

# Technologies XML

## Chapitre 2 : Langage XQuery

---

Dr Konan Marcellin BROU

[marcellin.brou@inphb.ci](mailto:marcellin.brou@inphb.ci)

[konanmarcellin@yahoo.fr](mailto:konanmarcellin@yahoo.fr)

2018-2019

# Sommaire

---

- ❑ Introduction
- ❑ Interpréteur pour XQuery
- ❑ Concept de base XQuery
- ❑ Les expressions
- ❑ Application : Recherche de composants
- ❑ Bibliographie

# I. Introduction

---

## ▣ 1.1. XML : eXtensible Markup Language

- Généralisation de HTML (HyperText Markup Language)
  - ▣ Jeu de balises prédéfinies ayant une signification prédéfinie.
  - ▣ Pas d'extension possible des balises.
- XML : l'auteur peut "inventer" ses propres balises.

## ▣ 1.2. But de XML

- Faciliter le traitement automatisé de documents et de données.
- Structurer les informations :
  - ▣ Pour qu'elles puissent être lues par des personnes sur le web ;
  - ▣ Pour qu'elles puissent être traitées par des applications de manière automatisée.

# I. Introduction

---

## □ 1.3. XQuery

- XML Query : langage de requête XML
- Une interrogation est représentée comme une expression.
- Expression
  - Instruction construite à partir de littéraux, d'opérateurs, d'appels de fonctions, d'itérateurs, ...
- XQuery est à XML ce que SQL pour les BD relationnelles.
  - SQL manipule des tables
    - Ensemble de n-uplets de valeurs atomiques

- XQuery manipule des séquences d'items
  - valeurs atomiques et nœuds d'arbres XML.

## ■ XQuery

- Basé sur Quilt, langage proposé au W3C
- Sorte d'extension de XPath
- Lié à XML Schema
- Janvier 2007 : XQuery 1.0
- Mars 2017 : XQuery 3.1, extension à JSON et XPath 3.1
  - JavaScript Object Notation.
  - Format léger de JavaScript d'échange de données comme XML.

# I. Introduction

---

## ■ Exemple avec XML

```
<menu id="file" value="File">
  <popup>
    <menuitem value="New" onclick="CreateNewDoc()" />
    <menuitem value="Open" onclick="OpenDoc()" />
    <menuitem value="Close" onclick="CloseDoc()" />
  </popup>
</menu>
```

## ■ Exemple avec JSON

```
{
  "menu": {
    "id": "file",
    "value": "File",
    "popup": {
      "menuitem": [
        {"value": "New", "onclick": "CreateNewDoc()"},
        {"value": "Open", "onclick": "OpenDoc()"},
        {"value": "Close", "onclick": "CloseDoc()"}
      ]
    }
  }
}
```

# I. Introduction

## □ 1.4. Exemples de requêtes

### ■ Exemple 0 : Rq0.xq

- Afficher tout le document.

Requête
document("bib.xml") Ou doc("bib.xml")

#### Résultat Rq0.xq

```
document {  
  <bib>  
    <book id="1" year="1994">  
      <title>TCP/IP Illustrated</title>  
      <author><last>Stevens</last>  
      <first>W.</first></author>  
      <publisher>Addison-Wesley</publisher>  
      <price>65.95</price>  
    </book>  
    ...  
    <book id="6" year="2001">  
      <title>Spatial Databases</title>  
      <author><last>Rigaux</last><first>P.</first></author>  
      <author><last>Scholl</last><first>M.</first></author>  
      <author><last>Voisard</last><first>A.</first></author>  
      <publisher>Morgan Kaufmann Publishers</publisher>  
      <price>35.00</price>  
    </book>  
  </bib>  
}
```

# I. Introduction

## ■ Exemple 1 : Rq1.xq

- ▣ Trouver tous les noms des auteurs des livres dans le document bib.xml.
- ▣ Le résultat n'est pas un document XML bien formé.

Requête
<code>document("bib.xml")//book/author/last</code>

## ■ Exemple 2 : Rq2.xq

- ▣ trouver tous les titres des livres édités après 1999.

Requête
<code>document("bib.xml")//book[@year &gt; "1999"]/title</code>

### Résultat Rq1.xq

```
<last>Stevens</last> ,  
<last>Stevens</last> ,  
<last>Abiteboul</last> ,  
<last>Buneman</last> ,  
<last>Suciu</last> ,  
<last>Amann</last> ,  
<last>Rigaux</last> ,  
<last>Rigaux</last> ,  
<last>Scholl</last> ,  
<last>Voisard</last>
```

### Résultat Rq2.xq

```
<title>Data on the Web</title>  
<title>Spatial Databases</title>
```

# I. Introduction

---

## □ 1.5. Expression de chemin

- On peut accéder à n'importe quel nœud sans passer par la racine de l'arbre.
  - Exemple : liste des auteurs
- Chemin absolu
  - Les éléments sont séparés par /

Requête
<code>document("bib.xml")//author</code>

Requête
<code>document("bib.xml")//author/last</code>
<code>Idem</code>
<code>document("bib.xml")//book/author/last</code>
<code>document("bib.xml")//bib/book/author/last</code>



## II. Interpréteur pour XQuery

---

### ▣ 2.1. Galax

- Implémentation de XQuery
  - ▣ <http://www.galaxquery.org/>
  - ▣ Version en cours : Galax 1.1
    - Trop compliqué
  - ▣ Version utilisée : Galax 0.5.0
- Développé par les laboratoires Bell Labs et AT&T.
- Requêtes stockées dans des fichiers
  - ▣ passés comme argument à l'interpréteur Galax.

## II. Interpréteur pour XQuery

---

### ■ Installation :

1. Décompresser Galax-0.5.0-MinGW.zip
2. Copier le dossier Galax-0.5.0-MinGW\Galax sur le disque C
3. Installer pcre-6.4-1.exe
4. Copier pcre3.dll du dossier C:\Program Files (x86)\GnuWin32\bin dans C:\Galax\bin

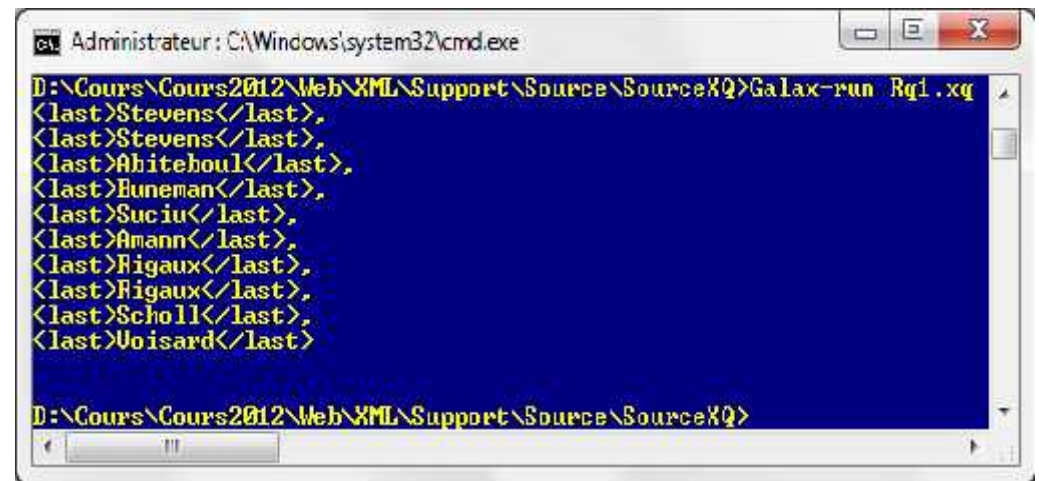
- pcre : Perl-compatible regular expressions
- La bibliothèque PCRE est une API contenant un ensemble de fonctions qui implémentent des expressions régulières utilisant la même syntaxe et la même sémantique que Perl.
- Expression régulière : chaîne de caractères, qui décrit, selon une syntaxe précise, un ensemble de chaînes de caractères possibles.
- Exemple chaîne de 4 caractères numériques : "[0-9]{4}"

4. renommer pcre3.dll en pcre.dll

## II. Interpréteur pour XQuery

### ■ Utilisation

- Ouvrir une invite de commande et saisir :
  - `path = %path%;C:\Galax\bin`
- Se placer dans le répertoire où sont copiés les fichiers XQuery.
- Exécuter la requête Rq1.xq :
  - `Galax-run Rq1.xq`
  - Le résultat est renvoyé directement dans la fenêtre de commande.




```
Administrateur: C:\Windows\system32\cmd.exe
D:\Cours\Cours2012\Web\XML\Support\Source\SourceXQ>Galax-run Rq1.xq
<last>Stevens</last>,
<last>Stevens</last>,
<last>Abiteboul</last>,
<last>Buneman</last>,
<last>Suciu</last>,
<last>Amann</last>,
<last>Rigaux</last>,
<last>Rigaux</last>,
<last>Scholl</last>,
<last>Voisard</last>

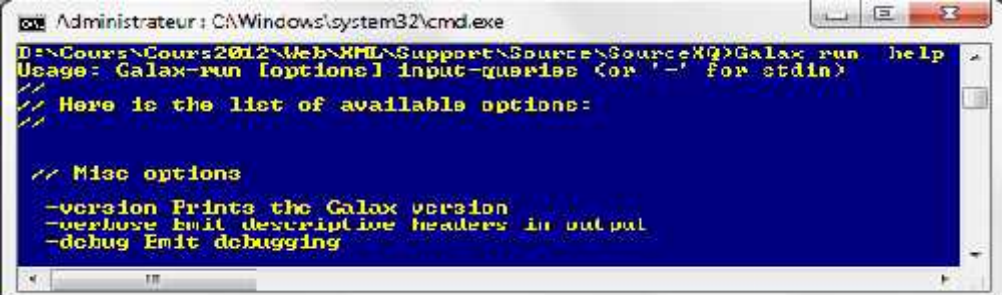
D:\Cours\Cours2012\Web\XML\Support\Source\SourceXQ>
```

## II. Interpréteur pour XQuery

- Envoyer le résultat dans un fichier xml.
  - `Galax-run Rq1.xq -output-xml Rq1.xml`
  - Ou
  - `Galax-run Rq1.xq > Rq1.xml`
  - Visualiser le vicher Rq1.xml
  - Ce n'est pas un document XML bien formé.
- Liste des options de la commande :
  - `Galax-run -help`



```
Administrateur : C:\Windows\system32\cmd.exe
D:\Cours\Cours2012\Web\XML\Support\Source\SourceXQ>Galax-run Rq1.xq -output-xml Rq1.xml
D:\Cours\Cours2012\Web\XML\Support\Source\SourceXQ>
```



```
Administrateur : C:\Windows\system32\cmd.exe
D:\Cours\Cours2012\Web\XML\Support\Source\SourceXQ>Galax-run -help
Usage: Galax-run [options] input-queries <or '-' for stdin>
// Here is the list of available options:

// Misc options
-version Prints the Galax version
-verbose Emit descriptive headers in output
-debug Emit debugging
```

## II. Interpréteur pour XQuery

## 2.2. Galax Server

- Interface Web permettant d'utiliser Galax
- Développée par les étudiant de l'INP-HB
- Utilisation
  - Lancer Wamp Server
  - Créer un alias : galaxServer
    - Chemin : le dossier contenant GalaxServer
  - Lancer un navigateur
  - Saisir l'URL : <http://localhost/galaxServer/>



# III. Concepts de base de XQuery

## ■ 3.1. Structure d'une requête XQuery

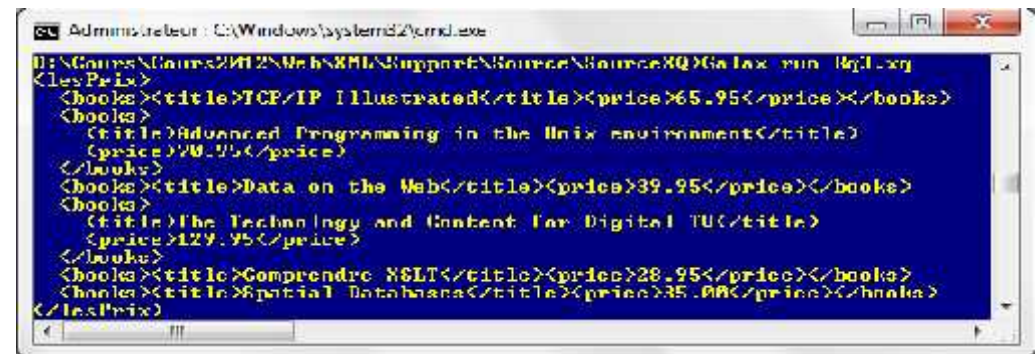
- Requête composée d'expressions
  - Instructions XQuery
- Requête formée :
  - d'un **prologue** : composé d'une suite de déclarations ;
  - d'un **corps** : composé d'une expression dont la valeur est le résultat de la requête.
- Valeur d'une requête
  - Valeur de l'expression qui constitue son corps, évaluée dans le contexte défini par le prologue.

## ■ Exemple 3 : Titre et prix des livres (Rq3.xq)

```
declare function local:prix($nom)
{ document("bib.xml")//book[title =
$nom]/price } ;
<lesPrix>
{ let $d := document("bib.xml")//book
  for $m in $d/title
  let $p := local:prix($m)
  return (<books>{ $m, $p } </books> )
}
</lesPrix>
```

} Prologue

} Corps



```
Administrateur : C:\Windows\system32\cmd.exe
D:\Games\Games2012\Web\XML\Support\Source\SourceXQ\Galax run Rq3.xq
<lesPrix>
  <books><title>TCP/IP Illustrated</title><price>65.95</price></books>
  <books>
    <title>Advanced Programming in the Unix environment</title>
    <price>20.95</price>
  </books>
  <books><title>Data on the Web</title><price>39.95</price></books>
  <books>
    <title>The Technology and Content for Digital TV</title>
    <price>129.95</price>
  </books>
  <books><title>Comprendre XSLT</title><price>28.95</price></books>
  <books><title>Spatial Databases</title><price>35.00</price></books>
</lesPrix>
```

# III. Concepts de base de XQuery

---

## ■ Prologue contient :

- Déclarations de Name space (espace de nom) ;
- Importations de schémas ;
- Importation de Modules
  - ensemble de déclaration de variables et de fonctions;
- Définitions de fonctions ;
- Déclarations de variable globales et externes ;
- Contrôle pour la gestion des espaces.

## ■ Espace de nom XML

- XML namespace
- **Problème** : Plusieurs formats basés sur XML peuvent utiliser les mêmes noms d'éléments pour repérer des concepts différents.
- Identifie une partie du Web qui est utilisée pour identifier un ensemble de noms (d'éléments, d'attributs).
- Objectifs des espaces de nom :
  - distinguer les éléments et les attributs de différentes applications XML qui ont le même nom ;
  - grouper les éléments et les attributs d'une même application XML pour que les logiciels puissent les reconnaître.
- Exemple : fusion de deux document contenant l'élément titre

# III. Concepts de base de XQuery

■ Exemple :

```
<?xml version="1.0"?>
<auteur>
  <nom>Poulard</nom>
  <prénom>Philippe</prénom>
  <titre>Baron</titre>
</auteur>
```

```
<?xml version="1.0"?>
<cours>
  <titre>Fondamentaux XML</titre>
  <contenu>
    .../...
  </contenu>
</cours>
```

Confusion sur le sens  
de l'élément <titre>

```
<?xml version="1.0"?>
<cours>
  <titre>Fondamentaux XML</titre>
  <auteur>
    <nom>Poulard</nom>
    <prénom>Philippe</prénom>
    <titre>Baron</titre>
  </auteur>
  <contenu>
    .../...
  </contenu>
</cours>
```



# III. Concepts de base de XQuery

---

- Déclaration des espaces de nommage :
  - Ils se déclarent dans un élément, avec le préfixe spécial xmlns
  - Exemple 1 :
  - Porté de co : balise <co:cours>
  - Porté de pe : balise <pe:auteur>

```
<?xml version="1.0"?>
<co:cours xmlns:co="http://www.foo.com/cours" >
  <co:titre>Fondamentaux XML</co:titre>
  <pe:auteur xmlns:pe="http://www.bar.com/individus" >
    <pe:nom>Poulard</pe:nom>
    <pe:prénom>Philippe</pe:prénom>
    <pe:titre>Baron</pe:titre>
  </pe:auteur>
  <co:contenu>
    .../...
  </co:contenu>
</co:cours>
```

# III. Concepts de base de XQuery

## ■ Exemple 2 :

- Porté de co : balise  
<co:cours>
- Porté de pe : balise  
<co:cours>

## ■ Corps contient :

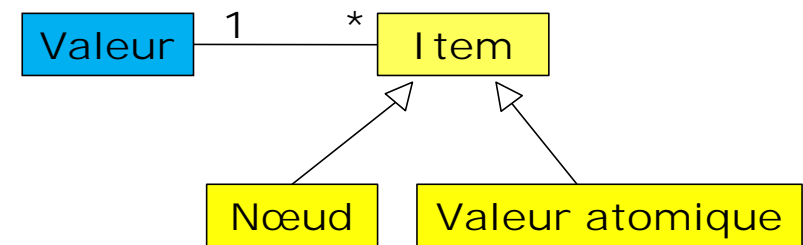
- Une expression qui définit le résultat de la requête.
- Une expression est construite à partir de littéraux, d'opérateurs, d'appels de fonctions, d'itérateurs, ...

```
<?xml version="1.0"?>
<co:cours xmlns:co="http://www.foo.com/cours"
  pe:auteur xmlns:pe="http://www.bar.com/individus" >
  <co:titre>Fondamentaux XML</co:titre>
  <pe:auteur>
    <pe:nom>Poulard</pe:nom>
    <pe:prénom>Philippe</pe:prénom>
    <pe:titre>Baron</pe:titre>
  </pe:auteur>
  <co:contenu>
    .../...
  </co:contenu>
</co:cours>
```

# III. Concepts de base de XQuery

## ■ 3.2. Modèle de données

- Expression construite à partir :
  - ▣ de littéraux, d'opérateurs, d'appels de fonctions, d'itérateurs, ...
- Toute expression a un type et une valeur.
- valeur d'une expression :
  - ▣ Séquence de 0 ou plusieurs items.
- Deux sortes d'items :
  - ▣ une valeur atomique ;
  - ▣ des nœuds d'arbres de documents XML.



# III. Concepts de base de XQuery

---

## □ 3.3. Notion d'Item

- Valeur atomique ou un nœud de l'arbre d'un document XML.

- Valeur atomique :

- Instance de l'un des types atomiques de XML Schema : nombres, chaînes, dates...

- Nœud défini comme dans XPath 2, de type :

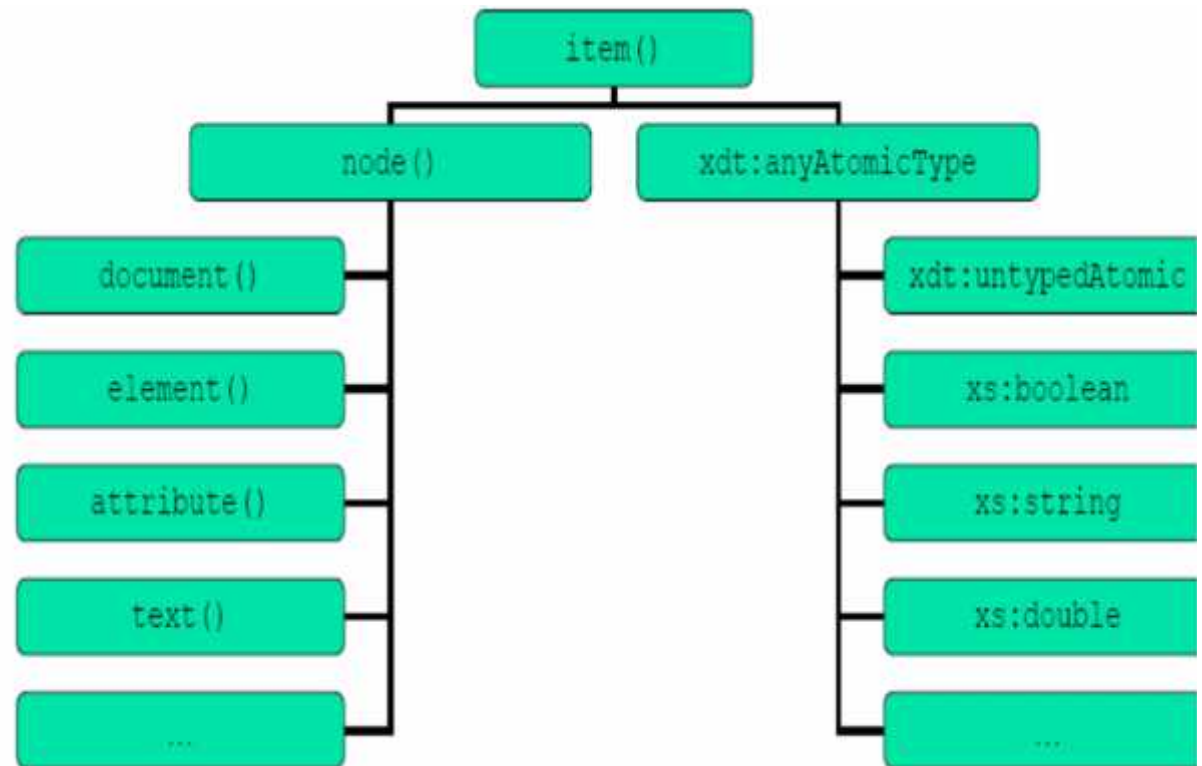
- Document ;
    - Élément ;
    - Attribut ;
    - Texte ;
    - Instruction ;
    - espace de noms ;
    - commentaire.

- Possède :

- un nom et une valeur textuelle (string()).

# III. Concepts de base de XQuery

## ■ Hiérarchie d'un item



# III. Concepts de base de XQuery

---

## ■ 3.4. Notion de Séquence

- Liste ordonnée d'items séparés par des virgules :
  - (1, 2, 3)
- Séquence vide : contient 0 items.
  - ()
- Séquence singleton : contient un et un seul item.
  - (26)
  - Un item est identique à une séquence singleton.

- Pas d'imbrication :
  - (1, (2, 3), ()) ÷ (1, 2, 3)
- Peut contenir des valeurs hétérogènes :
  - (1, "toto", <toto/>)
- Tri des séquences :
  - (1, 2) ∩ (2, 1).

# III. Concepts de base de XQuery

## ■ 3.5. Notion de Type

- Système de type compatible avec celui de XML Schema.
- Ensemble des types disponibles d'une requête :
  - types prédéfinis de XQuery ;
  - types définis dans les schémas importés dans le prologue de cette requête.
- Exprimer un type en XQuery
  - On utilise une notation appelée type de séquence
  - Une valeur en XQuery est une séquence.

## ■ 3.6. Type de séquence

Type	Explication
xs:date	Item de type prédéfini xs:date
attribute()?	Séquence de 0 ou 1 attribut
element()	Élément quelconque
element(title, xs:string)	Nœud élément de nom title et de type xs:string
element(description, *)	Nœud élément de nom description et de type quelconque
element(*, xs:string)	Nœud élément de nom quelconque et de type xs:string
node()*	Séquence de longueur quelconque nœud de type quelconque
item() +	Séquence d'au moins un item

# IV. Les expressions

---

## □ 4.1. Présentation

- Une expression est une instruction XQuery
- Différentes sortes d'expression :
  - Littéral ;
  - Variable ;
  - Appel de fonction prédéfinie ou définie ;
  - Expression de chemin ;
  - Opération construite à l'aide des opérateurs prédéfinis :
    - arithmétiques, comparateurs, ...
  - Construction de nœud ;
  - Expression itérative : FLWOR ;
  - Quantification ;
  - Expression conditionnelle ;
  - Expression impliquant des tests de type ;
  - Expression de validation ;
  - ...



# IV. Les expressions

---

## □ 4.2. Les littéraux

### ■ Booléens :

- Pas de littéraux booléens
- Obtenus par les appels de fonctions prédéfinies :
  - `true()` et `false()`.

### ■ Chaînes de caractères :

- Entre simples ou doubles quotes :
  - `"Addison-Wesley"` ou `'Addison-Wesley'`

### ■ Nombres : entiers ou réels

- `2009`
- `3.14`
- `1.602e-19`

## □ 4.3. Les commentaires

### ■ XML

- `<!-- Commentaire XML -->`

### ■ XQuery

- `(: Commentaire XQuery :)`

# IV. Les expressions

---

## □ 4.4 Les Variables

- Nom d'un variable
  - Précédé du caractère \$ :
  - **\$nom**
- Valeur d'une variable
  - Celle qui a été le plus récemment liée à ce nom.

## □ 4.5. Appel de fonction

- **f(exp1, ..., expn)**
  - f : nom de la fonction
  - exp1...expn : paramètres effectifs
- Appel d'une fonction locale
  - Nom de la fonction précédée de **local:**
  - Exemple : **local:prix(\$m)**
- Appel d'une fonction prédéfinie
  - Son nom appartient à l'espace de noms <http://www.w3.org/2004/10/xpath-functions> de préfixe **fn**.
  - Exemple :  
**fn:count(document("bib.xml")//book)**
    - fonction prédéfinie qui compte le nombre d'occurrence d'un élément.

# IV. Les expressions

## □ 4.7. Opérations arithmétiques

### ■ Syntaxe

- Opérateur `/` est réservé aux expressions de chemin.

### ■ Exemples

- A tester dans fichier `.xq`

Opération	Explication
<code>exp1 + exp2</code>	Addition
<code>exp1 - exp2</code>	Soustraction
<code>exp1 * exp2</code>	Multiplication
<code>exp1 div exp2</code>	Quotient d'une division de réels
<code>exp1 idiv exp2</code>	Quotient d'une division entière
<code>exp1 mod exp2</code>	Reste d'une division entière
<code>+exp</code>	Changement de signe
<code>-exp</code>	Changement de signe

Requête	Résultat
<code>1 + 2</code>	<code>3</code>
<code>&lt;add&gt;{ 1 + 2 } &lt;/add&gt;</code>	<code>&lt;add&gt;3&lt;/add&gt;</code>
<code>&lt;div&gt;{ 6 div 2 } &lt;/div&gt;</code>	<code>&lt;div&gt;3&lt;/div&gt;</code>

# IV. Les expressions

## 4.8. Comparaison de valeurs

- Syntaxe
- Valeur de `exp1 op exp2`
  - Obtenue en comparant les valeurs atomisées de `exp1` et `exp2`.
- Exemples
- `nom = "Toto"` a la valeur :
  - `true` si le nœud contexte a un nœud fils `<nom>Toto</nom>`
  - `false` sinon.

Opération	Explication
<code>exp1 eq exp2</code>	Egalité
<code>exp1 ne exp2</code>	différence
<code>exp1 lt exp2</code>	Infériorité stricte
<code>exp1 le exp2</code>	Infériorité large
<code>exp1 gt exp2</code>	Supériorité stricte
<code>exp1 ge exp2</code>	Supériorité large

Requête	Résultat
<code>5 gt 7</code>	<code>false()</code>
<code>&lt;sup&gt;{ 5 gt 7 } &lt;/sup&gt;</code>	<code>&lt;sup&gt;false&lt;/sup&gt;</code>
<code>&lt;a&gt;8&lt;/a&gt; lt &lt;b&gt;9&lt;/b&gt;</code>	<code>true()</code>

# IV. Les expressions

## 4.9. Comparateurs généraux

- Ont une sémantique existentielle :  $\text{exp1 op exp2}$  a la valeur :
  - `true()` s'il existe un item  $i1$  dans  $\text{exp1}$  et un item  $i2$  dans  $\text{exp2}$  tel que  $i1 \text{ op } i2$  est vrai ;
  - `false()` sinon.
- Syntaxe
- Exemples :
  - Si l'item courant est un nœud élément dont la valeur est :  
`<auteur>Toto</auteur><auteur>Durand</auteur>`
  - alors l'expression : `auteur = "Toto"` a la valeur `true`

Opération	Explication
$\text{exp1} = \text{exp2}$	Egalité
$\text{exp1} \neq \text{exp2}$	différence
$\text{Exp1} < \text{exp2}$	Infériorité stricte
$\text{exp1} \leq \text{exp2}$	Infériorité large
$\text{exp1} > \text{exp2}$	Supériorité stricte
$\text{Exp1} \geq \text{exp2}$	Supériorité large

Requête	Résultat
$(1, 2) = (7, 2, 9, 12)$	<code>true()</code>
$5 > (2, 9, 3)$	<code>true()</code>
$15 < (2, 9, 3)$	<code>false()</code>

## IV. Les expressions

### ■ Exemples :

- Si l'item courant est un nœud élément dont la valeur est :  
`<auteur>Toto</auteur><auteur>Durand</auteur>`
- alors l'expression : `auteur = "Toto"` a la valeur true

Requête	Résultat
<code>(1, 2) = (7, 2, 9, 12)</code>	<code>true()</code>
<code>5 &gt; (2, 9, 3)</code>	<code>true()</code>
<code>5 &lt; (2, 1, 3)</code>	<code>false()</code>

# IV. Les expressions

## 4.10 Compérateurs de nœuds

### Syntaxe

Opération	Explication
exp1 <b>is</b> exp2	Egalié
exp1 <b>&lt;&lt;</b> exp2	Infériorité stricte
exp1 <b>&gt;&gt;</b> exp2	Supériorité stricte

### Valeurs des opérandes

- soit une séquence vide, soit un nœud.

### Un des opérandes a pour valeur la séquence vide

- la valeur de la comparaison est false.

### Valeur de exp1 = nœud n1,

### valeur de exp2 = nœud n2 :

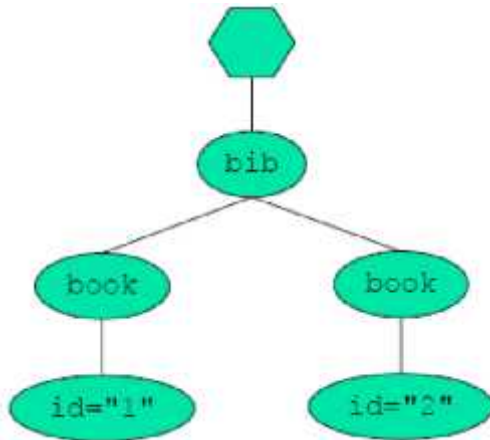
- exp1 is exp2** a la valeur
  - true, si n1 et n2 sont les mêmes (même identifiant), sinon elle a la valeur false.
- exp1 << exp2** a la valeur
  - true, si n1 précède n2 dans l'ordre du document, sinon elle a la valeur false.
- exp1 >> exp2** a la valeur
  - true, si n2 précède n1 dans l'ordre du document, sinon elle a la valeur false.

# IV. Les expressions

## ■ Exemple

### ■ L'expression

- `document("bib.xml")//book[@id="1"] << document("bib.xml")//book[@id="2"]`
- a la valeur true



## ■ 4.11. Opérations sur les séquences

- Séquence vide : `()`
- Séquence non vide : `exp1, ..., expn`
- Séquences d'entiers : `exp1 to exp2`
- Exemples :

Requête	Résultat
<code>1 + 5, 4 - 2, 7 * 3</code>	<code>6, 2, 21</code>
<code>((9, 1), 5, (8, 2))</code>	<code>9, 1, 5, 8, 2</code>
<code>2 to 5</code>	<code>2, 3, 4, 5</code>
<code>5 to 2</code>	<code>()</code>



# IV. Les expressions

- ❑ 4.12. Opérations sur les séquences de nœuds
  - Syntaxe
  - Dans la séquence résultante :
    - ❑ les doubles (nœuds de même identifiant) sont éliminés,
  - Exemple :  
`document("bib.xml")//(author | editor)`
    - ❑ a pour valeur la séquence des nœuds de type author et editor.

Rq4.xq

Opération	Explication
exp1 <b>union</b> exp2	Union
exp1   exp2	Union
exp1 <b>intersect</b> exp2	Intersection
exp1 <b>except</b> exp2	Différence

```
Administrateur: C:\Windows\system32\cmd.exe
D:\Cours\Cours2012\Web\XML\Support\Source\SourceXQ\Galax-run Rq4.xq
<author><last>Stevens</last><first>W.</first></author>,
<author><last>Stevens</last><first>W.</first></author>,
<author><last>Abiteboul</last><first>Serge</first></author>,
<author><last>Buneman</last><first>Peter</first></author>,
<author><last>Suciu</last><first>Dan</first></author>,
<editor>
  <last>Gerbarg</last>
  <first>Harry</first>
  <affiliation>CITI</affiliation>
</editor>,
<author><last>Nmann</last><first>E.</first></author>,
<author><last>Rigaux</last><first>P.</first></author>,
<author><last>Rigaux</last><first>P.</first></author>,
<author><last>Scholl</last><first>M.</first></author>,
<author><last>Voisard</last><first>O.</first></author>
```

## IV. Les expressions

- Exemple : Rq5.xq
- `document("bib.xml")//(author, price)`
  - a pour valeur la séquence des nœuds de type author suivi de prix.
  - On a changé l'ordre des nœuds.
- Remarque
  - `document("bib.xml")//(author, price)`
  - Ou `doc("bib.xml")//(author, price)`



```
Administrator: C:\Windows\system32\cmd.exe
D:\Course\Course2012\Web\XML\Support\Source\SourceXQ\Galax-run Rq5.xq
<author><last>Stevens</last><first>W.</first></author>,
<price>65.95</price>,
<author><last>Stevens</last><first>W.</first></author>,
<price>70.95</price>,
<author><last>Abiteboul</last><first>Serge</first></author>,
<author><last>Buneman</last><first>Peter</first></author>,
<author><last>Suciu</last><first>Dan</first></author>,
<price>39.95</price>,
<price>129.95</price>,
<author><last>Amann</last><first>B.</first></author>,
<author><last>Rigaux</last><first>P.</first></author>,
<price>28.95</price>,
<author><last>Rigaux</last><first>P.</first></author>,
<author><last>Scholl</last><first>M.</first></author>,
<author><last>Voisard</last><first>A.</first></author>,
<price>35.00</price>
```

# IV. Les expressions

## ■ 4.13. Constructions de nœuds

- Un constructeur produit un nouveau nœud :
  - document, élément, texte, attribut, espace de noms, commentaire, instruction de traitement.
- Un constructeur d'élément produit un nouveau nœud élément.
  - Ses nœuds fils sont eux-mêmes des nouveaux nœuds.
  - Il n'y a donc pas partage de nœuds entre deux éléments.
- On distingue :
  - les constructeurs directs ;
  - les constructeurs calculés.

## ■ Construction directe d'élément

- `<n a1="v1" ... ak="vk">c</n>`
- Valeurs d'attributs v1, ..., vk peuvent contenir :
  - des caractères, des entités ou bien des expressions XQuery placées entre accolades, appelées expressions incluses.
- L'évaluation de la valeur d'un attribut doit produire une chaîne de caractères.
- Le contenu c peut contenir des caractères, des constructions directes d'éléments ou des expressions incluses.
- L'évaluation du contenu doit produire une séquence de nœuds telle que :
  - elle ne contient pas de nœud document ;
  - les nœuds produits par une expression incluse ont été copiés pour produire de nouveaux nœuds ;
  - les nœuds attributs s'ils existent précèdent les autres nœuds ;
  - les nœuds textes adjacents sont fusionnés en un seul nœud texte.

## IV. Les expressions

### ■ Exemples :

Requête
<code>&lt;phrase&gt;1 + 1 = { 1 + 1 } &lt;/phrase&gt;</code>
<code>&lt;titre&gt;{ "Le ", " langage ", " XML" } &lt;/titre&gt;</code>
<code>&lt;personne num="n1" &gt;toto&lt;/personne&gt;</code>
<pre>let \$x := &lt;p&gt;&lt;n&gt;Toto&lt;/n&gt;           &lt;v&gt;Yakro&lt;/v&gt;           &lt;/p&gt; return &lt;personne nom="{ \$x/n} "        ville="{ \$x/v} " /&gt;</pre>

Résultat
<code>&lt;phrase&gt;1 + 1 = 2&lt;/phrase&gt;</code>
<code>&lt;titre&gt;Le langage XML&lt;/titre&gt;</code>
<code>&lt;personne nom="Toto" ville="Yakro" /&gt;</code>



The screenshot shows a Windows command prompt window titled "Administrateur : C:\Windows\system32\cmd.exe". The command entered is `D:\Cours\Cours2012\Internet\XML\Support\Source\SourceXQ>galax-run RqConstN.xq`. The output displayed is `<personne nom="Toto" ville="Yakro"/>`.

# IV. Les expressions

## ■ Constructions calculées

- Les constructeurs calculés permettent de construire des nœuds dont les composantes (nom, fils, valeur, ...) sont les valeurs d'expressions XQuery.

- `element nom {exp}`
- `element {exp1} {exp2}`
- `attribute nom {exp}`
- `attribute {exp1} {exp2}`
- `document {exp}`
- `text {exp}`

- Exemple :

Requête
<pre>element livre { attribute num { 2 } ,   element titre { "Le langage XML" } }</pre>
Résultat
<pre>&lt;livre num="2"&gt;   &lt;titre&gt;Le langage XML&lt;/titre&gt; &lt;/livre&gt;</pre>

- Exercice : créer les nœuds suivants :

```
<eleve mat="2">
  <classe codecl="3">INFO3</classe>
</eleve>
```

```
<eleve mat="2">
  <classe codecl="3">
    <nomCl>INFO3</nomCl>
  </classe>
</eleve>
```

## IV. Les expressions

### ■ Correction :

Requête	Résultat
<pre>element eleve { attribute mat { 2} ,   element classe     { attribute codecl { 3} ,       "Info3"     } }</pre>	<pre>&lt;eleve mat="2" &gt;   &lt;classe codecl="3" &gt;     INFO3   &lt;/classe&gt; &lt;/eleve&gt;</pre>

Requête	Résultat
<pre>element eleve { attribute mat { 2} ,   element classe     { attribute codecl { 3} ,       element nomCI { "Info3" }     } }</pre>	<pre>&lt;eleve mat="2" &gt;   &lt;classe codecl="3" &gt;     &lt;nomCI&gt;INFO3&lt;/nomCI&gt;   &lt;/classe&gt; &lt;/eleve&gt;</pre>

# IV. Les expressions

---

## □ 4.14. Expression itérative : FLWOR

- Prononcer "flower"
- Expression FLWOR (For, Let, Where, Order by, Return)
  - Similaire SELECT-FROM-WHERE de SQL.
- Expression FLWOR peut contenir :
  - une clause **FOR** : attache une ou plusieurs variables à une séquence de valeurs renvoyées par une autre expression (généralement une expression de chemin), et boucle dans les valeurs ;
  - une clause **LET** : idem **FOR**, mais sans itération ;

- une clause **WHERE** : contient un ou plusieurs prédicats qui filtrent le jeu de nœuds générés par les clauses FOR/LET.
- Une clause **ORDER BY** : pour le tri ;
- une clause **RETURN** :
  - Génère le résultat de l'expression FLWOR ;
  - Contient un ou plusieurs éléments constructeurs et/ou des références à des variables ;
  - Exécutée une fois pour chaque nœud renvoyé par les clauses FOR/LET/WHERE.

# IV. Les expressions

- Syntaxe :
- Explication
  - **for** *a* **in** *f* **where** *c* **return** *a'*:
    - **for** : permet d'accéder à un arbre *a* de la forêt *f*
    - **where** : permet de tester si l'arbre *a* vérifie la condition *c*
    - **return** : permet de construire un nouvel arbre *a'*, à partir de l'arbre *a*

```
(for $x1 in f1, ..., $xn in fn : itérations imbriquées
| let $y1 := l1, ..., $ym := lm) : variables locales
(where w)? : condition
(order by o1 d1, ..., om dk)? : clés de tri
return r : item à retourner
```

où :

$x_1, \dots, x_n, y_1, \dots, y_k$  sont des noms de variables,  
 $f_1, \dots, f_n, l_1, \dots, l_m, w, o_1, \dots, o_k, r$  sont des expressions,  
 $d_i \{ \text{ascending, descending, ...} \}$  spécifie la direction du tri.



## IV. Les expressions

- Exemple 1 : noms des auteurs du livre 3 (R6.xq, R6b.xq)
- Contenu de la séquence :

### Requête

```
let $al :=  
document("bib.xml")//book[3]/author/last  
return  
<livre nbAuteurs="{ count($al)} ">  
  { $al }  
</livre>
```

```
ou  
for $b in document("bib.xml")//book[3]  
let $al := $b/author/last  
return  
<livre nbAuteurs="{ count($al)} ">  
  { $al }  
</livre>
```

- Contenu de la séquence :

```
$b =  
(<last>...</last>,  
  <last>...</last>,...,  
  <last>...</last>  
)
```

### Résultat

```
<livre nbAuteurs="3">  
  <last>Abiteboul</last>  
  <last>Buneman</last>  
  <last>Suciu</last>  
</livre>
```

## IV. Les expressions

- Exemple 2 : nom de tous les auteurs de livre (Rq7.xq)

Requête
<pre>&lt;auteurs&gt; { for \$b in document("bib.xml")//book   let \$al := \$b/author/last   return     &lt;livre nbAuteurs="{ count(\$al)}" &gt;       { \$b/title, \$al }     } &lt;/auteurs&gt;</pre>

Résultat
<pre>&lt;auteurs&gt;   &lt;livre nbAuteurs="1" &gt;     &lt;title&gt;TCP/IP Illustrated&lt;/title&gt;     &lt;last&gt;Stevens&lt;/last&gt;   &lt;/livre&gt;   &lt;livre nbAuteurs="1" &gt;     &lt;title&gt;Advanced Programming in the Unix environment&lt;/title&gt;     &lt;last&gt;Stevens&lt;/last&gt;   &lt;/livre&gt;   &lt;livre nbAuteurs="3" &gt;     &lt;title&gt;Data on the Web&lt;/title&gt;     &lt;last&gt;Abiteboul&lt;/last&gt;     &lt;last&gt;Buneman&lt;/last&gt;     &lt;last&gt;Suciu&lt;/last&gt;   &lt;/livre&gt;   ... &lt;/auteurs&gt;</pre>

## IV. Les expressions

- Exemple 3 : nom des livres dont le nom de l'auteur est Stevens (Rq8.xq).

### Requête

```
<livres>
{ for $a in document("bib.xml")//book
  for $b in $a/author
  where $b/last = "Stevens"
  return $a/title
}
</livres>
```

### Résultat

```
<livres>
  <title>TCP/IP Illustrated</title>
  <title>Advanced Programming in the
Unix environment</title>
</livres>
```

# IV. Les expressions

---

- ▣ 4.15. Expression conditionnelle : if-then-else
  - Exemple : livres de Abiteboul publiés après 1994 (Rq9.xq).

## Requête

```
<livres>
{ for $b in document("bib.xml")//book
  where $b/author/last = "Abiteboul"
  return
    if ($b/@year gt "1994") then
      <livre recent="true"> { $b/title} </livre>
    else
      <livre> { $b/title} </livre>
}
</livres>
```

## Résultat

```
<livres>
  <livre recent="true">
    <title>TCP/IP Illustrated</title>
  </livre>
  <livre>
    <title>Advanced Programming in
the Unix environment</title>
  </livre>
</livres>
```

# IV. Les expressions

## □ 4.16. Expressions de quantification

- **some** \$var **in** expr1 **satisfies** expr2
  - il existe au moins un nœud retourné par l'expression expr1 qui satisfait expr2.
- **every** \$var **in** expr1 **satisfies** expr2
  - tous les nœuds retournés par l'expression expr1 satisfont expr2
- Exemple : nom des auteurs dont la maison d'édition est "Morgan Kaufmann Publishers" (Rq10.xq)

### Requête

```
<auteurs>
{ for $a in document("bib.xml")//author
  where every $b in
document("bib.xml")//book[author/last = $a/last]
satisfies $b/publisher="Morgan Kaufmann Publishers"
  return <nom>{ string($a/last)} </nom>
}
</auteurs>
```

### Résultat

```
<auteurs>
  <nom>Abiteboul</nom>
  <nom>Buneman</nom>
  <nom>Suciu</nom>
  <nom>Scholl</nom>
  <nom>Voisard</nom>
</auteurs>
```

# IV. Les expressions

## ■ 4.17. Expression de tri

### ■ Syntaxe

- `Expr1 order by (Expr2)ascending | descending`
  - Trier les éléments de `Expr1` par les valeurs retournées par `Expr2`.

- Exemple : titre et année d'édition des livres triés par année décroissante (Rq11.xq).

### Requête

```
<livres>
{ for $b in document("bib.xml")//book
order by ($b/@year)descending return
<livre>
{ $b/@year, $b/title }
</livre>
}
</livres>
```

### Résultat

```
<livres>
<livre><title>Comprendre XSLT</title> </livre>
<livre year="2001">
<title>Spatial Databases</title></livre>
<livre year="2000"><title>Data on the Web</title></livre>
<livre year="1999">
<title>The Technology and Content for Digital TV</title>
</livre>
<livre year="1994"><title>TCP/IP Illustrated</title></livre>
<livre year="1992">
<title>Advanced Programming in the Unix environment</title>
</livre>
</livres>
```

# IV. Les expressions

---

- ▣ 4.18. Jointure
  - Fichier adresse.xml

```
<adresse>
  <personne>
    <nom>Stevens</nom>
    <pays>France</pays>
    <institution>CNAM</institution>
  </personne>
  <personne>
    <nom>Abiteboul</nom>
    <pays>France</pays>
    <institution>CNAM</institution>
  </personne>
  <personne>
    <nom>Buneman</nom>
    <pays>Germany</pays>
    <institution>FU Berlin</institution>
  </personne>
</adresse>
```

## IV. Les expressions

- Exemple : titre des livres, nom et institution des auteurs.
  - Nœuds non calculés (Rq12.xq)

### Requête : Rq12.xq

```
<livres>
{ for $b in
document("bib.xml")//book
return
  <livre titre="{ $b/title} ">
    { for $a in $b/author
      for $p in
doc("adresse.xml")//personne
return
  if($a/last = $p/nom) then
    <auteur nom="{ $a/last}"
institut= "{ $p/institution}" />
    else
      ()
    }
  }
</livre>
}
</livres>
```

### Résultat

```
<livres>
  <livre titre="TCP/IP Illustrated">
    <auteur nom="Stevens" institut="CNAM"/>
  </livre>
  <livre titre="Advanced Programming in the Unix
environment">
    <auteur nom="Stevens" institut="CNAM"/>
  </livre>
  <livre titre="Data on the Web">
    <auteur nom="Abiteboul" institut="CNAM"/>
    <auteur nom="Buneman" institut="FU Berlin"/>
  </livre>
  <livre titre="The Technology and Content for
Digital TV"/>
  <livre titre="Comprendre XSLT"/>
  <livre titre="Spatial Databases"/>
</livres>
```



## IV. Les expressions

- Exemple 2 : titre des livres, nom et institution des auteurs.
  - ▣ Nœuds calculés (Rq12b.xq)

### Requête : Rq12b.xq

```
<livres>
{ for $b in document("bib.xml")//book
  return element livre
    { attribute titre { $b/title},
      for $a in $b/author
        return element auteur
          { attribute nom { $a/last},
            for $p in
              document("adresse.xml")//personne
                return if ($a/last = $p/nom) then
                  attribute institut
                    { $p/institution}
              else ()
            }
          }
    }
}
```

### Résultat

```
<livres>
  <livre titre="TCP/IP Illustrated">
    <auteur nom="Stevens"
institut="CNAM"/>
  </livre>
  <livre titre="Advanced Programming in
the Unix environment">
    <auteur nom="Stevens"
institut="CNAM"/>
  </livre>
  <livre titre="Data on the Web">
    <auteur nom="Abiteboul" institut="FU
Berlin"/>
    <auteur nom="Buneman" institut="FU
Berlin"/>
    <auteur nom="Suciu"
institut="CNAM"/>
  </livre>
  ...
</livres>
```

## IV. Les expressions

- Exemple 3 : titre des livres, nom, pays et institution des auteurs (Rq12c.xq).

### Requête : Rq12c.xq

```
<livres>
{ for $b in document("bib.xml")//book
  return element livre
    { attribute titre { $b/title},
      for $a in $b/author
        return element auteur
          { attribute nom { $a/last},
            for $p in
              document("adresse.xml")//personne
                return
                  if ($a/last = $p/nom) then
                    (attribute institut { $p/institution},
                     element pays { string($p/pays)})
                  else ()
            }
          }
    }
  }
</livres>
```

### Résultat

```
<livres>
  <livre titre="TCP/IP Illustrated">
    <auteur nom="Stevens" institut="CNAM">
      <pays>France</pays></auteur>
    </livre>
  <livre titre="Advanced Programming in the
  Unix environment">
    <auteur nom="Stevens" institut="CNAM">
      <pays>France</pays></auteur>
    </livre>
  <livre titre="Data on the Web">
    <auteur nom="Abiteboul" institut="FU
  Berlin">
      <pays>Germany</pays>
    ...
  </livres>
```

# IV. Les expressions

## □ 4.19. Fonctions

- Les fonctions et opérateurs sont
  - Typées (XML schema) et manipulent des séquences et des valeurs typées : entiers, chaînes de caractères, dates, . . .
- Fonction prédéfinie : 4 catégories :
  - chaînes de caractères (string) ;
  - nombres (number) ;
  - booléens (boolean) ;
  - nœuds (node).
- Catégorie "node"

Nom	Rôle
count()	Compte les éléments de la sélection
id()	Permet de sélectionner des éléments par leur identifiant
last()	Dernier élément d'une sélection
position()	Renvoie l'index de la position du nœud relativement au nœud parent
text()	Valeur d'un élément
name()	Nom d'un élément

## IV. Les expressions

### ■ Catégorie "string"

Nom	Rôle	Exemple	Résultat
concat()	Retourne la concaténation d'arguments	concat("About", " ", "XML")	'About XML'
contains()	Retourne vrai si la première string contient la deuxième	contains("XML", "X")	true
string()	Convertit un objet en string	string(12.20)	'12,20'
substring()	Retourne une sous string	substring("XML",2,3)	'ML'
translate()	Convertit des caractères d'une string	translate("XML","X","W")	'WML'

### ■ Catégorie "number"

Nom	Rôle	Exemple	Résultat
round()	Arrondi un nombre à virgule	round(12.20)	12
number()	Convertit un argument en nombre	number(price)	Noeud 'price' convertit en nombre
sum()	Retourne la somme de chaque nœud appartenant au jeu de nœuds passé en argument	sum(//book/price)	85.5

## IV. Les expressions

- Catégorie "boolean"

Nom	Rôle
boolean()	Converti un argument en booléen
false()	Retourne faux
true()	Retourne vrai
not()	Retourne vrai si son argument est faux, ou faux dans le cas contraire

- Exemple 1 : prix moyen des livres d'un éditeur (AVG)

- Rq13.xq

Requête
<pre>&lt;prixMoyen&gt; { for \$p in distinct-values(document("bib.xml")//publisher) let \$l: = document("bib.xml")//book[publisher = \$p] return element publisher { attribute name { string(\$p)}, attribute avgPrice { avg(\$l/price) }} } &lt;/prixMoyen&gt;</pre>
Résultat
<pre>&lt;prixMoyen&gt; &lt;publisher name="Addison-Wesley" avgPrice="68.45" /&gt; &lt;publisher name="Morgan Kaufmann Publishers" avgPrice="37.475" /&gt; &lt;publisher name="Kluwer Academic Publishers" avgPrice="129.95" /&gt; &lt;publisher name="OReilly" avgPrice="28.95" /&gt; &lt;/prixMoyen&gt;</pre>

## IV. Les expressions

- Exemple 2 : somme des prix des livres d'un éditeur

- ▣ Rq13b.xq

Requête
<pre>&lt;montant&gt; { for \$p in distinct-values(document("bib.xml")//publisher)   let \$l := document("bib.xml")//book[publisher = \$p]   return element publisher { attribute name { string(\$p) } ,                            attribute sumPrice { sum(\$l/price) } } } &lt;/montant&gt;</pre>
<pre>&lt;montant&gt; &lt;publisher name="Addison-Wesley" sumPrice="136.9"/&gt; &lt;publisher name="Morgan Kaufmann Publishers" sumPrice="74.95"/&gt; &lt;publisher name="Kluwer Academic Publishers" sumPrice="129.95"/&gt; &lt;publisher name="OReilly" sumPrice="28.95"/&gt; &lt;/montant&gt;</pre>

# IV. Les expressions

## ■ Fonctions définies par l'utilisateur

### ■ Syntaxe

```
declare function  
  local:nomFonction($arg1 [as  
    type1], ...) [as typen]  
  { ... }
```

### ■ Les différents types :

- xs:integer, xs:double ,  
 xs:string, xs:date, xs:boolean  
 xs:anyURI

### ■ Exemple 1 : nombre des auteurs (Rq14.xq)

### Requête

```
declare function local:nombreAuteurs($b) as xs:integer  
{ count(distinct-values($b/author)) };  
  
(: appel de la fonction :)  
let $a := document("bib.xml")//book  
return  
  <nbAuteurs>{ local:nombreAuteurs($a)} </nbAuteurs>
```

### Résultat

```
<nbAuteurs>  
  8  
</nbAuteurs>
```

## IV. Les expressions

- Exemple 2 : nombre des auteurs du 3<sup>e</sup> livre (Rq14.xq)

Requête
<pre>declare function local:nombreAuteurs(\$b) as xs:integer { count(distinct-values(\$b/author)) };  (: appel de la fonction :) let \$a := document("bib.xml")//book[3] return   &lt;nbAuteurs&gt;{ local:nombreAuteurs(\$a) } &lt;/nbAuteurs&gt;</pre>

Résultat
<pre>&lt;nbAuteurs&gt; 3 &lt;/nbAuteurs&gt;</pre>

- Exemple 3 : prix d'un livre connaissant son titre (Rq14b.xq).

Requête
<pre>declare function local:prix(\$nom) { document("bib.xml")//book[title = \$nom]/price }; (: appel de la fonction :) &lt;lesPrix&gt; { let \$d := document("bib.xml")//book   for \$m in \$d/title     let \$p := local:prix(\$m)     return (&lt;books&gt;{ \$m, \$p } &lt;/books&gt;) } &lt;/lesPrix&gt;</pre>

Résultat
<pre>&lt;lesPrix&gt;   &lt;books&gt;&lt;title&gt;TCP/IP   Illustrated&lt;/title&gt;&lt;price&gt;65.95&lt;/price&gt;&lt;/books&gt;   &lt;books&gt;     &lt;title&gt;Advanced Programming in the Unix     environment&lt;/title&gt;     &lt;price&gt;70.95&lt;/price&gt;   &lt;/books&gt;   ... &lt;/lesPrix&gt;</pre>



## IV. Les expressions

- Exemple 4 : prix d'un livre connaissant son titre (Rq14c.xq).

Requête
<pre>declare function local:prix(\$nom) { document("bib.xml")//book[title = \$nom]/price };  (: appel de la fonction :) &lt;lesPrix&gt; { let \$d := document("bib.xml")//book   for \$m in \$d/title     let \$p := local:prix(\$m)     return (&lt;books&gt;{ \$m, \$p } &lt;/books&gt; ) } &lt;/lesPrix&gt;</pre>

Résultat
<pre>&lt;lesPrix&gt;   &lt;books&gt;&lt;title&gt;TCP/IP   Illustrated&lt;/title&gt;&lt;price&gt;65.95&lt;/price&gt; &lt;/books&gt;   &lt;books&gt;     &lt;title&gt;Advanced Programming in the     Unix environment&lt;/title&gt;     &lt;price&gt;70.95&lt;/price&gt;   &lt;/books&gt;   ... &lt;/lesPrix&gt;</pre>

# IV. Les expressions

## □ 4.20. Fonctions récursives

- Exemple : calcul de  $n! = n * (n-1)!$

□ Rq15.xq

Requête
<pre>declare function local:fac(\$n as xs:integer) as xs:integer { if (\$n &lt; 0) then   0   else     if (\$n = 0) then       1     else       \$n * local:fac(\$n - 1) }; &lt;resultats&gt; { for \$x in 1 to 10   return     &lt;fac&gt;fac({ \$x }) = { local:fac(\$x) } &lt;/fac&gt; } &lt;/resultats&gt;</pre>

Résultat
<pre>&lt;resultats&gt;   &lt;fac&gt;fac(1)=1&lt;/fac&gt;   &lt;fac&gt;fac(2)=2&lt;/fac&gt;   &lt;fac&gt;fac(3)=6&lt;/fac&gt;   &lt;fac&gt;fac(4)=24&lt;/fac&gt;   &lt;fac&gt;fac(5)=120&lt;/fac&gt;   &lt;fac&gt;fac(6)=720&lt;/fac&gt;   &lt;fac&gt;fac(7)=5040&lt;/fac&gt;   &lt;fac&gt;fac(8)=40320&lt;/fac&gt;   &lt;fac&gt;fac(9)=362880&lt;/fac&gt;   &lt;fac&gt;fac(10)=3628800&lt;/fac&gt; &lt;/resultats&gt;</pre>

## IV. Les expressions

- Exemple 2 : profondeur d'un arbre
  - ▣ Rq15b.xq
  - ▣ La racine est de profondeur 0

### Requête

```
declare function local:depth($n) as xs:integer
{ if (empty($n/*)) then
  1
  else
    max(for $c in $n/* return local:depth($c)) + 1
};

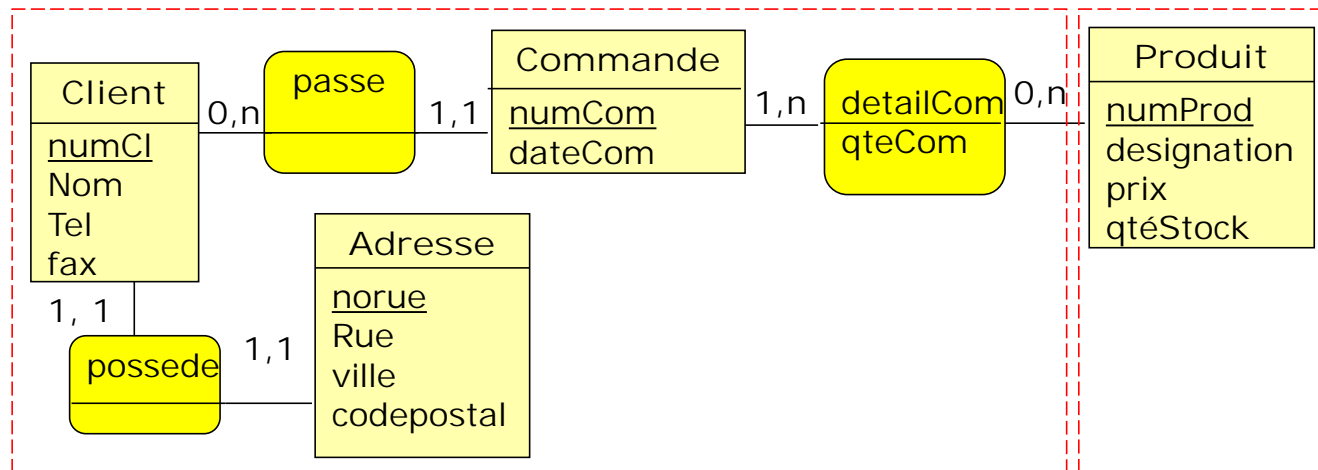
<profondeur>
{ let $x := document("bib.xml")//book
  return local:depth($x)
}
</profondeur>
```

### Résultat

```
<profondeur> 3</profondeur>
```

# IV. Les expressions

## □ TD7 + TP7



### 1. XML

- Créer les documents XML correspondants (2 documents).
- Ajouter 5 produits, 3 clients, 2 commandes par client, 2 produits par commande.

# IV. Les expressions

---

## 2. Requêtes :

- Rq1 : Liste des clients.
- Rq2 : Liste des clients (numCl, nom).
- Rq3 : Liste des clients avec leur adresse (numCl, nom, adresse).
- Rq4 : Liste des clients (numCl, nom, adresse) avec leurs commandes (numCom, dateCom).
- Rq5 : Lister tous les numéros, nom, adresse des clients avec leurs commandes (numCom, dateCom) et le détail des commandes.
- Rq6 : Lister tous les numéros, nom, adresse du client Toto avec ses commandes (numCom, dateCom) et le détail de ses commandes (numProd, qteCom).
- Rq7 : numéros, nom, adresse du client Toto avec ses commandes (numCom, dateCom) et le détail de ses commandes (numProd, désignation, qteCom, prix, qteCom\*prix).
- Rq8 : numéro, nom, adresse du client Toto avec ses commandes (numCom, dateCom) et le détail de sa commande de numéro 1 (numProd, désignation, qteCom, prix, qteCom\*prix)
- Rq9 : Ecrire une fonction qui détermine le nombre de commande d'un client donné.
- Rq10 : Ecrire une fonction qui liste le numéros, nom, adresse du client Toto avec sa commande numéro 1 (numCom, dateCom) et le détail de cette commandes (numProd, désignation, qteCom, prix, qteCom\*prix).

## IV. Les expressions

---

- ▣ Rq11 : Montant total d'une commande d'un client donné.
- ▣ Rq12 : Montant total de toutes les commandes d'un client donné.
- ▣ Rq13 : Créer un module contenant les fonctions issues des requêtes Rq9 à Rq12.

# IV. Les expressions

---

## ▣ 4.21. Module

### ■ Collection de fonctions et de déclarations

#### ▣ Rq16.xq

#### Module

```
module namespace  
biblio="http://www.Cours/Cours2011/XML/Support/Sources/SourceXQ";
```

(: Déclaration de module :)

```
declare function biblio:nombreAuteurs($b) as xs:integer  
{ count(distinct-values($b/author))};
```

```
declare function biblio:prix($nom)  
{ doc("bib.xml")//book[title = $nom]/price};
```

```
declare variable $biblio:TestVariable := 7.5;
```

# IV. Les expressions

## ■ Utilisation (Rq16b.xq)

```
import module namespace
biblio="http://www.Cours/Cours2011/XML/Support/
Sources/SourceXQ" at "Rq16.xq";
(: Utilisation de module :)
<lesPrix>
{ let $d := doc("bib.xml")//book
  for $m in $d/title
    let $p := biblio:prix($m)
    return (<books>{ $m, $p} </books>)
}
</lesPrix>
```

```
<lesPrix
xmlns:biblio="http://www.cours/cours2010/XML/X
ML_2009/source/sourceXQ" >
  <books><title>TCP/IP
  Illustrated</title><price>65.95</price></books>
  <books>
    <title>Advanced Programming in the Unix
  environment</title><price>70.95</price>
  </books>
  <books><title>Data on the
  Web</title><price>39.95</price></books>
  <books>
    <title>The Technology and Content for Digital
  TV</title><price>129.95</price>
  </books>
  <books><title>Comprendre
  XSLT</title><price>28.95</price></books>
  <books><title>Spatial
  Databases</title><price>35.00</price></books>
</lesPrix>
NB. L'attribut de la balise lesPrix est visible avec
Galax en mode commande et non GalaxServe
```



# V. Application : Recherche de composants

---

## ■ 5.1. Présentation

- Le développement des ressources de « l'informatique libre » est très rapide et engendre des composants de très grande qualité.

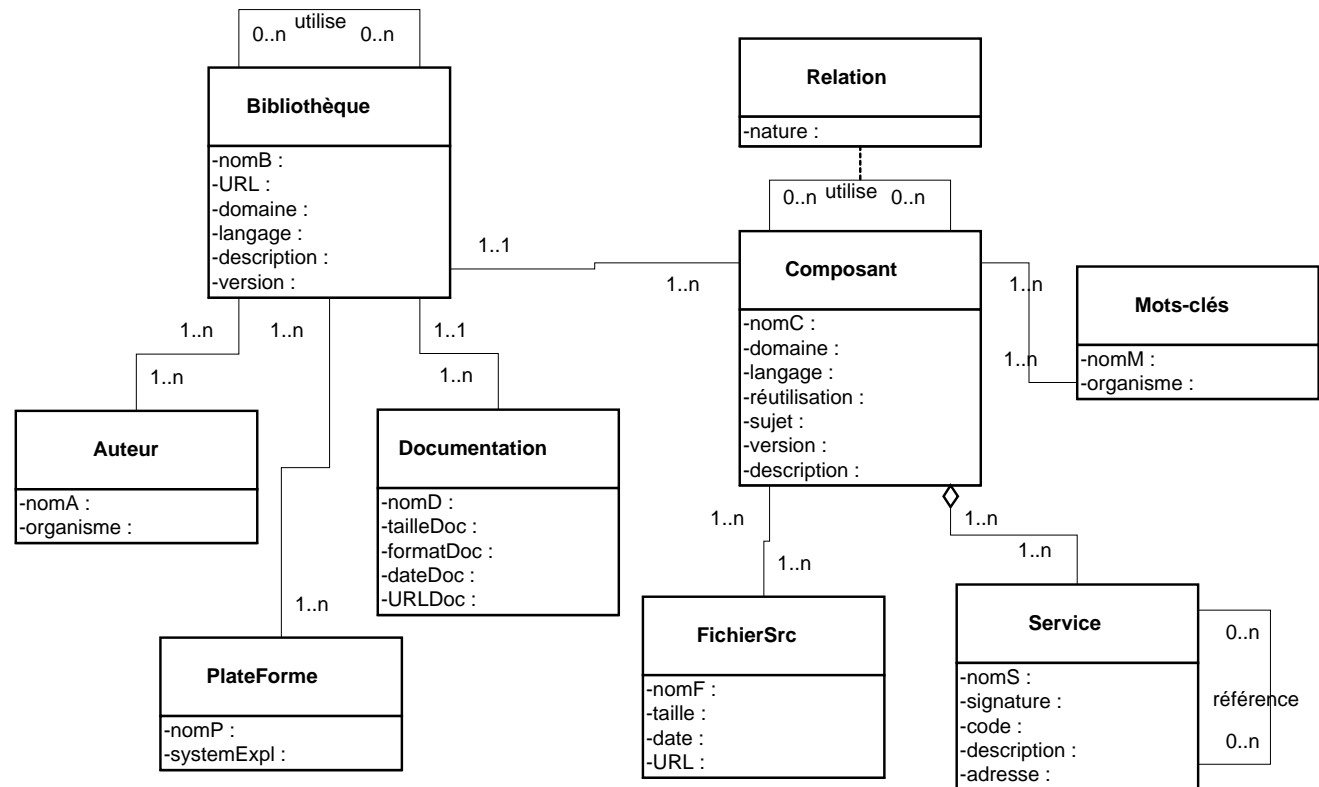
- Un composant est un logiciel qui rend un service au moyen d'une interface externe.
- Pour nous un composant est une classe ou une fonction.
- Nous nous intéressons à la réutilisation de ces composants afin de réduire le coût de production et de maintenance des logiciels.

## ■ Problèmes rencontrés lors de la réutilisation de composants

- Liés à leur disponibilité, au temps de développement, à la recherche et à la sélection du composant approprié.
- Comment tirer le meilleur profit de ces ressources ?
  - une plate-forme opérationnelle doit pouvoir en premier lieu disposer d'un système de recherche d'information efficace permettant de trouver le bon composant et sa documentation.

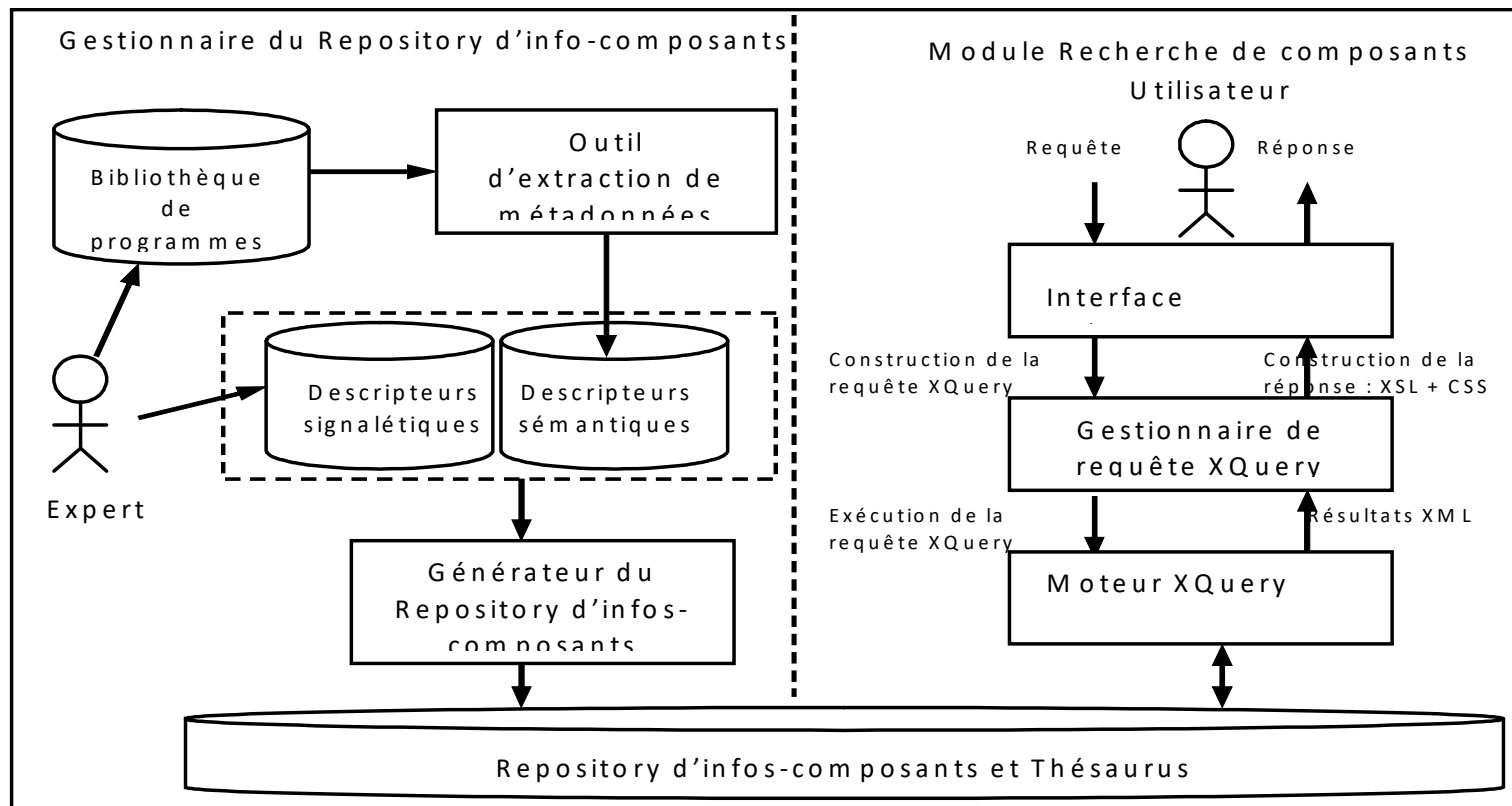
# V. Application : Recherche de composants

## ■ Modèle de données



# V. Application : Recherche de composants

## 5.2. Architecture du système



# V. Application : Recherche de composants

---

- Outil d'extraction :
  - Il permet d'extraire la description sémantique d'une bibliothèque et de ses composants.
- Descripteurs sémantiques
  - Informations qui ne peuvent être extraites automatiquement : composant, méthodes, fichiers sources...
- Descripteurs signalétiques
  - Informations qui ne peuvent être extraites que manuellement (données signalétiques) : Bibliothèque, Documentation, Auteur, Plate-forme, Mots-clés...
- Générateur du Repository d'infos-composants :
  - Permet de générer automatiquement un info-composant (document XML) à partir des descriptions signalétiques et sémantique d'une bibliothèque et de ses composants.

# V. Application : Recherche de composants

---

## ■ Module de recherche

- ▣ Chargé de la construction des requêtes, de leur exécution, de la construction et de la présentation du résultat à l'utilisateur.

### ▣ Interface utilisateur :

- Permet à l'utilisateur de créer une requête XQuery sans connaître le schéma XML du catalogue.
- Ses fonctions sont : recevoir une requête basée sur les termes du thesaurus du domaine ; avoir le résultat de la requête en XML du Gestionnaire de requêtes XQuery, le convertir en HTML à l'aide de feuilles de style XSL et CSS, et l'afficher sur l'écran.

## ▣ Gestionnaire de requêtes XQuery :

- (1) traduire une requête utilisateur en XQuery avant de la passer au Moteur XQuery ; (2) recevoir la réponse de la requête du Moteur XQuery ; (3) transformer les résultats en données XML ; (4) Retourner le résultat en XML à l'interface utilisateur.

## ▣ Moteur XQuery :

- Permet d'exécuter la requête XQuery et de délivrer le résultat XML au gestionnaire de requêtes XQuery.

# V. Application : Recherche de composants

## ■ Exemple d'info-composant MV++.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/css" href="catalogue.css"?>
<bibliotheque id="MV++">
  <nomB>MV++</nomB>
  <url>C:\MDMS\Bibliotheques\mv</url>
  <dohtml>C:\MDMS\Bibliotheques\mv\html</dohtml>
  <composant id="1" domaine="" sujet="" version="">
    <nomC>FMV_Vector< > TYPE > </nomC>
    <fichierSrc>
      <nomF>mvvtp.h</nomF>
      <taille>13.1Ko </taille>
      <date>23:07:04</date>
```

```
<?
<urlF>C:\MDMS\Bibliotheques\mv\include\mvvtp.h</urlF>
  </fichierSrc>
  <service id="1" >
    <nomS>FMV_Vector</nomS>
    <signature>template<class TYPE> FMV_Vector (
unsigned int n )</signature>
    <adresse>
      <debut>00382 </debut>    <fin>00382 </fin>
      <fichier>mvvtp.h </fichier>
    </adresse>
  </service>
...
```

# V. Application : Recherche de composants

- Interface Web
  - Alias PsRep sur  
C:\PSRep\PSRep\_Web\SearchInterface



# V. Application : Recherche de composants

## ■ Interface d'interrogation

The screenshot displays a web browser window titled "Components Search System" with the address bar showing "localhost/itskep/admin.htm". The interface is divided into two main panels:

- Components Search Interface (Left Panel):**
  - Thesaurus terms:** A dropdown menu set to "Matrix" and a "Search" button.
  - Descriptors:**
    - Library:** A dropdown menu set to "Lapack++".
    - Component:** A dropdown menu set to "LaBandFactDouble".
    - Service:** A dropdown menu set to "LaBandFactDouble".
  - Search:** A "Search" button.
  - Keywords:** A section with labels "Descriptors:", "Library:", "Component:", and "Service:" followed by empty input fields and a "Search" button.
  - Footer:** Links for "Home", "Back", and "Forward", followed by contact information: "Amrane HOCINE, hocine@univ-pau.fr - Konan Marcellin BROU, brou@uniph.edu.ci".
- Components Management Results Interface (Right Panel):**
  - Header:** "Components Management Results Interface".
  - Content:** A list of results: "Amrane HOCINE, hocine@univ-pau.fr - Konan Marcellin BROU, brou@uniph.edu.ci".



# V. Application : Recherche de composants

## ■ Utilisation du thesaurus

The screenshot displays a web browser window titled 'Components Search Syst.' with the address bar showing 'localhost/PSRep/admin.htm'. The interface is divided into two main panels: 'Components Search Interface' on the left and 'Components Search Results' on the right.

**Components Search Interface:**

- Thesaurus terms:** A dropdown menu shows 'Rectangular matrix' with a 'Search' button next to it.
- Descriptors:**
  - Library:** A dropdown menu shows 'Lapack++'.
  - Component:** A dropdown menu shows 'LaBandFactDouble' with a 'Search' button.
  - Service:** A dropdown menu shows 'LaBandFactDouble'.
- Keywords:** A section with a 'Search' button and radio buttons for 'Library', 'Component', and 'Service'.
- Navigation:** Links for 'Home', 'Back', and 'Forward' are present at the bottom.

**Components Search Results:**

- Header:** 'The following components have been found for "Rectangular matrix"'.
- Search Bar:** A dropdown menu shows 'LaGenMatComplex' with a 'Search' button.
- Navigation:** Links for 'Library', 'Authors', 'Sources', 'Component', 'Services', and 'Signatures' are present.
- Results:** A list of authors is displayed: 'Amrane HOCINE, hocine@univ-pau.fr - Konan Marcellin BROU, brou@inphb.edu.ci'.

# V. Application : Recherche de composants

## ■ Résultat de la recherche sur un service

The screenshot displays a web application titled "Components Search Interface". On the left, there are search filters: "Thesaurus terms" with a dropdown set to "Matrix" and a "Search" button; "Descriptors" with dropdowns for "Library" (set to "Lapack++"), "Component" (set to "LaBandFactDouble"), and "Service" (set to "LaBandFactDouble"), along with a "Search" button; and "Keywords" with a text input and a "Search" button. At the bottom left are links for "Home", "Back", and "Forward", and contact information for Amrane HOUCHE and Konan Marcellin BROU.


On the right, a table titled "Services LaBandFactDouble informations" lists search results. The table has columns for Library, Component, Signature, Address (Begin and End), and Source.

Library	Component	Signature	Address		Source
			Begin	End	
Lapack++	LaBandFactDouble	LaBandFactDouble ( ) [inline]	00047	00054	bfd.h
Lapack++	LaBandFactDouble	LaBandFactDouble ( int , int , int ) [inline]	00057	00065	bfd.h
Lapack++	LaBandFactDouble	LaBandFactDouble ( LaBandMatDouble & ) [inline]	00068	00077	bfd.h
Lapack++	LaBandFactDouble	LaBandFactDouble ( LaBandFactDouble & ) [inline]	00080	00089	bfd.h

At the bottom right, the same contact information for Amrane HOUCHE and Konan Marcellin BROU is repeated.

# V. Application : Recherche de composants

## ■ Accès au code source



The screenshot displays a web browser window with the address bar showing `localhost/TsRep/admin.htm`. The page title is "Components Search Interface". The interface includes a "Thesaurus terms:" section with a dropdown menu set to "Matrix" and a "Search" button. Below this is a "Descriptors:" section with three dropdown menus: "Library:" set to "Lapack++", "Component:" set to "LaGenMatDouble", and "Service:" set to "LaBandFactDouble". A "Search" button is also present. At the bottom of the interface, there are links for "Home", "Back", and "Forward", and a footer with the text "Amrane HOCINE, hocine@univ-pau.fr" and "Kenan Mareslim Brou, brou@inphb.cau.ac.cn".

On the right side of the browser window, the source code for the component is displayed. The code is as follows:

```
inline LaBandFactDouble::LaBandFactDouble(int N, int k1, int ku)
: b_(N, k1, ku), pivot_(k1, ku+1)
{
    #ifndef BANDFACTDOUBLE_DEBUG
        cout << " called LaBandFactDouble::LaBandFactDouble(int,int,int) "
    << endl;
    #endif
    into_ = 0;
}
```

# Bibliographie

---

## □ Livres

- "XML, Langage et applications", Alain Michard, Eyrolles.

## □ Webographie

- Site du W3C sur XQuery: <http://www.w3.org/XML/Query>
- [http:// www.lri.fr/~benzaken/documents/xquery.pdf](http://www.lri.fr/~benzaken/documents/xquery.pdf)
- <http://cui.unige.ch/eao/www/xml/Plan.html#main>
- <http://h2ptm.hypermedia.univ-paris8.fr/mkadmi/coursXML/sommaire.htm>
- <http://wam.inrialpes.fr/courses/VQ-EcoleEte3-4-Jun03/slide4-0.html>
- <http://gilles.chagnon.free.fr/cours/xml/schema.html>
- <http://www.galaxquery.org/>
- <http://www.w3.org/TR/xmlquery-use-cases/>
- [http://css.mammothland.net/css\\_de\\_base.php](http://css.mammothland.net/css_de_base.php)