 'valide'**

### **Documents 'bien formé'**

- Documents respectant les règles présentées précédemment

### **Documents 'valide'**

- Est un document bien formé
- Est associé à un document décrivant les imbrications entre balises en utilisant des fichiers de définition
  - DTD
  - Schéma XML
  - Relax NG



# **Documents 'bien formé' et 'valide'**

## **Notes**

## Les sections CDATA

### Balise permettant d'utiliser des caractères réservés (<, >, etc.)

- Le contenu ne sera pas analysé

```
<![CDATA[Une section CDATA avec caractère non analysé (<, >).]]>
```

- Commence par : <![CDATA[
- Se termine par ]]>

---

Il n'est pas possible d'imbriquer des sections CDATA

---

# Les sections CDATA

## Notes

- Document Type Definition
- Les types pré définis

(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects, est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).



## Document Type definition

- Une DTD XML est analogue à une DTD SGML
- Une DTD XML est analogue à une grammaire (format EBNF : Extended Backus-Naur Form)

### Exemple

- Une anthologie comporte au moins un poème
- Un poème comporte un titre optionnel et au moins une strophe
- Un titre est un élément terminal (chaîne de caractères)
- Une strophe contient au moins une ligne
- Une ligne est un élément terminal (chaîne de caractères)

```
<!ELEMENT anthologie (poeme+)>
<!ELEMENT poeme (titre?, strophe+)>
<!ELEMENT titre (#PCDATA)>
<!ELEMENT strophe (ligne+)>
<!ELEMENT ligne (#PCDATA)>
```

### Contenu d'une DTD

- Types prédéfinis : ANY, EMPTY, PCDATA
- Règles d'utilisation : ?, +, ...
- Eléments avec attributs
- Entités

# Document Type Definition

## Notes

Les balises d'un document XML sont libres. Pour pouvoir valider si le document est correct, il faut définir un document nommé DTD qui est optionnel. Sans sa présence, le document ne peut être validé : on peut simplement vérifier que la syntaxe du document est correcte.

Une DTD est un document qui contient la grammaire définissant le document XML. Elle précise notamment les balises autorisées et comment elles s'imbriquent.

La DTD peut être incluse dans l'en-tête du document XML ou être mise dans un fichier indépendant. Dans ce cas, la directive `< !DOCTYPE>` dans le document XML permet de préciser le fichier qui contient la DTD.

Il est possible d'utiliser une DTD publique ou de définir sa propre DTD si aucune ne correspond à ces besoins.

Pour être valide, un document XML doit avoir une syntaxe correcte et correspondre à la DTD, dans le cas où l'usage de cette dernière est définie.

## Types prédéfinis DTD

### Trois possibilités

- ANY : l'élément peut contenir tout type de données
- EMPTY : l'élément n'est pas fermé
- PCDATA : l'élément contient une chaîne de caractères dont le contenu est ignoré lors de l'analyse du document

`<!ELEMENT nom (#PCDATA)>`



## **Types prédéfinis DTD**

### **Trois possibilités**

L'usage de ANY est généralement déconseillé.

## Règles d'utilisation des éléments de DTD

### Exemple

```
<!ELEMENT web-app (icon?, display-name?, description?, distributable?, context-param*, filter*, filter-mapping*, listener*, servlet*, servlet-mapping*, session-config?, mime-mapping*, welcome-file-list?, error-page*, taglib*, resource-env-ref*, resource-ref*, security-constraint*, login-config?, security-role*, env-entry*, ejb-ref*, ejb-local-ref*)>
```

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

### Signification

- ? signifie que l'élément est optionnel
- \* signifie que l'élément peut-être absent ou répété plusieurs fois
- + signifie que l'élément est présent au moins une fois et peut être répété
- (A | B) signifie que soit A soit B peut être présent
- (A , B) signifie que les deux éléments A et B doivent être présents, dans cet ordre

## Règles d'utilisation des éléments de DTD

### Exemple

Cet exemple est tiré de la spécification Servlet de J2EE : il s'agit du Document Type Définition d'un descripteur de déploiement d'une application web J2EE.

Le DTD de cet exemple est défini par Sun Microsystems.

## Les attributs des éléments de DTD

- Permet d'ajouter des propriétés à un élément

```
<! ATTLIST Elément Attribut Type >
```

### ATTLIST

- Type peut être :
  - un littéral
  - une énumération
  - atomique : identifiant unique défini avec le mot-clé ID
- Chaque attribut peut être suivi d'un mot-clé qui spécifie un niveau parmi:
  - #IMPLIED : optionnel
  - #REQUIRED : obligatoire
  - #FIXED : suivi d'une valeur par défaut si non défini

```
<! ATTLIST Elément Attribut (Valeur1 | Valeur2 ) "valeur par défaut"
```

```
<! ATTLIST disque IDdisk ID #REQUIRED type(K7|MiniDisc|Vinyl|CD)"CD" >
```

## Les attributs des éléments de DTD

### ATTLIST

Le dernier exemple signifie que l'on affecte à l'élément disque deux attributs IDdisk et type.

Le premier attribut est de type atomique, il s'agit d'un identifiant unique obligatoire.

L'élément type peut être soit K7, MiniDisc, Vinyl ou CD, sachant que ce dernier sera affecté par défaut.

## Déclarer des entités dans une DTD

- Une entité permet de définir un texte de substitution

### Déclarer des entités

```
<!ENTITY tecfaUnit "Unité de technologies de formation et apprentissage">  
<!ENTITY tecfaDesc SYSTEM "http://tecfa.unige.ch/..tecfa_description.xml"  
<!ENTITY pm "Patrick Mendelsohn"><!ENTITY acirc "&#194;">  
<!ENTITY espace "&#160;"><!ENTITY copyright "&#xA9;">  
<!ENTITY explication SYSTEM "project1a.xml">
```

### Utiliser des entités

```
<para> &pm; sort du ch&acirc;teau, s'</para>
```

```
<para>  
<citation> &explication; </citation>  
</para>
```

## Déclarer des entités dans une DTD

### Déclarer des entités

Une entité peut s'utiliser lorsqu'on doit souvent faire apparaître le même élément dans un document.

***Valider des documents XML avec des schémas XML***

---

- Présentation de XML Schema
- XML Schema et Java

(c) Leuville Objects. Tous droits de traduction, d'adaptation et de reproduction par tous procédés, réservés pour tous pays.

Toute reproduction ou représentation intégrale ou partielle, par quelque procédé que ce soit des pages publiées dans le présent ouvrage, faite sans l'autorisation de Leuville Objects, est illicite et constitue une contrefaçon (loi du 11 mars 1957 et code de la propriété intellectuelle du 1er juillet 1992, articles L 122-4, L 122-5 et L 335-2).





## XMLSchema

### Tentative d'amélioration des DTD

- Typage fort et types normalisés par le W3C
- Support des cardinalités autres que 0, 1, plusieurs
- Mécanisme de dérivation
- Espaces de nommage
- Exprimé en XML

### Mais

- Plus difficiles à construire que des DTD
- Plus difficiles à lire (pour des schémas complexes)

# XMLSchema

## Tentative d'amélioration des DTD

Pour pallier les inconvénients des Document Type Définitions, les schémas ont été créés.

Ce sont de vrais fichiers XML dont le but est de valider les documents.

Un schéma est un fichier de type .XSD ; il y a de fortes chances que les schémas supplantent les DTD dans les années futures. Toutefois, les DTD sont plus simples et plus faciles à écrire que les schémas.

## Exemple XMLSchema

### Données

```
<?xml version="1.0" ?>
<films xsi:noNamespaceSchemaLocation="cine.xsd"
  xmlns:xsi="http://www.w3.org/...">

  <film>
    <titre>le silence des agneaux</titre>
    <acteur>jodie foster</acteur>
    <acteur>anthony hopkins</acteur>
  </film>

  <film>
    <titre>conan le barbare</titre>
    <acteur>arnold schwarzenegger</acteur>
  </film>

</films>
```

### Structure en XSD

```
<?xml version="1.0" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">

  <xsd:element name="acteur" type="xsd:string"/>
  <xsd:element name="titre" type="xsd:string"/>

  <xsd:element name="film">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="titre"/>
        <xsd:element ref="acteur" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="films">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="film" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

## Exemple XMLSchema

### Notes

La balise `<films xsi:noNamespaceSchemaLocation="cine.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">` inséré dans le document XML permet d'associer le document à un XMLSchema.

## Les éléments d'un schéma XML

### Types simples

- Numériques : xsd:float, xsd:double, xsd:decimal, xsd:integer, xsd:nonPositiveInteger...
- Dates : xsd:gYear, xsd:gMonth, xsd:gDay, xsd:time, xsd:date
- Chaines de caractères : xsd:string, xsd:normalizedString, xsd:token
- xsd:ENTITY, xsd:NOTATION, xsd:IDREF

### Types complexes

```
<complexType name="unbean">
  <sequence>
    <element name="x" type="xsd:int"/>
    <element name="y" nillable="true" type="xsd:string"/>
  </sequence>
</complexType>
```

- Peuvent être imbriqués
- Ordre des champs spécifiable : xsd:sequence, xsd:choice, xsd:all

# Les éléments d'un schéma XML

## Types simples

Les types pré-définis peuvent être contrôlés de façon précise, comme le montre le transparent suivant.

## Types complexes

- `xsd:all` pour que chaque élément du groupe apparaisse au plus une fois, l'ordre n'étant pas important,
- `xsd:choice` pour qu'un seul élément n'apparaisse ou pour que de N à M éléments du groupe apparaisse dans un ordre quelconque,
- `xsd:sequence` pour que chaque élément du groupe apparaisse exactement une fois dans l'ordre indiqué.

## Contrôles sur les types XMLSchema

### Attributs de contrôle des valeurs possibles

- `xsd:minExclusive` pour indiquer la valeur minimale (non comprise)
- `xsd:maxInclusive` pour indiquer la valeur maximale (comprise)
- `xsd:maxExclusive` pour indiquer la valeur maximale (non comprise)
- `xsd:enumeration` pour donner une liste de valeurs valides
- `xsd:whiteSpace` pour indiquer comment gérer espaces dans l'élément
- `xsd:pattern` pour utiliser une expression régulière de comparaison
- chaînes de caractères : contraintes de longueur avec `length`, `minLength`, et `maxLength`.



## Contrôles sur les types XMLSchema

### Attributs de contrôle des valeurs possibles

Pour les chaînes de caractères, on peut préciser des contraintes de longueur avec `length`, `minLength`, et `maxLength`.

Pour les nombres, `totalDigits` et `fractionDigits` jouent un rôle similaire.

Avec `xsd:pattern` on peut préciser des expressions régulières étendues (comprenant des spécifications pour l'unicode) qui permettent de restreindre principalement le type `xsd:string`.

## Types XMLSchema dérivés

- Type dérivé construit à partir d'un type de base sur lequel on pose des restrictions

### Exemple

- Type de base xsd:gYear
- Restriction de validité : valide seulement si  $\geq 1900$

```
<xsd:simpleType name="anNaissance">  
  <xsd:restriction base="xsd:gYear">  
    <xsd:minInclusive value="1900" />  
  </xsd:restriction>  
</xsd:simpleType>
```

## Types XMLSchema dérivés

### Notes

Il est possible de définir un type dérivé qui utilise un type de base avec des restrictions.

L'exemple définit un type année de naissance en imposant une restriction de validité.

## Les espaces de nommage XMLSchema

### Utilité

- Définir une zone dans laquelle il y a unicité des noms
- Permet d'éviter les collisions de noms d'éléments avec d'autres éléments importés

### Utiliser un espace de nommage

```
<wsdl:definitions targetNamespace="http://localhost:8080/axis/WSTest1.jws"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:apachesoap="http://xml.apache.org/xml-soap"
  xmlns:impl="http://localhost:8080/axis/WSTest1.jws"
  xmlns:intf="http://localhost:8080/axis/WSTest1.jws"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <wsdl:types>
    <schema targetNamespace="TP9" xmlns="http://www.w3.org/2001/XMLSchema"
      <import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
      <complexType name="unbean">
        <sequence>
          <element name="x" type="xsd:int"/>
          <element name="y" nillable="true" type="xsd:string"/>
        </sequence>
      </complexType>
    </schema>
  </wsdl:types>
```

# Les espaces de nommage XMLSchema

## Notes