

Spécifications liées à XML

Standards de base

- SGML (p. 2) - XML (p. 5) - Unicode XML (p. 8)

Définition de modèles

- Namespace (p. 10)- Schema (p. 13)- DTD (p. 18)- DSDL (p. 21)- RELAX NG (p. 23)

Représentation et fragmentation

Représentation

- Canonical (p. 24) - Infoset (p. 26)

Fragmentation

- XML Catalogs (p. 29) - Fragment Interchange (p. 31) - XInclude (p. 33)

Utilisation des documents XML

Construction

Identification des contenus

- XML Language (p. 35) - XPointer (p. 37)

Accès aux contenus

- XLink (p. 39) - XPath (p. 41)

Exploitation

Traitements

- DOM (p. 44) - SAX (p. 48) - XProc (p. 50) - XSLT (p. 51)

Présentation

- DSSSL (p. 55) - XSL (p. 57)

Accès aux contenus

- XML Query (p. 61)

SGML, Langage normalisé de balisage généralisé

Rédaction : Pierre Attar

Comme XML, SGML est un langage de codage de données dont l'objectif est de permettre, dans un échange entre systèmes informatiques, de transférer, en même temps, des données textuelles et leurs structures. SGML était très utilisé avant qu'XML n'existe : c'est d'ailleurs à partir de ce langage (et en tenant compte de ses différentes utilisations) qu'a été élaboré le standard XML.

On a dit beaucoup de choses sur SGML : que le langage était lourd, que les implémentations étaient difficiles, etc. De fait, ce n'était pas la technologie qui était lourde, mais les applications elles-mêmes. En effet, s'il est nécessaire de manipuler un manuel de maintenance aéronautique de quelques centaines de méga-octets, avec des arbres d'une profondeur allant jusqu'à des dizaines de niveaux, la technologie n'y fait rien (qu'elle soit XML ou SGML) : les manipulations sont de toutes les façons complexes !

C'est ce dont est en train de se rendre compte la communauté XML, qui comprend que ce standard sera utilisé pour différentes classes d'applications : depuis l'échange de messages entre applications, avec des informations de l'ordre du kilo-octet jusqu'à des documents beaucoup plus lourds et beaucoup plus sémantiques, comme le sont un dictionnaire ou un ouvrage juridique.

En termes de technologie, la lourdeur de SGML est certainement liée à ses DTD qui mélangeaient deux type d'information : la notion de modèle de donnée et le format de codage des documents conformes à ce modèle. Par ailleurs, le langage lui-même proposait beaucoup de facilités de saisie simplifiées des noms d'élément : toute chose difficiles à prendre en compte dans les outils de traitement.

D'un point de vue technique donc, la grande avancée conceptuelle de XML est donc de différencier définition de modèle et format de codage des documents. C'est en cela que XML supprime des lourdeurs de SGML.

L'avenir de SGML ? D'un point de vue technique, ce format peut sans problème être remplacé par XML : tout ce qui est nécessaire à SGML existe à peu près dans XML. D'un point de vue pratique, les utilisateurs de SGML y gagneront en diversité d'outils utilisables et apprécieront à son juste avantage la prise en compte par XML d'Unicode XML.

D'un point de vue plus stratégique, beaucoup se posent la question de passer de SGML à XML, en raison des organismes de normalisation qui s'en occupent. Pour SGML, c'est l'ISO qui assure la normalisation, avec ses processus de maturation d'une norme et ses représentations nationales assurant la qualité des normes ; en ce qui concerne la normalisation de XML, c'est le travail du W3C, un Consortium qui n'est représentatif que de

lui-même et qui, comme son nom l'indique, ne s'intéresse qu'au point de vue du Web. Cet argument devrait rapidement être balayé par la volonté commune des deux organismes de promouvoir XML au sein des normes documentaires de l'ISO.

Objectifs

L'objectif de SGML était de définir un format neutre de rétention d'information chez un photocomposeur. ce format neutre devait permettre aux éditeurs de mieux faire jouer la concurrence et d'autoriser un éditeur à renégocier ses contrats avec les photocomposeurs pour, par exemple, pouvoir transférer ses documents chez celui qui proposerait le meilleur prix. Par la suite, et comme pour XML, l'objectif de SGML fut de définir un format neutre de codage de document permettant de mettre en place des notions de réutilisation de la même information pour différentes publications, sur différents médias.

Aujourd'hui, l'objectif de SGML est le même que celui de XML et cette norme est appelée à être remplacée par XML.

D'un point de vue pratique, tout document SGML peut être transformé de façon totalement automatisée en document XML. Les DTD de SGML devront subir quelques modifications qui les rendront certainement parfois plus laxiste. En effet, XML ne permet pas de prendre en compte les mécanismes d'inclusion et d'exclusion" propres à SGML. Il permet aussi un typage des attributs plus restreint. Cette dernière limite est maintenant bien levée avec la standardisation des Schema.

Principes

À la différence d'XML, un document SGML doit toujours être valide au regard d'une DTD. Les concepteurs, en raison de leur orientation documentaire, s'intéressaient beaucoup à la notion de validation électronique et automatique. Cette validation au regard d'une DTD permet de s'assurer que l'information est saisie conformément au modèle et que l'on peut donc lui appliquer les traitements automatisés sans avoir à valider la structure du document au préalable. A noter que cela reste vrai pour XML mais que cette dernière recommandation supprime l'obligation de validation pour lecture car des processus d'assurance qualité peuvent remplacer cette validation à toutes les étapes d'un processus informatique.

Dans les DTD, une des erreurs de jeunesse de SGML était de définir, en même temps que le modèle, les variations possibles du format de codage des documents eux-mêmes : options de minimisation des balises et, si nécessaire, options de redéfinition des balises elles-mêmes. Du fait de la standardisation d'Unicode XML et des nouvelles capacités des ordinateurs actuels, toutes ces options ont été supprimées, pour arriver à l'obligation de "document bien formé", notion qui était, de fait, la sortie standard de tous les éditeurs SGML.


Enfin, d'autres mécanismes peu utilisés existaient, comme ceux d'inclusion et d'exclusion. Il n'ont pas été jugés utiles dans XML, car apportant de la complexité aux traitements sans pour autant faciliter la vie du rédacteur.

Recommandations(s)

■ ■ *Traitement de l'information - Systèmes bureautiques - Langage normalisé de balisage généralisé*

Standard ISO, version 8879:1986, du 17-08-1996

Document sur [http://www.iso.ch/iso/fr/CatalogueDetailPage.CatalogueDetail?
CSNUMBER=16387](http://www.iso.ch/iso/fr/CatalogueDetailPage.CatalogueDetail?CSNUMBER=16387)

 *Information processing - Text and office systems - Standard Generalized Markup Language (SGML)*

Standard ISO, version 8879:1986, du 17-08-1996

Document sur [http://www.iso.ch/iso/en/CatalogueDetailPage.CatalogueDetail?
CSNUMBER=16387](http://www.iso.ch/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=16387)

TIRÈME SARL

XML, Langage de balisage extensible

Rédaction : Pierre Attar

Recommandation(s) liée(s) : [Canonical](#) - [XInclude](#) - [Infoset](#) - [Fragment Interchange](#) - [XML Base](#) - [DTD](#) - [XML Language](#)

XML est un **langage de codage de données** dont l'objectif est, dans un échange entre systèmes informatiques, de transférer, en même temps, des données et leurs structures.

Permettant de coder n'importe quel type de donnée, depuis l'échange EDI jusqu'aux documents les plus complexes en passant par les échanges de données inter-applications, son potentiel est de devenir le standard universel et multilingue d'échange d'informations. Appliqué aux documents textuels, il permet d'identifier, de façon logique, la structure et l'organisation de l'information textuelle.

Objectifs

XML est un format textuel très flexible dérivé de SGML. Initialement conçu pour relever les défis de l'édition électronique de grande puissance, XML joue également un rôle de plus en plus important dans l'échange d'une grande variété de données, que ce soit sur le Web ou pour n'importe quel échange inter-applicatif.

XML permettra, comme le souligne le W3C :

- l'édition électronique internationalisée, de façon indépendante des logiciels et des systèmes ;
- aux industries de définir des protocoles, indépendants des logiciels et des systèmes, pour l'échange des données (particulièrement les données du commerce électronique) ;
- de fournir de l'information aux agents utilisateurs sous une forme qui permette un traitement automatique après réception, par exemple pour toutes les applications de téléphonie mobile ;
- de faciliter le développement logiciel dès lors qu'il s'agit de manipuler l'information spécialisée et répartie ;
- de faciliter les traitements de données avec des logiciels peu coûteux ; à ce titre, l'avenir d'XML et d'un certain nombre de ses recommandations associées (XSLT, XPath, Infoset, etc.) est de devenir partie intégrante des couches hautes des systèmes d'exploitation ;
- aux utilisateurs du Web d'afficher l'information reçue avec la feuille de styles qu'ils souhaitent ;
- de faciliter la fourniture de Métadonnées (données descriptives de documents) qui aide à trouver de l'information.

Principes

Un document XML se compose, d'une part, de texte, et, d'autre part, d'informations de structure. Les informations de structure servent le plus souvent à délimiter du texte, pour en identifier la sémantique. Ainsi, `<métier>Consultant</métier>` permet de dire que la chaîne de caractères "Consultant" doit être comprise comme étant une définition de métier. Il est possible de délimiter des chaînes de caractères ; il est aussi possible de délimiter tout ensemble d'informations mélangeant texte et structure. Par exemple, dans le document suivant :

```
<métier>
  <nom>
    Consultant
  </nom>
  <descr>
    Le consultant travaille pour des consultés qui ...
  </descr>
</métier>
```

La notion de métier introduit, d'une part, un nom de métier et, d'autre part, une description de métier. C'est l'appartenance hiérarchique qui définit que tout cela parle bien du même métier : elle permet de spécialiser des description : un nom de métier et une description de métier.

Pour compléter cette description, il est parfois nécessaire de *valuer* la signification d'un objet. Par exemple, `<auteur affiliation="rennes2">Jean Dupont</auteur>`, permet d'identifier un chaîne de caractères comme étant un auteur et, en plus, de décrire cet auteur, par *valuation*, comme appartenant à l'Université de Rennes II.

Les balises (`<maBalise> ... </maBalise>`) délimitent des objets typés ; les attributs (`<maBalise type="standard">`) définissent des *valuations* d'objets. Un document XML est alors un arbre d'objets typés et *valués*.

Pour finir, un document XML doit définir le jeu de caractères qu'il utilise, ainsi que la version de la recommandation XML. Du coup, l'exemple précédent s'écrira :

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

Ce document est parfaitement décrit, il représente, au sens de la recommandation, un document "bien formé" (*wellformed*). Si nécessaire, un modèle documentaire peut lui être adjoint (voir DTD). Celui-ci définira les contraintes associées à ce document : le document devra alors être "valide", au regard de ce modèle. Définis de façon électronique, les applicatifs de type *parsers* seront capables de valider, de façon automatisée, la conformité d'un document à sa classe, à son modèle. L'avantage ? Il sera possible d'appliquer des processus automatiques sur une classe de documents (les algorithmes étant écrits au regard de la classe et non pas au regard de chaque instance de document).

Les différents aspects de la recommandation XML

La recommandation XML s'intéresse à des notions fort différentes :

- un langage de codage, basé sur Unicode XML, de documents, avec des éléments, des attributs, un jeu de caractères, etc. ;
- un langage de définition de modèles documentaires, avec les DTD ;
- un langage d'expression d'inclusions de fichiers, avec les `internal` et `external subset` des DTD ;

l'expression d'informations applicatives, comme les deux attributs `xml:lang` et `xml:space`, dont le rôle est, respectivement, d'indiquer la langue de rédaction d'un fragment de contenu et la façon de traiter les caractères d'espacement d'un contenu ;
un langage de prise en compte extrêmement limité des espaces de noms.

Le fait de mettre tous ces aspects dans une même recommandation a un aspect politique important, car cela oblige à prendre *tout* en compte sans différenciation. D'un point de vue technique, cela peut parfois poser des problèmes de compréhension : par exemple, pourquoi prendre en compte seulement les notions de langue et pas celles d'URI ? Cela peut aussi poser des problèmes d'architecture d'application, dès lors que, par exemple, un [Schema](#) permet d'exprimer la même chose, et davantage qu'une DTD... sauf les notions d'inclusions de données spécifiées de façon indépendante des documents eux-mêmes, au travers des entités générales.

Recommandations(s)

- ■ *Le langage de balisage extensible (XML) 1.1*
Recommandation, version 1.1, du 04-02-2004
Document sur <http://www.yoyodesign.org/doc/w3c/xml11/>
- 🇺🇸 *Extensible Markup Language (XML) 1.1 (Second Edition)*
Recommandation, version 1.1, du 16-08-2006
Document sur <http://www.w3.org/TR/xml11>



Unicode XML, Unicode in XML and other Markup Languages

Unicode XML propose un jeu de caractères "universel". Cet objectif est extrêmement important sur Internet, où des textes dans toutes les langues, voir multilingues, cohabitent.

Actuellement, et dans sa troisième version, Unicode couvre la plupart des séquences de caractères utilisées dans le monde. Il contient également les caractères spéciaux pour l'interopérabilité avec des jeux de caractères plus anciens, ainsi que les caractères de commande usuels.

XML utilise Unicode XML, selon ses propres recommandations (voir). L'avantage est alors de détenir un véritable langage universel, au moment où les échanges de données sont de plus en plus universel, surtout sur l'Internet.

Si, grâce à Unicode XML, les caractères échangés sont non ambigus, cela ne veut pas pour autant dire que l'affichage de ces caractères est toujours possible... On se rappellera qu'il existe une différence fondamentale entre la *définition* d'un caractère et sa *représentation* (souvent appelée *police* de caractère ou plus précisément *glyphe*).

Objectifs

Information descriptive non finalisée ; n'hésitez pas à nous contacter pour rédiger et/ou maintenir cette information à jour.

Recommandations(s)

Unicode

Recommandation, version 4.1.0, du 03-2005
Document sur <http://www.unicode.org/standard/versions/enumeratedversions.html#Latest>

Unicode dans XML et les autres langages de balisage. Rapport technique Unicode n° 20

Note, version 20030613, du 13-06-2003
Document sur <http://www.yoyodesign.org/doc/w3c/unicode-xml-20030613/>

Unicode in XML and other Markup Languages - Unicode Technical Report 20

Note, version 20030613, du 13-06-2003
Document sur <http://www.w3.org/TR/unicode-xml/>

Un modèle de caractères pour le Web 1.0 : les principes de base

Recommandation, version 1.0, du 15-02-2005
Document sur <http://www.yoyodesign.org/doc/w3c/charmod-20050215/index.html#>

Character Model for the World Wide Web 1.0

Recommandation, version 1.0, du 15-02-2005
Document sur <http://www.w3.org/TR/charmod/>

NameSpace, Espace de nom

Rédaction : Pierre Attar

Recommandation(s) liée(s) : [DTD - Schema](#)

Les espaces de noms d'[XML](#) permettent de qualifier de manière unique des éléments et des attributs. On sait alors à quel domaine de définition se rapporte un objet et comment il doit être interprété, selon sa spécification.

Le seul fait d'utiliser l'espace de noms `xml` pour un attribut `lang` (voir [XML Language](#)) permet, par exemple, de définir la langue de codage d'un document (ou d'une portion de document), ainsi que la façon dont l'attribut se propage sur un sous-arbre.

Par ailleurs, différencier des espaces de noms permet de faire coopérer, dans un même document, des objets ayant le même nom, mais une signification différente, souvent liée à un modèle de contenu différent. Par exemple, il devient possible, avec les espaces de noms, de faire coopérer, dans un même document, des [OASIS Tables](#) et [HTML](#) : les deux spécifications utilisent toutes les deux un élément `table` de plus haut niveau qui sera, par exemple, codé `oasis:table` et `html:table`.

Cette spécification est une avancée importante car, à partir du moment où beaucoup de formats s'expriment selon [XML](#) (voir [SVG](#), par exemple), les risques de "collision de noms" deviennent plus importants et cette spécification prend toute son importance.

Le seul regret, dans l'attente des [Schema](#) et de leur implémentation dans les outils, est que cette spécification ne vienne qu'au-dessus de [XML](#) dans le cadre d'une architecture en couches et que, donc, elle ne soit pas prise en compte dans les [DTD](#).

Objectifs

L'objectif des espaces de noms est de donner aux éléments et aux attributs des noms uniques, sur Internet. Cette identification permet d'avoir une connaissance mutuelle d'objets, entre plusieurs [DTD](#), sans pour autant qu'il y ait contradiction entre l'interprétation des éléments et, surtout, leurs contenus.

L'utilité des espaces de noms est de pouvoir avancer sur les notions de documents composites, quelle que soit la nature des contenus des documents. Par exemple, supposons un document qui contienne ses propres paragraphes (nommés `P`) et qui doive par ailleurs comporter des fragments de contenus [HTML](#). [HTML](#) définissant ses propres paragraphes (aussi nommés `P`), il peut y avoir contradiction entre les deux définitions. La

notion d'espace de noms lève cette contradiction et permet aux deux éléments de coopérer. L'un, par exemple celui issu du modèle [HTML](#), sera préfixé par son espace de noms : `html:P`.

Cette solution des espaces de noms est très intéressante, dès lors que l'on s'intéresse aux processus de traitements d'un document [XML](#). En effet, grâce aux espaces de noms, il est possible d'avoir des traitements factorisés et dans l'exemple, de faire confiance à un système de consultation [HTML](#), afin d'interpréter directement les objets de l'espace de noms [HTML](#). Il devient alors possible de redéfinir des contextes de traitement, sans pour autant être obligé de modifier tous les attributs des objets de ce contexte.

Différents exemples, dans différents domaines, permettent de mieux comprendre l'utilité des espaces de noms :

- pour le [W3C](#), les espaces de noms servent à faire coopérer ses différentes recommandations ; par exemple, à pouvoir avoir une feuille de styles utilisant en même temps [XSLT](#), [HTML](#) et [XSL](#), sans risques de collision de noms ;

- pour le monde documentaire, si on utilise des modèles de fragments complexes, comme ceux définis pour les graphiques vectoriels ([SVG](#)), pour les notations typographiques mathématiques ([MathML](#)) ou, encore, pour les tableaux ([OASIS Tables](#)), la notion de factorisation de traitement prend toute sa valeur, car afficher un graphique, un tableau ou une équation n'est pas une opération triviale. L'intérêt est donc de pouvoir partager une variété vaste d'outils ;

- cette factorisation de compréhension et, donc, de traitements s'applique ensuite à tous les domaines. Par exemple, dans les applications EDI, il est intéressant de pouvoir construire une base de données qui contienne les éléments nécessaires à un EDI particulier, ces éléments coopérant avec d'autres, plus localisés à des *process* internes d'entreprise. Il en va de même, et de façon plus générale, dans le monde du commerce électronique ;

- enfin, tous les *portails* s'occupant de syndication de données sont intéressés par les espaces de noms. Cela leur permet, d'une part, de savoir d'où viennent les données qu'ils présentent, et, d'autre part, de se reposer sur le site source pour connaître les moyens de les traiter (par exemple, pour les afficher). Ainsi, un site commercial sera capable d'avoir des documents mixtes contenant une information *prix*, selon le modèle du portail, et une information *commerciale*, directement issue du modèle de l'entreprise source, utilisant les outils de cette entreprise pour l'affichage.

Principes

On peut définir un espace de noms sur n'importe quel élément d'un document : son champs d'utilisation est alors celui du sous-arbre délimité par l'élément. Pour ce faire, il suffit d'ajouter à l'élément, un attribut identifiant l'espace.

Par exemple, `<monélément xmlns:mutuxml="http://www.mutu-xml.org/DTDProjet" version="1.0">` définit un espace de noms dénommé `http://www.mutu-xml.org/DTDProjet`; cette déclaration définit aussi que tous les éléments de cet espace de noms seront préfixés par `mutuxml`. Ainsi, il sera possible d'écrire :

```
<monélément xmlns:mutuxml="http://www.mutu-xml.org/DTDProjet/1.0">
  <p>un paragraphe ... de l'espace de noms par défaut</p>
  <mutuxml:p>un paragraphe ... de l'espace de noms mutuxml</p>
</monélément>
```

... où deux éléments `p` cohabitent, appartenant à deux espaces de noms distincts.

Dans cet exemple, la seule donnée identifiant l'espace de noms est l'URI ! Si, par exemple, un autre document définit un espace de noms avec : `xmlns:mualisationxml="http://www.mutu-xml.org/DTDProjet/1.0"`, le

paragraphe `<mutuxml:p>` de l'exemple ci-devant est bien le même type d'objet que `<mutualisationxml:p>`, de cette nouvelle définition. Le préfixe est donc peu signifiant, sauf dans le champ de codage syntaxique défini par l'élément sur lequel l'espace est défini.

Pour finir, il est fondamental de noter qu'un espace de noms n'a qu'une valeur lexicale : il sert à "désambigüiser" des noms d'objets (éléments ou attributs). En effet, la recommandation n'oblige pas à lier des mots d'un espace de noms à un modèle, qu'il soit sous forme de DTD ou de Schema. Ceci est en outre intéressant, car cela peut s'appliquer aussi aux documents "bien formés". Ceci est gênant dès lors qu'une application utilise des modèles documentaires sous forme de DTD, car il existe peu de choses définies quant à l'intégration de modèles liés à un espace de noms.

Recommandations(s)

■ ■ *Les espaces de nommage dans XML 1.1*

Recommandation, version 20040204, du 04-02-2004

Document sur <http://www.yoyodesign.org/doc/w3c/xml-names11/>

🇺🇸 *Namespaces in XML*

Recommandation, version 1.1 (Third Edition), du 08-12-2009

Document sur <http://www.w3.org/TR/xml-names/>

🇺🇸 *Namespaces in XML 1.1 Requirements*

Projet en cours, version 1.1, du 03-04-2002

Document sur <http://www.w3.org/TR/xml-names11-req/>

🇺🇸 *Uniform Resource Names (URN) Namespace Definition Mechanisms*

Projet en cours, version 200202, du 02-2002

Document sur <http://www.ietf.org/rfc/rfc3406.txt>

TIRÈME SARL

Schema, Schéma

Rédaction : Pierre Attar

Recommandation(s) liée(s) : [DSDL](#) - [DTD](#) - [Infoset](#) - [RELAX NG](#) - [XML](#) - [XML Base](#) - [XMI](#) - [XPath](#) - [WSDL](#)

Partant du principe que les [DTD](#) issues de la Recommandation [XML](#) ne permettaient pas de contraindre suffisamment les données et les structures, partant également du principe qu'il n'y avait aucune raison pour que les modèles ne soient pas codés sous forme [XML](#), [Schema](#) propose une méthode de réalisation de modèles, alternative aux [DTD](#).

Cette spécification, plus moderne de conception que les [DTD](#), permet d'écrire des modèles de façon plus efficace. De fait, et dans son utilisation, cette spécification supplante maintenant l'utilisation des [DTD](#) et, hors environnement contractuel spécifique, il reste peu d'endroits où de nouveaux projets utilisent des [DTD](#).

L'avenir de cette spécification ? Trouver une intégration harmonieuse avec la spécification [DSDL \(RELAX NG\)](#) qui a un champ de recouvrement important (voir par exemple la dernière version de [DocBook](#) qui préfère se reposer sur [RELAX NG](#) plutôt que d'utiliser les [DTD](#) ou les [Schema](#).

Objectifs

L'objectif des [Schema](#) est de définir des contraintes sur des classes de documents conformes à un même modèle. À la différence des [DTD](#), qui ne définissaient *que* les relations entre les différents composants d'un document, les [Schema](#) peuvent typer les données et, par là-même, en documenter leur sens et, donc, leur utilisation.

L'objectif n'est pas d'avoir un langage extensible permettant d'exprimer n'importe quelle contrainte : seules, les contraintes consensuelles sont exprimables. Celles-ci doivent pouvoir être validées par un *parser* et être mises en jeu pour assister à la création d'information, surtout dans les éditeurs.

Enfin, les [Schema](#) ne s'intéressent pas à la validation d'un document [XML](#) (d'un fichier) en tant que tel. Un outil validant se basera sur la représentation en arbre d'objets typés et *valués*, désérialisée selon le standard [Infoset](#). Cette volonté permet de rendre le processus de validation indépendant d'une syntaxe de codage quelconque d'un document.

Principes

Les spécifications sur les [Schema](#) sont divisées en deux parties : une sur les *structures* et une sur les *types de données*. Pour mieux comprendre ces deux recommandations, une introduction est aussi proposée.

D'un point de vue structurel, un **Schema** se définit comme étant l'assemblage, sous forme d'arbre **XML**, d'un ensemble de composants. Il en existe 12, dont les principaux sont :

composants de définition de types, les simples ou les complexes :

ils permettent de typer des éléments pour leur affecter un modèle de contenu. Il en va de même pour les attributs. Ainsi, une *adresse de livraison* et une *adresse de facturation* relèveront toutes deux d'un même type *adresse*.

C'est en utilisant les types simples que seront définis, dans la partie "*datatypes*" (), des types de base comme les chaînes de caractères, les nombres, les **URI** ou encore toutes sortes de dates.

Une définition de "type" peut être réalisée en *restriction* ou en *extension* d'une définition existante. Par exemple, le type *adresse* précédent peut être restreint à des adresses françaises, en jouant sur la définition de la valeur du code pays. Une limite du mécanisme d'extension est que l'on ne peut rajouter des éléments *qu'à la fin* d'un type pré-existant et que l'on ne peut pas les modifier.

D'un point de vue programmation, les notions de types de données permettent de se reposer sur ces types de données pour effectuer des traitements partagés à différents éléments du même type. Ceci simplifie beaucoup les méthodes de programmation, tout en nécessitant d'avoir toujours accès au modèle, en même temps que l'on effectue les traitements.

composants de déclaration : éléments, attributs et notations

Ces composants permettent de déclarer, comme avec les **DTD**, des éléments, avec leurs modèles de contenus, et ainsi que des attributs. Pour ce faire, et dans le but de factorisation précédemment expliqué, les déclarations peuvent se reposer sur des composants de définition de type.

composants groupes : pour la factorisation d'informations de modèles de contenus ou d'attributs.

Ces composants permettent de définir des modèles qui seront souvent réutilisés, soit pour définir des composants de déclaration, soit pour définir des composants de définition de types. Dans les **DTD** traditionnelles, ces groupes étaient formalisés par des entités paramètres, qui permettaient, par exemple, de définir tous les composants *blocks* (paragraphes, listes, remarques, etc.) nécessaires lors de la définition de modèles de contenus.

D'un point de vue typage de données, la spécification propose un ensemble important de types et une façon d'étendre ces types, pour une application donnée, à des choses plus complexes.

Finalement, un **Schema** est un modèle d'information défini sous forme électronique (comme l'était une **DTD**), mais avec des outils de modélisation plus puissants. Dans la sphère **XML**, le **Schema** pourra être utilisé comme outil de validation interactif de données, lors de la création de celles-ci, dans un éditeur, voire, plus tard, dans des **XForms**, sur Internet.

Des limites ? En lisant les listes de discussion, on peut trouver beaucoup de limitations exprimées sur ces possibilités de validation. Par exemple, d'aucuns parlent du fait de ne pas pouvoir valider un modèle de contenu en fonction d'une valeur d'attribut d'un élément englobant. C'est toute l'ambiguïté de l'utilisation des **Schema** qui est alors posée : est-ce un langage de programmation permettant d'exprimer des contraintes de validation ou est-ce un langage de définition de modèles ?

En avançant dans le sens de l'utilisation de modèles pour des activités d'ingénierie, et grâce aux nouvelles possibilités de typage, on peut avoir plusieurs **DTD** et **Schema**, fonctionnant sur une même classe de documents et, par des définitions de typage, déclencher dessus des processus applicatifs *ad hoc*. C'est ce qu'avaient fort bien compris les concepteurs de l'ICADD [sur internet : www.oasis-open.org/cover/gen-apps.html#icadd], pour rendre accessibles les documents aux non-voyants. Dans leurs spécifications [sur internet : www.oasis-open.org/]

cover/gen-apps.html#cadd], ils expliquent comment, en jouant seulement sur les attributs d'une DTD, on peut ajouter automatiquement de l'information aux documents, au moment de leur *parsing*, pour le plus grand profit des non-voyants qui pourront parcourir n'importe quel type de documents en se basant sur ces repères.






Informations connexes

Liste des types définis dans les schémas.


Catégorie	Type	Signification	Modèles de contenus	Attributs
Chaînes de caractères	string	Chaîne	O	O
	Name	Nom XML	O	O
	QName	Nom XML qualifié	O	O
	NCName	Nom XML non qualifié	O	O
Nombres	decimal	Décimal	O	O
	boolean	Valeur booléenne	O	O
	float	16bit floating point number	O	O
	double	32bit floating point number	O	O
	integer	Infinite accuracy integer	O	O
	nonPositiveInteger	Infinite accuracy integer less than 0	O	O
	negativeInteger	Infinite accuracy integer less than 0	O	O
	long	64bit integer	O	O
	int	32bit integer	O	O
	short	16bit integer	O	O
	byte	8bit integer	O	O
	nonNegativeInteger	Infinite accuracy integer more than 0	O	O
	positiveInteger	Infinite accuracy integer more than 1	O	O
	unsignedLong	Unsigned 64bit integer	O	O
unsignedInt	Unsigned 32bit integer	O	O	

Catégorie	Type	Signification	Modèles de contenus	Attributs
	unsignedShort	Unsigned 16bit integer	O	O
	unsignedByte	Unsigned 8bit integer	O	O
Temps	timeInstant	Date+Time	O	O
	timeDuration	Progress time	O	O
	recurringInstant		O	O
	date	Date	O	O
	time	Time	O	O
XML1.0 compatibilité	ID	XML1.0 ID	-	O
	IDREF	XML1.0 IDREF	-	O
	ENTITY	XML1.0 ENTITY	-	O
	NOTATION	XML1.0 NOTATION	-	O
	IDREFS	XML1.0 IDREFS	-	O
	ENTITIES	XML 1.0 ENTITIES	-	O
	NMTOKEN	XML 1.0 NMTOKEN	-	O
	NMTOKENS	XML 1.0 NMTOKENS	-	O
Autres.	binary	Binary	O	O
	uriReference	URI	O	O
	language	Language	O	O

Recommandations(s)


-  **XML Schema tome 0 : Introduction**
 Recommandation, version 20010502, du 02-05-2001
 Document sur <http://xmlfr.org/w3c/TR/xmlschema-0/>
-  **XML Schema tome 1 : Structures**
 Recommandation, version 20010502, du 02-05-2001
 Document sur <http://xmlfr.org/w3c/TR/xmlschema-1/>
-  **XML Schema Part 0: Primer**
 Recommandation, version Second Edition, du 28-10-2004
 Document sur <http://www.w3.org/TR/xmlschema-0/>
-  **XML Schema Part 1: Structures**
 Recommandation, version Second Edition, du 28-10-2004
 Document sur <http://www.w3.org/TR/xmlschema-1/>
-  **XML Schema Part 2: Datatypes**

Recommandation, version Second Edition, du 28-10-2004
Document sur <http://www.w3.org/TR/xmlschema-2/>

 *Associating Schemas with XML documents 1.0 (First Edition)*


Projet en cours, version 1.0, du 15-04-2010

Document sur <http://www.w3.org/TR/xml-model/>

 *Schema-Related Markup in Documents Being Validated*

Recommandation, version 20010502, du 02-05-2001

Document sur http://www.w3.org/TR/xmlschema-1/#Instance_Document_Constructions

 *XML Schema: Formal Description*


Projet en cours, version 20010320, du 20-03-2001

Document sur <http://www.w3.org/TR/xmlschema-formal/>

 *XML Schema: Component Designators*


Projet en cours, version 20050329, du 29-03-2005

Document sur <http://www.w3.org/TR/xmlschema-ref/>

 *Requirements for XML Schema 1.1*

Projet en cours, version 1.1, du 20030121

Document sur <http://www.w3.org/TR/2003/WD-xmlschema-11-req-20030121/>

 *Semantic Annotations for WSDL and XML Schema*

Recommandation, version 20070828, du 28-08-2007

Document sur <http://www.w3.org/TR/sawsdl/>

DTD, Définition de type de document

Rédaction : Pierre Attar

Recommandation(s) liée(s) : [DSDL](#) - [RELAX NG](#) - [Schema](#) - [XML](#) - [XMI](#)

La notion de [DTD](#) est partie intégrante de la spécification [XML](#), héritière de [SGML](#), mais délestée de beaucoup de fonctionnalités aujourd'hui superflues.

Pour des *parsers* "validants", la notion de [DTD](#) dans [XML](#) permet de définir un modèle de données qui servira ensuite à valider, de façon électronique, toutes les instances supposées être conformes au modèle. Pour des *parsers* "non validants", la notion de [DTD](#) sert simplement à fournir un réservoir permettant de décrire où sont des fichiers qui doivent être inclus dans les documents [XML](#) au travers d'entités générales.

Le modèle lui-même permet de spécifier une hiérarchisation d'objets typés et *valués* selon des attributs. Il permet encore, au travers des entités paramètres, de modulariser l'écriture des [DTD](#). Ce modèle est largement implémenté par tous les outils de création et de modification de documents issus du monde [SGML](#).

Objectifs

L'objectif d'une [DTD](#) est de pouvoir définir un modèle de données formel, sous forme électronique, afin que les outils puissent valider, de façon automatisée, la conformité d'une classe de documents [XML](#) à ce modèle.

Cette activité est nécessaire pour tous les processus de traitement de l'information [XML](#), qui doivent pouvoir présupposer de ce qui est contenu dans un document. Elle est aussi extrêmement utile pour les processus d'édition interactive de documents [XML](#) qui, s'adaptant à une [DTD](#), pourront aider l'utilisateur dans ses activités de création et de modification de structure et dans ses activités de validation.

Par ailleurs, l'objectif d'une [DTD](#) est de pouvoir définir tout ce qui est nécessaire comme ressources à tous les documents d'une classe donnée. Si un document [XML](#) doit inclure des fragments externes, ceux-ci seront définis et identifiés de façon formelle au sein de la [DTD](#), afin de pouvoir ensuite être appelés au sein des documents (mécanisme des entités *parsées* générales).

Si le modèle devait recenser tous les fragments appelables dans un document, le fichier issu de ce modèle deviendrait extrêmement imposant et surtout très difficile à maintenir. Du coup, les concepteurs de la recommandation autorisent l'utilisation de **DTD**, complétées de déclarations n'affectant qu'un seul document : on parle alors des entités *parsées* générales de l'*Internal Entity* .

Principes

Modèles de documents

Une **DTD** permet de définir une organisation d'éléments entre eux et des typages de ces éléments via des notions d'attribut (de *valuation*).

Plus précisément, une **DTD** définit un modèle d'organisation hiérarchique de documents **XML** en utilisant :

- des *éléments*, organisés hiérarchiquement. Pour ce faire, un modèle de contenu est défini pour chaque élément. Le modèle de contenu peut faire appel aux notions d'organisation d'éléments entre eux (séquence, choix ou agrégat), en utilisant des indicateurs d'occurrence (optionabilité, répétabilité). Il peut se réduire aussi à de simples caractères. Lorsqu'il définit un regroupement de caractères et d'éléments, on parle alors de contenus mixtes (*mixed contents*). Dans ce dernier cas, des règles très strictes sont définies ;

- des *attributs*, qui permettent de valuer un élément. Les attributs sont typés ou énumérés. Ils peuvent avoir une valeur fixe par défaut ou laissée au libre choix des outils de traitement ;

- d'autres informations qui peuvent être ajoutées aux éléments et aux attributs, telles les *notations*, les *entités paramètres ou générales*, etc.

La syntaxe utilisée pour écrire des **DTD** est une syntaxe *ad hoc*. Par exemple, déclarer un élément `article` contenant une en-tête et un corps s'écrira :

```
<!ELEMENT article (tete, corps) >
```

Cette syntaxe est concise et simple, mais présente le grand désavantage de ne pas être celle utilisée pour écrire des documents **XML** et, donc, de ne pas pouvoir être analysée par des outils **XML** standard. C'est une des raisons qui amène à la définition des nouveaux modèles **Schema**.

DTD et entités

Les DTD permettent aussi de définir des entités qui implémentent le mécanisme d'inclusion cher à tous les langages de programmation. *Générale et parsée*, l'entité définit des fragments **XML** que le *parser* doit inclure lors de la validation d'un document (que ce soit pour vérifier s'il est "bien formé" ou s'il est valide au regard de son modèle) ; le contenu du fragment appelé doit lui-même être un arbre bien formé. Si elle est *paramètre*, l'entité définit des fragments de **DTD** à intégrer lors du *parsing* de la **DTD** elle-même.

L'entité peut être externe au document. Par exemple :

```
<!ENTITY fichier "-//MUTU-XML//Un fichier externe//FR" "http://www.monsite.fr/fragment.xml">
```

définit une entité générale (un fragment **XML** bien formé), qui se nomme `-//MUTU-XML//Un fichier externe//FR` et qui se trouve à l'adresse `http://www.monsite.fr/fragment.xml`.


L'entité peut être interne. Par exemple :

```
<!ENTITY xml "eXtended Markup Language">
```

définit un fragment **XML** directement dans la déclaration interne liée au document.

Ce mécanisme d'inclusion est extrêmement utile pour toutes les opérations de modularisation et de réutilisation. Il souffre cependant d'une lacune, qui est l'obligation de définir dans le document, d'une part, un nom logique et, d'autre part, une localisation physique. Cette lacune existait aussi dans les premiers temps de **SGML** et était fort contraignante, dès lors que des documents étaient échangés et que les organisations de fichiers n'étaient pas les mêmes chez l'émetteur et le récepteur d'un document. (Par exemple, un développeur de site Web utilisera souvent des adresses basées sur ses propres disques locaux, voire basées sur son site de test. Lorsqu'il publiera l'information, les adresses seront alors basées sur le site réel.)

Recommandations(s)

 *Extensible Markup Language (XML) 1.1 (Second Edition)*
Recommandation, version 1.1, du 16-08-2006
Document sur <http://www.w3.org/TR/xml11#sec-prolog-dtd>

TIRÈME SARL

DSDL, Document Schema Definition Languages

Rédaction : Pierre Attar

Recommandation(s) liée(s) : [DTD](#) - [RELAX NG](#) - [Schema](#) - [XMI](#)


« The objective of developing ISO/IEC 19757 Document Schema Definition Languages (DSDL) is to create a framework within which multiple validation tasks of different types can be applied to an XML document in order to achieve more complete validation results than just the application of a single technology. »

DSDL.ORG [sur internet : www.dSDL.org]

Objectifs

Information descriptive non finalisée ; n'hésitez pas à nous contacter pour rédiger et/ou maintenir cette information à jour.


Recommandations(s)

 *Document Schema Definition Languages (DSDL) — Part 1: Overview*

Recommandation, version ISO/IEC CD 19757-1, du 16-11-2004

Document sur [http://www.iso.ch/iso/en/CatalogueDetailPage.CatalogueDetail?](http://www.iso.ch/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=37606&scopelist=ALL)


CSNUMBER=37606&scopelist=ALL

 *Document Schema Definition Languages (DSDL) — Part 2: Regular-grammar-based validation -- RELAX NG*

Recommandation, version ISO/IEC 19757-2:2003, du 28-11-2003

Document sur [http://www.iso.ch/iso/en/CatalogueDetailPage.CatalogueDetail?](http://www.iso.ch/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=37605&scopelist=ALL)


CSNUMBER=37605&scopelist=ALL

 *Document Schema Definition Languages (DSDL) — Part 3: Rule-based validation -- Schematron*

Recommandation, version ISO/IEC CD 19757-3, du 01-12-2004

Document sur [http://www.iso.ch/iso/en/CatalogueDetailPage.CatalogueDetail?](http://www.iso.ch/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=40833&scopelist=ALL)










CSNUMBER=40833&scopelist=ALL

 *Document Schema Definition Languages (DSDL) — Part 4: Namespace-based Validation Dispatching Language -- NVDL*

Recommandation, version ISO/IEC FCD 19757-4, du 10-01-2005

Document sur [http://www.iso.ch/iso/en/CatalogueDetailPage.CatalogueDetail?](http://www.iso.ch/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=38615&scopelist=ALL)

CSNUMBER=38615&scopelist=ALL

-  *Document Schema Definition Languages (DSDL) — Part 5: Extensible Datatypes*
Recommandation, version ISO/IEC WD 19757-5, du 2008-12-17
Document sur [http://www.iso.ch/iso/en/CatalogueDetailPage.CatalogueDetail?
CSNUMBER=41006&scopelist=ALL](http://www.iso.ch/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=41006&scopelist=ALL)
-  *Document Schema Definition Languages (DSDL) — Part 6: Path-based integrity constraints*
Recommandation, version ISO/IEC WD 19757-6, du 20-04-2004
Document sur [http://www.iso.ch/iso/en/CatalogueDetailPage.CatalogueDetail?
CSNUMBER=41007&scopelist=ALL](http://www.iso.ch/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=41007&scopelist=ALL)
-  *Document Schema Definition Languages (DSDL) — Part 7: Character repertoire validation*
Recommandation, version ISO/IEC WD 19757-7, du 20-04-2004
Document sur [http://www.iso.ch/iso/en/CatalogueDetailPage.CatalogueDetail?
CSNUMBER=41008&scopelist=ALL](http://www.iso.ch/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=41008&scopelist=ALL)
-  *Document Schema Definition Languages (DSDL) — Part 8: Declarative document manipulation*
Recommandation, version ISO/IEC WD 19757-8, du 20-04-2004
Document sur [http://www.iso.ch/iso/en/CatalogueDetailPage.CatalogueDetail?
CSNUMBER=40836&scopelist=ALL](http://www.iso.ch/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=40836&scopelist=ALL)
-  *Document Schema Definition Languages (DSDL) — Part 9: Datatype- and namespace-aware DTDs*
Recommandation, version ISO/IEC WD 19757-9, du 20-04-2004
Document sur [http://www.iso.ch/iso/en/CatalogueDetailPage.CatalogueDetail?
CSNUMBER=41009&scopelist=ALL](http://www.iso.ch/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=41009&scopelist=ALL)
-  *Document Schema Definition Languages (DSDL) — Part 10: Validation Framework*
Recommandation, version ISO/IEC WD 19757-10, du 20-04-2004
Document sur [http://www.iso.ch/iso/en/CatalogueDetailPage.CatalogueDetail?
CSNUMBER=41011&scopelist=ALL](http://www.iso.ch/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=41011&scopelist=ALL)
-  *Compact Syntax*
Recommandation, version ISO/IEC 19757-2:2003/FPDAmd 1, du 18-01-2005
Document sur [http://www.iso.ch/iso/en/CatalogueDetailPage.CatalogueDetail?
CSNUMBER=40774&scopelist=ALL](http://www.iso.ch/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=40774&scopelist=ALL)
-  *Document Schema Definition Languages (DSDL) — DSSSL library for complex compositions*
Recommandation, version ISO/IEC TR 19758:2003, du 31-03-2003
Document sur [http://www.iso.ch/iso/en/CatalogueDetailPage.CatalogueDetail?
CSNUMBER=33896&scopelist=ALL](http://www.iso.ch/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=33896&scopelist=ALL)
-  *RELAX NG Specification*
Recommandation, version 20011203, du 03-12-2001
Document sur <http://www.oasis-open.org/committees/relax-ng/spec-20011203.html>

TIRÈME SARL

RELAX NG, Regular Language description for XML NG

Recommandation(s) liée(s) : DSDL - DTD - Schema - XMI

RELAX NG est maintenant partie intégrante de DSDL : deuxième section. Voir cette entrée pour toutes les informations sur RELAX NG.

Objectifs

Voir [DSDL](#).

Recommandations(s)

 *RELAX NG Specification*

Recommandation, version 20011203, du 03-12-2001

Document sur <http://www.oasis-open.org/committees/relax-ng/spec-20011203.html>

TIRÈME SARL

Canonical, XML Canonical

Rédaction : Pierre Attar

Cette spécification décrit ce que doit faire un "canoniseur" XML pour produire une représentation physique (la forme canonique) d'un document XML, qui ne change pas quelles que soient les variations syntaxiques de l'entrée XML.

Avec cette spécification, si un document XML est changé par une application, mais si sa forme canonique XML n'a pas changé, le document changé et le document initial peuvent être considérés comme équivalents par les applications les utilisant.

Cette spécification est mineure dans l'ensemble de celles issues du W3C. Elle sera certainement utilisée dans des ingénieries logicielles très spécifiques, dès lors qu'il sera nécessaire de s'intéresser à des versions de documents.

Objectifs

L'objectif de cette spécification est de pouvoir s'appuyer sur une forme XML canonique pour définir l'équivalence de deux documents XML, de façon indépendante des variations syntaxiques qui peuvent avoir été introduites, comme des variations de codage de caractères (on se souviendra que XML accepte différents codages), d'ordre de définition d'attributs, de syntaxe d'écriture des éléments `empty`, etc.

Cette notion d'équivalence est nécessaire, dès lors que l'on introduit, par exemple, des notions de signature de document (XML-Signature) ou dès lors que l'on s'intéresse, dans XPath, à des positions relatives d'objets les uns par rapport aux autres.

Cette spécification est mineure : son utilité n'est définie que pour des environnements applicatifs extrêmement précis. Elle n'a pas de volonté de devenir une des recommandations de base du W3C, comme peuvent l'être XPath ou XSLT.

Principes

Un "canoniseur" peut générer un document XML "bien formé" ; il peut aussi générer une partie de document. Dans ce dernier cas, la spécification de ce qui doit être canonisé est réalisée en utilisant une expression issue du langage XPath.

D'un point de vue fonctionnel, le codage de caractères du document généré est le codage UTF-8. Basée sur le modèle de donnée de XPath, le reste de la spécification s'intéresse surtout à des notions d'ordonnancement d'objets (les attributs, par exemple) et à des notions de normalisation d'attributs et de "retour-chariot".

Canonical définit la spécification d'un outil, de ce qu'il doit interpréter dans un document et ce qu'il doit générer. Cette spécification n'a pas pour objectif d'établir une méthode définissant l'équivalence de deux documents XML dont les formes canoniques sont identiques : elle se contente de donner aux applicatifs les moyens de le faire.

Recommandations(s)

■ ■ *XML canonique*

Recommandation, version 1.0, du 15-03-2001

Document sur <http://www.yoyodesign.org/doc/w3c/xml-c14n/index.html>

🇺🇸 *Canonical XML*

Recommandation, version 1.1, du 02-05-2008

Document sur <http://www.w3.org/TR/xml-c14n>

🇺🇸 *XML Canonicalization Requirements*

Note, version 19990605, du 05-06-1999

Document sur <http://www.w3.org/TR/NOTE-xml-canonical-req>

■ ■ *La canonisation XML exclusive*

Recommandation, version 1.0, du 18-07-2002

Document sur <http://www.yoyodesign.org/doc/w3c/xml-exc-c14n/>

🇺🇸 *Exclusive XML Canonicalization*

Recommandation, version 1.0, du 18-07-2002

Document sur <http://www.w3.org/TR/xml-exc-c14n>

TIRÈME SARL

Infoset, XML Information Set

Rédaction : Pierre Attar

Pour comprendre [Infoset](#), il faut se souvenir qu'[XML](#) définit un format d'échange de données structurées selon leur modèle. Ce format d'échange est, par définition, contenu dans un fichier séquentiel comportant un début et une fin. [Infoset](#) est alors la vue "*désérialisation*" d'un fichier [XML](#) : c'est alors la structure logique d'un document décrit par la syntaxe [XML](#).

La spécification définit le jeu de données abstrait qu'il est nécessaire de construire à partir de la lecture d'un document [XML](#) "bien formé", lors de sa représentation sous forme d'arbre d'objets typés et ordonnés. Cette représentation est celle que doit présupposer n'importe quelle application manipulant de l'information [XML](#) : les chargeurs [XML](#) sont alors supposés reporter, d'une façon ou d'une autre, ces éléments d'informations aux applications.

Le jeu d'informations différencie ce qui est absolument nécessaire (document, élément, attribut, *processing instruction*, entités, caractères, notations, déclarations d'espaces de noms) et ce qui est périphérique ([DTD](#), commentaires, etc.).

Objectifs

L'objectif de cette spécification est de faire partager à des spécifications de plus haut niveau, une vision commune de ce qui est significatif dans un document [XML](#) (par exemple, [DOM Niveau 3](#) est basé sur l'[Infoset](#)). Tel que défini dans la spécification, [Infoset](#) est alors la représentation commune à tous les outils d'un document [XML](#).

Le fait de définir une représentation commune est extrêmement important, cela à plusieurs titres. En tout premier, les développeurs d'applications [XML](#) se noient, aujourd'hui, dans tous les modèles de données proposés représentant ce qui est réellement contenu dans un document, après *désérialisation*. Du coup, écrire des programmes est souvent ardu, il faut savoir quel outil on utilise pour se souvenir des données qu'il manipule. Par ailleurs, avoir un modèle de données partagé permet aux outils de création d'informations [XML](#) de mieux ajuster la façon dont ils utilisent et délivrent les fichiers [XML](#). Enfin, et comme dans le cas de la spécification [Canonical](#), cette représentation commune permettra une simplification des outils, puisque le modèle supprime les différentes formes possibles pour le codage [XML](#) d'une même information logique. Par exemple, [Infoset](#) permet de voir la même information logique dans les codages suivants : `<test type="essai">`, `<test type='essai'>`, `<test type="essai">`, `<test type="essai">`, `<test type='essai'>`. Ceci est possible, car il existe dans l'[Infoset](#) une normalisation du codage des caractères.

À noter que [XPath](#) et, donc, [XSLT](#) n'utilisent pas exactement le même jeu d'informations, du fait de la non-disponibilité de cette spécification au moment de l'élaboration de ces deux standards. Cependant, l'annexe B de [XPath](#) explique comment interpréter l'arbre abstrait [XPath](#) en fonction d'[Infoset](#).

Principes

Les différents éléments d'informations reconnus sont les suivants :

- . document (obligatoire),
- . éléments (obligatoire),
- . attributs (obligatoire),
- . processing instructions (obligatoire),
- . références à des entités non lues par un *parser* non validant (obligatoire),
- . caractères (obligatoire),
- . notation (obligatoire),
- . déclaration d'espaces de noms (obligatoire),
- . [DTD](#) (périphérique),
- . entités générales déclarées dans la [DTD](#) (obligatoire pour les entités non *parsées*, sinon, périphérique),
- . début et fin de localisation d'entité incluse (périphérique),
- . début et fin de de section définie comme étant `CDATA` (périphérique),
- . commentaires (périphérique).

À chaque élément d'information est lié un ensemble de propriétés, qui diffère selon l'élément d'information. Par exemple, pour un élément, les propriétés définies s'intéressent à son espace de nom et son nom, à son parent, ses fils et ses attributs, aux déclarations d'espaces de noms sur cet élément et à tous les espaces de noms propagés par ses parents, à l'[URI](#) de base (voir [XML Base](#)), si elle existe, propagée par ses parents.

[Infoset](#) s'intéresse à un document *parsé* et validé. Du coup, aucune information n'est donnée sur la [DTD](#), si ce n'est, de façon optionnelle, sa désignation. Charge aux programmes de récupérer la [DTD](#) elle-même s'ils en ont besoin.

Pour conclure, beaucoup de débats existent sur l'utilité d'une telle spécification. Par exemple, si [Infoset](#) présuppose une organisation sous forme d'arbre, avec des noeuds comportant des parents et des fils, certains rejettent cette vision d'[XML](#), en argumentant que lorsqu'ils échangent des champs de tables issues de bases de données, il n'ont pas besoin de toute cette information liée à des relations dans un arbre.

De fait, la spécification est très controversée et beaucoup de débats ont lieu sur le sujet (pour plus de détails sur les argumentaires, voir la liste de débat [[Monthly Archives for xml-dev](#)], dans ses [[Archives de juillet 2000](#)] et ses [[Archives d'août 2000](#)] 2000).

Il semble cependant important de disposer de ce type de spécification, dès lors que beaucoup de recommandations du [W3C](#) s'intéressent au traitement des documents [XML](#). En effet, quelle serait la validité d'une spécification de traitement, si le modèle de donnée utilisé était laissé au libre choix de l'implémenteur ?

Recommandations(s)


[L'ensemble d'information XML \(Infoset\)](#)

Recommandation, version 20011024 , du 24-10-2001

Document sur <http://www.a525g.com/programmation/xml-infoset.htm>

[XML Information Set \(Second Edition\)](#)

Recommandation, version second edition, du 04-02-2004
Document sur <http://www.w3.org/TR/xml-infoset/>

 *XML Information Set Requirements*
Note, version 19990218, du 18-02-1999
Document sur <http://www.w3.org/TR/NOTE-xml-infoset-req>



XML Catalogs, XML Catalogs

L'objectif des catalogues d'XML est de pouvoir utiliser, dans les "*internal subset*" des DTD, des noms d'entités externes symboliques : charge alors aux logiciels de parsing, s'occupant de la résolution des entités, de chercher, au sein d'un XML Catalogs, le nom physique du fichier à associer au nom logique.

L'avantage ? Il devient alors possible d'utiliser des adresses physiques de fichiers qui puissent être différentes d'un système à l'autre.

L'idée est simple mais très efficace. OASIS Open, aujourd'hui promoteur de ce standard, l'a déjà mise en place avec succès pour les applications SGML (voir spécification de SGML [sur internet : www.oasis-open.org/specs/a401.htm]).

Objectifs

Lorsqu'une entité externe est déclarée dans la DTD d'un fichier XML, il est toujours nécessaire, même si elle utilise un nom publique, d'identifier le fichier XML correspondant. Sur Internet, tout va bien ! Le nom du fichier désigne une URL qui est toujours accessible. Cependant, XML est aujourd'hui aussi utilisé hors la sphère Internet.

Ainsi, si un fichier toto.xml est utilisé sur un répertoire local, on le désigne par le chemin suivant : /temp/mesDoc/toto.xml. La déclaration de ce fichier sous forme d'entité paramètre s'écrira alors :

```
<!ENTITY toto PUBLIC "Mon Beau Document" "/temp/mesDocs/toto.xml">
```

Du coup, si ce document est transféré dans un autre environnement ou si le fichier toto.xml est stocké dans un autre répertoire, la résolution de la déclaration ne pourra plus être effectuée ! La communauté SGML a depuis longtemps compris ce problème et a mis en place un système de catalogue [sur internet : www.oasis-open.org/specs/a401.htm] permettant, sur chaque système, de réaliser une correspondance entre des noms *logiques* et des noms *physiques* de ressources.

C'est ce même système qui est aujourd'hui proposé par XML Catalogs pour les documents XML. Un XML Catalogs XML Catalogs est donc présent sur chaque système et l'indépendance des systèmes de fichiers est alors assurée.

Pour finir, il faut noter la nécessité d'étendre l'utilisation des XML Catalogs à d'autres informations que les entités externes. Ainsi, un standard comme XInclude devrait pouvoir aussi profiter de cette indépendance des systèmes de fichiers et/ou d'URL.

Recommandations(s)

 XML Catalogs, Committee specification
Projet en cours, version 1.0, du 24-10-2002

Document sur <http://www.oasis-open.org/committees/entity/specs/cs-entity-xml-catalogs-1.0.html>

TIRÈME SARL

Fragment Interchange, XML Fragment Interchange

Rédaction : Pierre Attar

La norme XML supporte des documents logiques dont le modèle abstrait est un arbre d'objets typés. Si, par exemple, une base de données stocke un document logique complet de quelques centaines de méga-octets (comme c'est fréquemment le cas pour une documentation technique de systèmes complexes), il peut être nécessaire de visualiser ou d'éditer une (ou plusieurs) entités (ou des parties d'entités), sans vouloir pour cela travailler sur le document entier.

Si l'activité doit être liée à la modification d'un sous-arbre, il est alors nécessaire de fournir à l'éditeur d'un tel fragment les informations appropriées concernant le contexte du fragment, dans le document englobant qui n'est pas à la disposition du destinataire.

Le Groupe de travail sur les fragments XML se propose donc de définir une façon d'échanger des fragments de documents XML.

Objectifs

Le Groupe de travail sur les fragments XML se propose donc de définir une façon d'échanger des fragments de documents XML, avec leur contexte d'utilisation qui contiendrait toute l'information nécessaire à des activités de *parsing*, de visualisation et de modification.

La dernière version de cette spécification date de juin 1999. Depuis lors, il semble que, d'une part, aucune force du W3C ne soit affectée à cette spécification et que, par ailleurs, elle ne suscite que peu d'intérêt, au regard de la liste de diffusion enregistrant les commentaires sur la spécification. Dire que cette spécification ne verra pas le jour rapidement relève alors du pléonasme. Pourtant, elle pourrait prendre toute sa valeur dans les activités d'édition et de modification de document, pour lesquelles se posent sans arrêt la question de savoir comment envoyer vers l'outil d'édition l'information minimum de contexte : souvent, la question est résolue par des transformations *ad hoc*.

Principes

Le principe de la spécification est de définir toute l'information de contexte nécessaire à des activités de *parsing*, de validation et d'édition d'un fragment XML. Ces informations de contexte s'intéresseront aux DTD, aux parents et frères du fragment échangé, pour y définir les attributs et leurs contenus directs.

Recommandations(s)

 *XML Fragment Interchange*

Recommandation Candidate, version 20010212, du 12-02-2001

Document sur <http://www.w3.org/TR/xml-fragment>

TIRÈME SARL

XInclude, XML Inclusions

Rédaction : Pierre Attar

Recommandation(s) liée(s) : [Infoset](#)

Beaucoup de langages de programmation fournissent un mécanisme d'inclusion, afin de faciliter la modularité. Cette proposition présente un mécanisme générique pour fusionner des documents [XML](#) dans un même et unique document [XML](#).

Objectifs

L'objectif de [XInclude](#) est de permettre de modulariser l'écriture de documents [XML](#) et d'atteindre des objectifs de réutilisation de fragments partagés.

Utiliser [XInclude](#) relève d'un processus basé sur la forme *désérialisée*, en [Infoset](#), de documents [XML](#). Plus précisément, *réaliser l'inclusion n'est en aucun cas le rôle d'un parser*, mais plutôt celui d'un processus, déclenché, si nécessaire, une fois générées les [Infoset](#) du document contenant les inclusions. Le résultat de ce processus sera un nouveau jeu d'[Infoset](#), résultant de la fusion du jeu primaire et de tous les jeux des objets à inclure. En ce sens, tous les processus applicatifs pourront décider quelle vue ils souhaitent obtenir du document primaire : une vue avec inclusions identifiées ou une vue avec inclusions réalisées.

Principes

Le mécanisme est différent de celui proposé par [XLink](#), car indépendant des applications. En effet, dans [XLink](#), ce que doit générer, en termes de comportement, un processeur de l'attribut `show= 'embed'`, n'est pas spécifié de façon obligatoire.

Le mécanisme est aussi différent de celui utilisé par les [DTD](#) pour les entités externes, pour deux raisons principales :

- ce sont les parseurs qui font la résolution des entités. En ce sens, l'information disponible dans le modèle abstrait de données est soit inexistante, soit extrêmement minimale. Cette information est, pourtant, par moment, intéressante, lors de la mise à jour de documents dont les différents morceaux sont à stocker dans différents documents partagés ;
- les notions d'entités générales nécessitent l'utilisation, même minimale, de [DTD](#), alors qu'actuellement, beaucoup d'utilisations de [XML](#) se font sans [DTD](#) et qu'il est à prévoir que ce processus s'amplifiera, dès lors que les [Schema](#) seront standardisés.

D'un point de vue technique, pour utiliser ce mécanisme d'inclusion, il faut se situer dans l'espace de noms <http://www.w3.org/1999/XML/xinclude>. Ceci fait, deux attributs assurent l'inclusion, étant posés sur n'importe quel élément :

`xinclude:href` : permet de définir l'objet à inclure. Cela peut être un document (`something.xml` ou une portion de document (`#xpointer(x/myinclude[1])`) ;
`xinclude:parse` : définit la méthode d'inclusion pour savoir s'il faut *parser* (valeur `xml`) ou non (valeur `text`) l'**Infoset** inclus.

Pour conclure, **XInclude** souffre de la même faiblesse que le système de gestion des entités externes dans les **DTD** : le mécanisme de désignation des fichiers est contenu dans les documents **XML** eux-mêmes. On suivra alors avec intérêt la façon dont **XInclude** évoluera, dès lors que les propositions de catalogue externe (**XML Catalogs**) seront mises en place.

Recommandations(s)

■ ■ *Les inclusions XML (XInclude)*

Recommandation, version 1.0, du 20-12-2004

Document sur <http://www.yoyodesign.org/doc/w3c/xinclude/#>

🇺🇸 *XML Inclusions (XInclude)*

Recommandation, version 1.0 (Second Edition), du 15-11-2006

Document sur <http://www.w3.org/TR/xinclude/>

TIRÈME SARL

XML Language, Langue d'un objet XML

Rédaction : Pierre Attar

Recommandation(s) liée(s) : [IETF LANG](#)

La notion de langue est partie intégrante de la spécification [XML](#) . Elle permet de définir, de manière commune à tous les documents du Web, dans quelle langue a été écrit le contenu d'un document ou d'un élément (sous-arbre) [XML](#).

Objectifs

L'objectif de cette partie de la recommandation est de standardiser la façon de décrire une langue. Les processus de traitement (ceux des vérificateurs orthographiques, mais surtout des afficheurs) pourront alors déclencher les traitements appropriés à une langue donnée. Par exemple, il devient possible, grâce à cette partie de la recommandation de juxtaposer un texte en arabe et un texte en français : l'afficheur pourra, grâce à la langue, savoir que certaines informations doivent être affichées de gauche à droite (le français) tandis que d'autres s'afficheront de droite à gauche (l'arabe).

Principes


La recommandation utilise, pour identifier de façon non ambiguë les codes pays et langues, une spécification de *IETF* , qui, elle-même, se repose sur les codifications de *ISO*.

Dans les documents [XML](#) eux-mêmes, tout élément est susceptible d'avoir un attribut réservé : `xml:lang`. Poser cet attribut sur un élément définit la langue utilisé pour *tout* le contenu du sous-arbre. L'utilisation de cet attribut fonctionne donc par propagation... tant qu'un nouvel élément ne vient pas contredire l'effet propagé.

Exemple

```
<article xml:lang="fr">
<!-- tout le contenu de l'article est par défaut en français -->
<titre>De la bonne utilisation des langues avec XML</titre>
<titre xml:lang="en">
  <!-- l'effet de propagation de xml:lang="fr" s'arrête, le
  contenu est maintenant en anglais -->
  Best use of language with XML
</titre>
</article>
```

Recommandations(s)

 *Extensible Markup Language (XML) 1.1 (Second Edition)*
Recommandation, version 1.1, du 16-08-2006
Document sur <http://www.w3.org/TR/xml11#sec-lang-tag>

TIRÈME SARL

XPointer, XML Pointer Language

Rédaction : Pierre Attar

Si *XPath* définit une base commune aux langages d'adressage pour pointer sur des objets contenus dans un document, les mécanismes d'adressage nécessaires aux liens de mise en relation nécessitent des fonctionnalités complémentaires.

Construite sur *XPath*, la spécification *XPointer* définit de *nouvelles fonctionnalités*, telle la possibilité de définir une région (un ensemble de mots), sur laquelle doit être faite la mise en relation. *XPointer* se définit alors en un ensemble d'extensions à *XPath*.

Objectifs

Avec *XLink*, l'objectif de *XPointer* est de définir les mécanismes nécessaires à la mise en relation d'informations contenues dans des documents. Si *XLink* s'intéresse au lien lui-même, *XPointer* définit les objet potentiellement "atteignables".

Principes

XPointer est une recommandation basée sur les *Infoset*. Elle permet, par itérations successives, de désigner, de façon la plus précise possible, une cible "atteignable" dans un document : un élément d'information structuré au sens *Infoset*, voire une portion de document.

Les extensions ajoutées permettent de désigner des objets autres que les seules structures reconnues dans *XPath* pour :

- permettre de pointer directement dans un chaîne de caractères, *XPointer* utilise la notion de `point`. Un point est, dans un noeud conteneur, une position numérique (`index`). Cette notion de point peut aussi s'appliquer aux noeuds contenus dans le conteneur ;
- permettre de désigner un ensemble séquentiel d'informations, de façon indépendante de l'arbre d'un document *XML*, *XPointer* utilise la notion de `range` qui se définit comme étant tout ce qui est entre un point de départ et un point d'arrivée. Il est alors possible de désigner une chaîne de caractères où le début se trouve dans un paragraphe et la fin dans un autre paragraphe ;
- mieux trouver un objet de la structure, *XPointer* propose aussi la possibilité de désignation par recherche de chaîne de caractères.

À ces extensions sont ajoutées des fonctions spécifiques de manipulation de `points` et de `range`.

Enfin, et dans un but de complétude, on notera que **XPointer** introduit les notions de schéma de pointage. Cela permettra, à l'avenir, de pouvoir utiliser de façon concurrente, différents mécanismes de pointage. Dans la recommandation actuelle, seul, un mécanisme est proposé (celui présenté plus haut). Du coup, pour permettre cette coopération, il est possible, dans la spécification, d'utiliser plusieurs expressions **XPointer** concurrentes, pour un même adressage.

Recommandations(s)

■ ■ *Le cadre XPointer*

Recommandation, version 20330325, du 25-03-2003

Document sur <http://www.yoyodesign.org/doc/w3c/xptr/framework/>

🇺🇸 *XPointer Framework*

Recommandation, version 20030325, du 25-03-2003

Document sur <http://www.w3.org/TR/xptr-framework/>

■ ■ *Le système element() de XPointer*

Recommandation, version 20030325, du 25-03-2003

Document sur <http://www.yoyodesign.org/doc/w3c/xptr/element/>

🇺🇸 *XPointer element() Scheme*

Recommandation, version 20030325, du 25-03-2001

Document sur <http://www.w3.org/TR/xptr-element/>

■ ■ *Le système xmlns() de XPointer*

Recommandation, version 20030325, du 25-03-2003

Document sur <http://www.yoyodesign.org/doc/w3c/xptr/xmlns/>

🇺🇸 *XPointer xmlns() Scheme*

Recommandation, version 20030325, du 25-03-2001

Document sur <http://www.w3.org/TR/xptr-xmlns/>

🇺🇸 *XML Pointer Language*

Projet en cours, version 1.0, du 16-08-2002

Document sur <http://www.w3.org/TR/xptr/>

TIRÈME SARL

XLink, XML Linking Language

Rédaction : Pierre Attar

Recommandation(s) liée(s) : [RDF](#) - [Topic Maps](#) - [XTM](#)

Si le web est une immense base de liens, il est alors nécessaire de proposer un modèle de lien qui permette de prendre en compte les notions de sources ou destinations multiples dans un lien. C'est ce que doit permettre [XLink](#), qui propose de construire des liens unidirectionnels ou multidirectionnels qui puissent être sémantisés et *valués*, pour indiquer l'objectif de la pose d'un lien par un auteur.

Objectifs

L'objectif de [XLink](#) est d'élargir les notions de liens extrêmement confinées, d'une part, dans [HTML](#) (où seul un lien unidirectionnel et à destination unique existe) et, d'autre part, dans [XML](#) où seul le mécanisme `ID/IDREF` est défini). L'objectif est aussi la mise en place des liens vers les contenus de documents [XML](#), sans pour autant être obligés d'identifier des destinations uniques dans le document cible... toute chose qui nécessiterait, sans ce type de spécification, de pouvoir intervenir sur ce même document.

[XLink](#) définit donc des liens : cette recommandation est complémentaire à [XPointer](#) qui définit des mécanismes de définitions d'adresse. On retrouve une fois de plus l'objectif de factorisation du *W3C* : il vaut mieux réaliser deux recommandations : l'une et l'autre pourront alors être réutilisable par ailleurs.

Peu d'acteurs se prononcent sur l'intégration de ces spécifications directement à l'intérieur des systèmes de navigation du Web. Il reste toujours possible de l'implémenter dans les bases de données et d'effectuer une transformation de cette information vers [HTML](#) : cela simplifiera les tâches de gestion de site, mais cela les simplifiera d'autant plus si des outils de gestion de liens basés sur le standard existent. Mais implémenter [XLink](#) sur un logiciel de navigation n'est pas suffisant : il faudra aussi disposer de véritables bases de données adaptées et disposer des outils de traitement, pour, par exemple, assurer la qualité des liens créés.

Principes

[XLink](#) définit un jeu d'attributs permettant de valuer *n'importe quel élément XML* d'un pouvoir de représentation de lien. Deux types de liens sont envisagés : les liens simples (*simple link*) et les liens étendus (*extended link*).

Comme pour un lien [HTML](#) de type `A` ou `IMG`, le *lien simple* permet de mettre en relation, de façon unidirectionnelle, une source (*starting resource*) et une destination (*ending resource*).

Outre la définition de la source, et à la différence des liens [HTML](#), le lien simple peut être *valué* :

- `role` : permet de définir une typologie de liens (par exemple, un lien est de type note de bas de page, de type référence bibliographique, de type CV d'un auteur, etc.) ;
- `title` : permet de titrer un lien (ce titre pourra être utilisé par un système de navigation) ;
- `show` : permet de définir ce que l'on doit faire de la destination pointée, qui peut remplacer (`replace`) celle existante, s'ajouter à l'existante (`embed`) ou simplement être vue ailleurs, dans une nouvelle fenêtre (`new`) ;
- `actuate` : permet de définir si le lien doit être activé au moment du chargement (`onLoad`) ou sur demande (`onRequest`).

À noter que la spécification ne définit pas de façon formelle comment doivent être interprétés et utilisés les attributs `role`, `title`, `show` et `actuate`. Elle se contente de donner des indications, car elle reste indépendante des usages et représentations.

Le *lien étendu* différencie la définition des objets participant à un lien (`resource`) de la façon dont ils sont mis en relation (`arc`). Les ressources peuvent être locales (à l'endroit où est défini le lien) ou éloignées (`remote`). Quand la ressource est locale, elle peut néanmoins être factorisée de façon externe au document. Pour ce faire, on utilisera les principes de `linkbase`.

Les `arcs`, quant à eux, sont définis en utilisant les rôles. Mettre en place un lien multidirectionnel consiste seulement à définir un type de rôle particulier et affecté à plusieurs ressources.

Recommandations(s)

- ■ *Le langage de liaison XML (XLink) version 1.0*
Recommandation, version 1.0, du 27-06-2001
Document sur <http://www.yoyodesign.org/doc/w3c/xlink/>
- 🇺🇸 *XML Linking Language (XLink)*
Recommandation, version 1.1, du 06-05-2010
Document sur <http://www.w3.org/TR/xlink11/>
- 🇺🇸 *XLink Markup Name Control*
Note, version 20001220, du 20-10-2000
Document sur <http://www.w3.org/TR/xlink-naming/>
- 🇺🇸 *HLink, Link recognition for the XHTML Family*
Projet en cours, version 20020913, du 13-09-2002
Document sur <http://www.w3.org/TR/hlink/>

TIRÈME SARL

XPath, XML Path Language

Rédaction : Pierre Attar

Recommandation(s) liée(s) : [XML Query](#)

[XPath](#) est un langage qui permet d'adresser, de désigner, des objets structurels contenus dans un document [XML](#). Il est conçu pour être utilisé, tant par [XSLT](#) que par [XPointer](#) ou, encore, par [XML Query](#), qui ajouteront à cette recommandation partagée les fonctionnalités propres qui leurs sont nécessaires. [XPath](#) est donc une recommandation de base, externalisée pour être partagée par un ensemble de recommandations de plus haut niveau.

Objectifs

Si un document [XML](#) contient en même temps des données et des informations permettant d'identifier la structure et le sens de ces données, il est alors utile de pouvoir s'appuyer sur cette information pour désigner une partie d'un document [XML](#).

C'est utile lorsque l'on réalise des applications de présentation, par exemple, pour faire une table des matières où l'on ne veut sélectionner que des titres. C'est également utile lorsque l'on veut réaliser des hyperliens sur des documents que l'on ne peut pas modifier pour leur ajouter des ancrs et que l'on souhaite pourtant désigner (par exemple, le deuxième alinéa du troisième chapitre d'un document).

Du coup et pour tous ces propos, il est nécessaire d'avoir un langage de désignation d'objets dans un document ; c'est l'objectif de [XPath](#). Du point de vue du [W3C](#), l'objectif de [XPath](#) est aussi de devenir un *standard de base*, réutilisable dans des recommandations de plus haut niveau.

Principes

Pour désigner un objet dans un document, [XPath](#) propose, d'une part, un langage d'adressage d'objets et, d'autre part, un ensemble de fonctions permettant d'augmenter le pouvoir d'expression du langage.

Le langage d'adressage utilise des chemins, pour désigner un ensemble d'objets ; la désignation de chemins se repose sur des notions d'axes et de sélection.

Plus précisément, un chemin, qui peut être absolu ou relatif, utilise des éléments (la recommandation parle d'*étapes*) de localisation qui se décomposent en :

un axe, choisi parmi les attributs (*attribute*), les fils (*child*), les parents (*parent*) ou tous les ancêtres (*ancestor*), les frères (*following-sibling* ou *preceding-sibling*), etc. ;

un test, sur un type de noeud, par exemple, les ancêtres de type `chapter` (`ancestor::chapter`);
 un ou plusieurs prédicats, par exemple, le dernier des fils de type `chapter` (`child::chapter[position()=Last()]`).

Ces étapes se composent entre elles en utilisant l'opérateur `/`. Par exemple, le titre du parent de type `chapitre` s'écrira `ancestor::chapter/titre`.

Le résultat d'un chemin d'adressage est un *ensemble de noeuds (node set)* dans l'arbre XML du document, noeuds qui sont traités ensuite par des processus applicatifs *ad hoc*, qui intègrent XPath comme chemin d'adressage.

La syntaxe complète de XPath étant, par moment, trop "verbeuse", la recommandation inclut une syntaxe abrégée, allant directement à l'essentiel. Ainsi, `//p` sera une abréviation de `self::node()/child::p`: tous les éléments `p` contenus dans le sous-arbre en cours d'exploration (utilisation de l'axe `self`) et, cela, quelle que soit leur position dans ce sous-arbre.

Enfin, pour définir les prédicats, XPath propose un langage d'expressions, utilisant un ensemble de fonctions de base, telles les extractions de chaînes de caractères (`concat`, `contains`, `start-with`, ...), les explorations de position d'un noeud parmi ses frères (`position`, `last`, `count`, ...), etc.

Modèle de données XPath et Infoset

Le modèle de données d'XPath comporte 7 types de noeuds (Infoset parle d'*éléments d'information*): racine, élément, texte, attribut, espace de nom, *processing instruction* et commentaire.

Les deux différences majeures avec la recommandation Infoset sont, d'une part, que les attributs sont considérés comme étant des noeuds d'arbre (même s'ils ont un statut spécifique, en ce sens qu'ils n'ont pas de fils) et, d'autre part, que les noeuds "texte", pour XPath, contiennent un ensemble de caractères, alors qu'Infoset distingue chaque caractère individuellement. La recommandation XPath définit alors, en annexe, comment interpréter les noeuds en fonction d'XPath.

Recommandations(s)

Langage XML Path

Recommandation, version 1.0, du 16-11-1999
 Document sur <http://xmlfr.org/w3c/TR/xpath>

XML Path Language

Recommandation, version 2.0, du 23-01-2007
 Document sur <http://www.w3.org/TR/xpath20/>

XPath Requirements

Projet en cours, version 2.0, du 03-06-2005
 Document sur <http://www.w3.org/TR/xpath20req>

XQuery 1.0 and XPath 2.0 Functions and Operators

Recommandation, version 1.0, du 23-01-2007
 Document sur <http://www.w3.org/TR/xpath-functions/>

XQuery 1.0 and XPath 2.0 Data Model (XDM)

Recommandation, version 20070123, du 23-01-2007
 Document sur <http://www.w3.org/TR/xpath-datamodel/>

 *XQuery and XPath Full Text 1.0*

Recommandation Candidate, version 1.0, du 28-01-2010

Document sur <http://www.w3.org/TR/xpath-full-text-10/>

 *XQuery and XPath Full Text 1.0 Requirements*

Projet en cours, version 20080516, du 16-05-2008

Document sur <http://www.w3.org/TR/xpath-full-text-10-requirements/>

 *XQuery and XPath Full Text 1.0 Use Cases*

Projet en cours, version 1.0, du 28-01-2010

Document sur <http://www.w3.org/TR/xmlquery-full-text-use-cases>

TIRÈME SARL

DOM, Modèle de document objet

Rédaction : Pierre Attar

Recommandation(s) liée(s) : [SAX](#) - [XSLT](#)

DOM définit, sous la forme d'une API orientée objet et indépendante des langages de programmation et de script, les outils nécessaires à l'accès et à la manipulation de documents structurés sous forme d'arbres d'objets typés. Les documents peuvent être typés selon le vocabulaire [HTML](#), en utilisant une syntaxe [XML](#) ou toute autre forme de syntaxe.

Ce sont les niveaux d'implémentation qui définissent le type de structure qu'ils reconnaissent :

- [DOM](#) niveau 1 prend en compte les documents [HTML](#) et [XML](#) ;
- [DOM](#) niveau 2 ajoute à cette liste les [CSS](#) ;
- [DOM](#) niveau 3 permettra aussi l'accès aux modèles (structurés sous forme de [DTD](#), de [Schema](#) ou tout autre formalisme).

Objectifs

L'objectif de cet ensemble de recommandations est de fournir des outils de haut niveau permettant d'accéder et de manipuler un document [XML](#). Ces outils étant définis selon une spécification formelle, indépendante des langages et des systèmes, cette spécification permet de réaliser des développements portables, dans différents environnements.

Au départ, l'objectif était de permettre la mise en oeuvre de programmes dans un logiciel de consultation du Web, de façon indépendante de son éditeur ; aujourd'hui, les recommandations vont beaucoup plus loin, pour s'intéresser à toute activité de programmation, basée sur un arbre d'objets typés et valués : une différence notable avec [SAX](#), qui s'intéresse aux événements rencontrés dans un document, lors de sa lecture.

Les différents niveaux de [DOM](#) élargissent le champ d'action de la spécification : en ajoutant plus de méthodes à des objets ou en définissant de nouvelles classes d'objets. Cependant, les outils logiciels peuvent décider de n'implémenter qu'un niveau particulier. Toutefois, s'ils implémentent le niveau 2, ils implémentent par là-même le premier niveau de la recommandation.

Principes

[DOM](#) est la définition d'une API (Application Programming Interface) orientée objet. La spécification est organisée en différentes parties, différenciant des niveaux que doit supporter une implémentation de [DOM](#).

Pour chaque niveau, les spécifications différencient ce qui est fondamental (et qui doit donc être implémenté pour être conforme) de ce qui est optionnel (les extensions). Par ailleurs, chaque spécification s'intéresse au "coeur" (*core*) qui est générique, pour n'importe quel document XML et à des interfaces spécifiques, pour des types de documents donnés (tels des documents XML, des CSS, voire des objets plus génériques comme des feuilles de styles).

Le "coeur" (*core*) de DOM définit les éléments ci-après :

- ce que sont une implémentation DOM et les nature d'erreurs reconnues ;
- les interfaces de gestion d'un document XML pour deux objectifs : y accéder (fonction d'accès à des éléments, attributs, etc.) ou le construire (fonctions de création de documents, d'éléments, d'attributs, etc.). Le niveau 2 prend notamment en compte les notions d'espace de noms ;
- les extensions au coeur, qui sont optionnelles dans les implémentations, ajoutent : les notions de sections CDATA et la gestion des entités externes (niveau 1), ainsi que les notions de DTD, de *processing instructions*, de notation (niveau 2).

Le niveau de 1, complété ensuite par le niveau 2, définissent les fonctionnalités nécessaires à la manipulation de documents HTML (il s'agit bien de HTML et non pas d'XHTML : voir annexe C.11 de la recommandation XHTML [*sur internet : www.la-grange.net/w3c/xhtml1/*]). Ce sont surtout des objets nouveaux qui apparaissent, comme les *frame*, les images ou les ancres. Les fonctionnalités de manipulation sont minimales, celles du *core* étant *a priori* souvent suffisantes. À noter que presque tout est défini dans le niveau 1 et que la spécification de niveau 2 n'ajoute que quelques détails.

Toutes les nouvelles interfaces sont ensuite définies dans le niveau 2, elles concernent des :

- fonctionnalités de manipulation des CSS, les feuilles de styles associées aux documents HTML. À noter que la spécification est prévue pour prendre en compte d'autres feuilles de styles, plus tard (par exemple, XSL). Pour ce faire, elle définit au préalable les notions de vues (*Document Object Model Views*) associées à un document et les notions de feuilles de styles (*Document Object Model StyleSheets*) ;
- fonctionnalités d'accès à des événements liées à des changements d'état d'un document ou à des actions de l'utilisateur sur un document (ou un objet contenu dans un document) ;
- fonctionnalités de parcours de document XML, avec ou sans filtre ;
- fonctionnalités d'identification de portion de document, basées sur les informations de structure ou sur des contenus quelconques, identifiés par leur ordre séquentiel. À noter que ces fonctionnalités sont essentielles, pour prendre en compte les notions de position associées à la recommandation XPointer.

Aujourd'hui, seuls les niveaux 1 et 2 ont le statut de recommandation. Le niveau 3 est en cours de validation. Si l'utilisation de DOM ne concerne que le fait de pouvoir manipuler un document dans un outil de consultation du Web, le niveau 1 apporte alors suffisamment de fonctionnalités pour mettre en place bon nombre d'applications, pas trop complexes.

Si DOM doit être utilisé comme langage généralisé de manipulation de documents XML, il est alors nécessaire d'utiliser le niveau 2.

Exemple

Voici, en Java, comment pourrait s'écrire un programme DOM dont le seul objectif serait de créer un document XML simple.

```
Document doc = new DocumentImpl();
Element outElem = doc.createElement("object");
outElem.setAttribute("identifiant", "A020");
Node subJNode = doc.createElement("subject");
```

```

outElem.appendChild(subjNode);
subjNode.appendChild("du texte dans le subject");
dumpSubTree(outElem, out); // une fonction qui génère la forme
sérialisée

```

Ce simple exemple créera le fichier XML suivant :

```


<object identifier="A020">
  <subject>du texte dans le subject</subject>
</object>


```


Recommandations(s)


- ■ *Modèle Objet de Documents (DOM) Spécification niveau 1*
Recommandation, version 1.0, du 01-10-1998
Document sur <http://xmlfr.org/w3c/TR/REC-DOM-Level-1/>
- ■ *La spécification du Modèle Objet de Document (DOM) niveau 2 - Noyau*
Recommandation, version 1.0, du 13-11-2000
Document sur <http://www.yoyodesign.org/doc/w3c/dom2/core/Overview.html>
- ■ *La spécification du Modèle Objet de Document (DOM) niveau 2 - Événements*
Recommandation, version 1.0, du 13-11-2000
Document sur <http://www.yoyodesign.org/doc/w3c/dom2/events/Overview.html>
- ■ *La spécification du Modèle Objet de Document (DOM) niveau 2 - HTML*
Recommandation, version 1.0, du 09-01-2003
Document sur <http://www.yoyodesign.org/doc/w3c/dom2/html/Overview.html>
- ■ *La spécification du Modèle Objet de Document (DOM) niveau 2 - Traversée et étendue*
Recommandation, version 1.0, du 13-11-2000
Document sur <http://www.yoyodesign.org/doc/w3c/dom2/traversal-range/Overview.html>
- ■ *La spécification du Modèle Objet de Document (DOM) niveau 2 - Style*
Recommandation, version 1.0, du 13-11-2000
Document sur <http://www.yoyodesign.org/doc/w3c/dom2/style/Overview.html>
- ■ *La spécification du Modèle Objet de Document (DOM) niveau 2 - Vues*
Recommandation, version 1.0, du 13-11-2000
Document sur <http://www.yoyodesign.org/doc/w3c/dom2/views/Overview.html>
- 🇺🇸 *Document Object Model Level 1 Specification*
Recommandation, version 1.0, du 01-10-1998
Document sur <http://www.w3.org/TR/REC-DOM-Level-1/>
- 🇺🇸 *Document Object Model (DOM) Level 2 Core Specification*
Recommandation, version 1.0, du 13-11-2000
Document sur <http://www.w3.org/TR/DOM-Level-2-Core/>
- 🇺🇸 *Document Object Model (DOM) Level 2 Events Specification*
Recommandation, version 1.0, du 13-11-2000
Document sur <http://www.w3.org/TR/DOM-Level-2-Events/>
- 🇺🇸 *Document Object Model (DOM) Level 2 HTML Specification*
Recommandation, version 1.0, du 09-10-2003
Document sur <http://www.w3.org/TR/DOM-Level-2-HTML/>
- 🇺🇸 *Document Object Model (DOM) Level 2 Style Specification*
Recommandation, version 1.0, du 13-11-2000
Document sur <http://www.w3.org/TR/DOM-Level-2-Style/>
- 🇺🇸 *Document Object Model (DOM) Level 2 Traversal and Range Specification*


Recommandation, version 1.0, du 13-11-2000
Document sur <http://www.w3.org/TR/DOM-Level-2-Traversal-Range/>


 *Document Object Model (DOM) Level 2 Views Specification*
Recommandation, version 1.0, du 13-11-2000
Document sur <http://www.w3.org/TR/DOM-Level-2-Views/>


 *Document Object Model Level 3 Core Specification*
Recommandation, version 1.0, du 07-04-2004
Document sur <http://www.w3.org/TR/DOM-Level-3-Core/>


 *Document Object Model Level 3 Validation Specification*
Recommandation, version 1.0, du 27-01-2004
Document sur <http://www.w3.org/TR/DOM-Level-3-Val/>

 *Document Object Model Level 3 Abstract Schemas Specification*
Note, version 1.0, du 25-07-2002
Document sur <http://www.w3.org/TR/2002/NOTE-DOM-Level-3-AS-20020725/>

 *Document Object Model Level 3 Events Specification*
Projet en cours, version 20090908, du 08-09-2009
Document sur <http://www.w3.org/TR/DOM-Level-3-Events/>

 *Document Object Model Level 3 Views and Formatting Specification*
Projet en cours, version 1.0, du 15-11-2000
Document sur <http://www.w3.org/TR/DOM-Level-3-Views/>

 *Document Object Model Level 3 XPath Specification*
Recommandation Candidate, version 1.0, du 31-03-2003
Document sur <http://www.w3.org/TR/DOM-Level-3-XPath/>

 *Document Object Model Requirements*
Projet en cours, version 20010419, du 19-04-2001
Document sur <http://www.w3.org/TR/DOM-Requirements/>

TIRÈME SARL

SAX, Simple API for XML

Rédaction : Pierre Attar

Recommandation(s) liée(s) : [DOM](#) - [XSLT](#)

[SAX](#) est une recommandation de fait (issue des listes de discussion [Monthly Archives for xml-dev]), qui permet d'interfacer des programmes avec un "*reader XML*" fournissant, au travers d'un [API](#), un séquence d'événements issus des objets rencontrés dans le fichier [XML](#).

Ces événements sont de type : début d'élément, attribut, fin d'élément. [SAX](#) repose donc, à la différence de [DOM](#) et de [XSLT](#) qui présupposent un arbre d'objets typés et *valués*, sur une vision séquentielle et événementielle de l'information.

Objectifs

L'objectif de [SAX](#) est de fournir une [API](#) simple, permettant d'accéder au contenu d'un document [XML](#). Pour ce faire, la liste de discussion `xml-dev`, a défini une spécification, basée sur une lecture séquentielle de ces documents, permettant d'agir sur chaque événement rencontré lors de cette lecture.

Principes


Dans sa version 2, [SAX](#) propose un ensemble d'interfaces permettant d'accéder à tous les événements disponibles lors de la lecture séquentielle d'un document [XML](#). On y trouve des notions de début et de fin de document ou d'élément, d'attributs et de valeurs d'attribut, de contenus textuels, d'entités, de *processing instructions*, etc. On y trouve aussi un certain nombre de fonctionnalités permettant de contrôler le comportement du "*reader*".

Par nature événementiel, le gros intérêt d'une telle spécification est d'être adapté au traitement de gros documents [XML](#). En effet, les approches [DOM](#), voire [XSLT](#), reposent sur la manipulation d'arbres et, du coup, nécessitent toutes deux de charger tout le document en mémoire, avant de pouvoir effectuer quelle que manipulation que ce soit.

La communauté des développeurs et des éditeurs de logiciels ne s'y est pas trompée, qui utilise alternativement [DOM](#) ou [SAX](#), en fonction du type de manipulation à réaliser.

D'un point de vue technique, un regret est que les acteurs [SAX](#) ne fasse pas la différence entre une spécification et une implémentation d'une spécification. En effet, sur les sites dédiés à [SAX](#), on ne trouve pas de spécification en tant que telle : on trouve des interfaces [SAX](#) pour Java, pour Python, pour... Le risque est alors que les spécifications évoluent en parallèle et qu'il existe autant de [SAX](#) que d'implémentations.

Recommandations(s)

-  *SAX: The Simple API for XML*
Recommandation, version 2.0.1, du 29-01-2002
Document sur <http://sourceforge.net/projects/sax/>

TIRÈME SARL

XProc, XProc

« An XML Pipeline specifies a sequence of operations to be performed on one or more XML documents. Pipelines generally accept one or more XML documents as input and produce one or more XML documents as output. Pipelines are made up of simple steps which perform atomic operations on XML documents and constructs similar to conditionals, loops and exception handlers which control which steps are executed. »

Source. *XProc: An XML Pipeline Language, version 20100309*

Objectifs

Recommandations(s)

 *XProc: An XML Pipeline Language*

Proposition de recommandation, version 20100309, du 09-03-2010

Document sur <http://www.w3.org/TR/xproc/>

TIRÈME SARL

XSLT, XSL Transformations

Rédaction : Pierre Attar

Recommandation(s) liée(s) : [DOM](#) - [SAX](#) - [XSL](#)

Cette spécification définit la syntaxe et la sémantique d'un langage de transformation de documents XML en autres documents XML.

XSLT fait partie de la recommandation XSL, un langage généralisé d'expression de feuilles de styles. En ce sens, XSLT n'est pas un langage de programmation générique d'accès aux documents XML comme DOM ou SAX peuvent l'être. C'est un langage spécialisé pour toutes les questions de transformations nécessaires à la présentation d'un document, sur papier, sur le Web...

Cependant, la spécification permet d'implémenter XSLT de façon indépendante de tout processus de formatage, basé sur les *formatting objects* de XSL ou sur HTML. Du coup, il existe une zone de recouvrement entre XSLT et les API DOM ou SAX.

Objectifs

La norme DSSSL de l'ISO définit les concepts nécessaires à la présentation d'un document : il est nécessaire de transformer l'arbre, avant de pouvoir le formater. Les transformations nécessaires sont, par exemple, le fait de prendre l'auteur d'une lettre, défini en attribut de lettre, pour générer une signature, en fin de lettre.

De façon plus générique, les transformations sont de l'ordre de :

- . la réorganisation d'informations ;
- . la duplication d'informations ;
- . la génération d'informations, fixe ou calculée ;
- . la suppression d'informations.

L'objectif d'XSLT, dans le cadre du Web, est d'être la recommandation de transformation liée à des processus de formatage. Cependant, puisqu'il transforme un document XML en un autre document XML (un arbre d'objets de formatage), XSLT peut être utilisé pour beaucoup d'autres besoins de transformation et, cela, même si, dans la volonté de ses concepteurs, il est spécialisé dans la présentation.

Principes

L'objectif de XSLT est de construire, à partir d'un document XML, un nouveau document XML ne contenant que des instructions de formatage (seule la spécification 1.1 permettra de construire plusieurs documents). Les arbres issus d'une transformation sont des arbres

HTML ou des arbres d'objets de formatage, selon la recommandation XSL. Ces arbres ne contiennent plus de sémantique de données, il ne contiennent que de la sémantique de présentation.

Pour ce faire, le langage est basé sur des sélections d'objets typés (`xsl:apply-templates`) auxquels on applique une transformation (`xsl:template`); la transformation par défaut copie la source vers la cible.

Plus précisément, c'est à partir de la racine (/) du document d'entrée que sont effectuées des sélections. Par exemple, si le document d'entrée est un ensemble de lettres commerciales, la transformation suivante fournira une liste de tous les titres et auteurs des lettres contenues :

```
<xsl:stylesheet ...>
  <xsl:template match="/">
    <!-- faire un traitement sur les objets typés lettre -->
    <xsl:apply-templates select="lettre"/>
  </xsl:template>
  <!-- définition du traitement des lettres -->
  <xsl:template match="lettre">
    <!-- générer un paragraphe -->
    <fo:block>
      <!-- prendre le contenu caractères de l'objet titre
      contenu dans la lettre -->
      <xsl:value-of select="titre"/>
      <!-- ajouter du texte de liaisons -->
      <xsl:text> ; auteur : <xsl:text>
      <!-- prendre le contenu caractères de l'objet auteur
      contenu dans la lettre -->
      <xsl:value-of select="auteur"/>
    </fo:block>
  </xsl:template>
</xsl:stylesheet>
```

La sélection d'objets typés est réalisée en utilisant le langage XPath, initialement conçu pour ce seul besoin.

Le vocabulaire d'XSLT permet de :

- créer de nouveaux éléments dans la cible (`xsl:element`, `xsl:attribute`, `xsl:attribute-set`);
- générer du texte (`xsl:text`) ou des sous-arbres issus de la source (`xsl:copy`);
- effectuer des tests (`xsl:if`) et, plus généralement, des traitements conditionnés par des choix (`xsl:choose`);
- déclarer des variables (`xsl:variable` et `xsl:param`). À noter que les notions de variables globales n'existent pas, du fait de la volonté de définir un langage "context-free" où l'on n'a pas besoin de connaître l'état de toute la spécification pour appliquer un changement local ;
- se déclarer des listes d'objets identifiés (`xsl:key`) qui seront ensuite utilisables dans les transformations (`key`);
- réaliser des boucles (`xsl:for-each`), triées si nécessaires (`xsl:sort`);
- modulariser les feuilles de styles en différents fichiers (`xsl:import` et `xsl:include`) ou en modules de traitement factorisés (`xsl:call-template`, `xsl:with-param`);
- contrôler le type d'écriture de la sortie (`xsl:output`, `xsl:strip-space`, `xsl:preserve-space`);
- ...

À ce vocabulaire sont ajoutés, outre les fonctions de XPath, un certain nombre de nouvelles fonctions permettant, d'une part, de mieux sélectionner et, d'autre part, de mieux contrôler le contenu de l'arbre cible.

Pour conclure, ce qui manque de façon la plus cruciale à **XSLT** est ce qui sera ajouté dans la version 1.1 de la recommandation : la possibilité de créer plusieurs arbres à partir d'une même source et la prise en compte de **XML Base** pour mieux contrôler les **URI** définies dans les cibles.

Dans la pratique

XSLT est un langage *a priori* difficile d'accès : son écriture sous forme **XML** le rend "verbeux" et les méthodes d'écriture par sélection d'objets typés nécessitent de revoir beaucoup de méthodes de programmation. En revanche, une fois cet apprentissage réalisé, ce langage est extrêmement puissant... si puissant que beaucoup l'utilisent même uniquement comme langage de transformation pur, en dehors de toute problématique de présentation.

Par exemple, écrire un programme de validation du fait que toutes les références bibliographiques d'un livre existent n'est pas, avec un langage de programmation usuel, quelque chose de trivial. Avec **XSLT**, l'opération est aisée à écrire :

```
<xsl:stylesheet ...>
  <!-- les items de bibliographie (bib-item) ont un attribut nom
d'identification -->
  <xsl:key name="itemDeBiblio" match="bib-item" use="./@nom"/>
  <xsl:template match="/">
    <!-- BIBREF réalise la référence, au travers d'un attribut
lien -->
    <xsl:for-each select="//BIBREF">
      <xsl:choose>
        <!-- le test permet de savoir si le résultat de
recherche dans la clé existe et correspond à la
valeur de l'attribut lien -->
        <xsl:when test="key('itemDeBiblio',@lien)/
@nom=@lien">
          <xsl:text>Tout va bien</xsl:text>
        </xsl:when>
        <xsl:otherwise>
          <xsl:text>Tout va mal</xsl:text>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

Par ailleurs, le langage **XSLT** n'est pas fermé et un mécanisme permet à chaque outil de se rajouter ses propres extensions. Aujourd'hui, ce mécanisme souffre de formalisation et la recommandation 1.1 devrait permettre de mieux le maîtriser.

Quoi qu'il en soit et du fait des extensions, il est nécessaire de choisir l'environnement applicatif dans lequel on se situe, avant d'utiliser des extensions. En effet, si, par exemple, on utilise des extensions dans une feuille de styles chargée dynamiquement par un logiciel de consultation du Web, on se contraint très rapidement à rendre dépendantes les applications du type de logiciel de consultation. Il est donc nécessaire de différencier les transformations génériques, utilisables par n'importe quel outil de traitement **XSLT** de celles que l'on décide, liées à un outil particulier, et ses extensions.

Recommandations(s)

■ *Transformations XSL (XSLT)*

Recommandation, version 1.0, du 16-11-1999

Document sur <http://xmlfr.org/w3c/TR/xslt/>

■ *XSL Transformations*


Recommandation, version 2.0, du 23-01-2007

Document sur <http://www.w3.org/TR/xslt20/>

 *XSLT Requirements*

Projet en cours, version 2.0, du 14-02-2001

Document sur <http://www.w3.org/TR/xslt20req>

 *XSLT 2.0 and XQuery 1.0 Serialization*

Recommandation, version 20072301, du 23-01-2007

Document sur <http://www.w3.org/TR/xslt-xquery-serialization/>

DSSSL, Sémantique de présentation de documents et langage de spécifications

Recommandation(s) liée(s) : [FOSI - XSL](#)

« A key feature of generalized markup is that the formatting and other processing information associated with the document is separate from the generic tags embedded in it.

In any generalized markup scheme, there is a method for associating processing specifications with the SGML markup. This method of association allows the information to be attached to specific instances of elements as well as to general classes of element types. The primary goal of DSSSL is to provide a standardized framework and methods for associating processing information with the markup of SGML documents or portions of documents.

DSSSL is intended for use with documents structured as a hierarchy of elements. For the purpose of describing in detail the concepts of DSSSL in the subsequent clauses of this International Standard, SGML terminology is used.


DSSSL enables formatting and other processing specifications to be associated with these elements to produce a formatted document for presentation. For example, a designer may wish to specify that all chapters begin on a new recto page and that all tables begin with a page-wide rule to be positioned only at the top or bottom of the page. During the DSSSL transformation process, formatting information may be added to the result of the transformation. This information may be represented as SGML attributes. These, in turn, may be used by the style language to create formatting characteristics with specific values. »

Source. Information technology - Processing languages - Document Style Semantics and Specification Language (DSSSL), version 10179:1996

Objectifs

Information descriptive non finalisée ; n'hésitez pas à nous contacter pour rédiger et/ou maintenir cette information à jour.

Recommandations(s)

 Information technology - Processing languages - Document Style Semantics and Specification Language (DSSSL)

Standard ISO, version 10179:1996, du 04-04-1996

Document sur [http://www.iso.ch/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?
csnumber=18196](http://www.iso.ch/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=18196)

TIRÈME SARL

XSL, Langage de feuilles de styles extensible

Rédaction : Pierre Attar

Recommandation(s) liée(s) : [DSSSL](#) - [FOSI](#) - [XSLT](#)

[XSL](#) est un langage d'expression de feuilles de styles définissant, d'une part, un langage de transformation de documents [XML](#) ([XSLT](#)) et, d'autre part, un langage permettant de spécifier une sémantique de formatage. C'est grâce à [XSL](#) que peuvent être réalisées des présentations de document [XML](#), que celles-ci soient liées au papier, à [WML](#), au Web, ou à tout autre support électronique.

D'aucuns diront que la présentation sur papier n'est pas de même nature que celle sur le Web... et ils auront raison. En revanche, il est intéressant d'utiliser un même langage de spécification pour définir les représentation nécessaires aux deux environnements. Un paragraphe est toujours un paragraphe, même s'il doit être présenté de façon différente.

Du coup, l'objectif de [XSL](#) est de fournir un langage de spécification de formatage seulement, et non pas un moteur de composition.

Objectifs

L'objectif de [XSL](#) est de définir un langage de présentation de document, indépendant des systèmes et des logiciels. Autant la validité de ce concept est prouvée sur Internet, autant, dès lors qu'il s'agit de présenter des documents sur papier, il n'existe pas ce type de méthode en dehors de la sphère [SGML](#).

L'intérêt ? En environnement ouvert, comme Internet, il devient possible d'exprimer des intentions de présentation ; charge au logiciel de restitution d'utiliser au mieux cette information. De façon plus générale, surtout pour l'impression, l'intérêt est de pouvoir définir de façon formelle une spécification de présentation et d'utiliser, ensuite, des logiciels de présentation, de façon indifférenciée.

Si l'objectif n'était *que* Internet, la spécification ne serait pas aussi importante ! C'est parce que l'objectif est de "sortir" de ces pages Internet construites à partir de tableaux et, aussi, parce que l'objectif est le formatage papier que la recommandation est gigantesque : modéliser les acquis de présentation hérités de plus de 200 années de culture papier nécessite un ensemble d'objets de formatage important. La taille de la spécification est également liée à un objectif d'internationalisation : elle doit permettre d'écrire en anglais et en français ; elle doit aussi permettre d'écrire en arabe (de droite à gauche) et en japonais (de haut en bas).

Principes

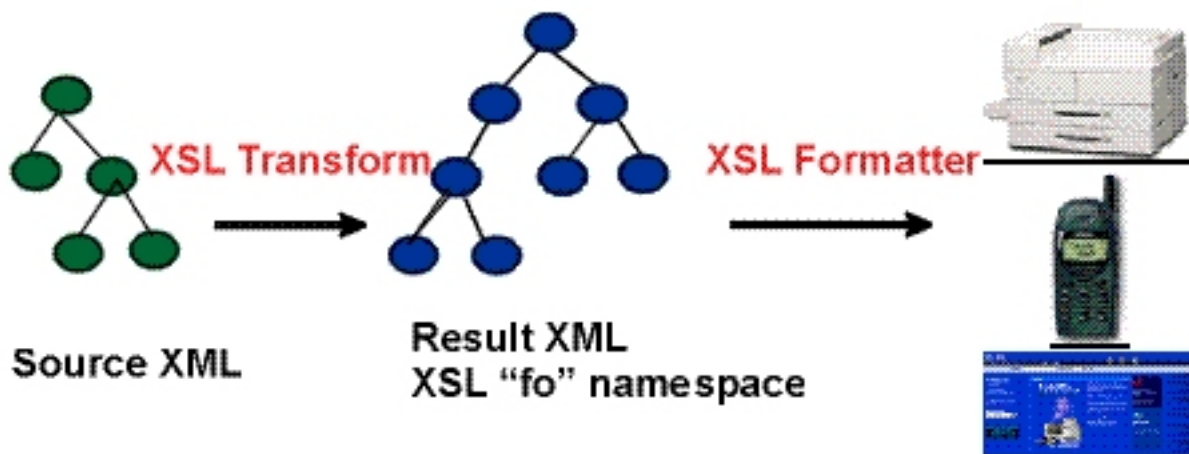
Une feuille de styles **XSL** est définie comme un document **XML** contenant des spécifications de transformation et de formatage d'objets. Elle permet de transformer un document **XML** d'entrée en un autre document **XML**, dont les éléments de structure sont tous liés à des éléments typographiques représentant des intentions de formatage : des pages, des fenêtres, des paragraphes, des listes, etc.

Le document **XML** résultant de la transformation doit ensuite être pris en charge par un outil de formatage, qui créera une version papier, PDF, DVI, RTF, **HTML**, Wap, ou tout autre format. La spécification différencie donc deux processus : la transformation d'arbre (*tree transformation*) et le formatage (*formatting*).

Le document présenté pouvant être structurellement très différent du document **XML** d'origine, tout le pouvoir de transformation d'**XSLT** doit pouvoir être utilisé dans la première partie du processus, pour ajouter, par exemple, des tables de matières ou encore filtrer et réordonner des informations.

Figure. Principe de fonctionnement des processeurs XSL
(issu de la spécification "Extensible Stylesheet Language")

XSL Two Processes: Transformation & Formatting



Result XML is the result of XSLT processing. Syntactically, it is XML elements and attributes.

Pour permettre de réaliser une impression papier de qualité et automatisée, le modèle de présentation introduit des notions d'*aires*, dans lesquelles se coulent des flux de texte. Ainsi, il est possible de composer un document bilingue, où chaque paragraphe est écrit en français et en italien, en coulant le flux français dans une *aire* représentant la colonne de gauche, tandis que l'autre flux se coule dans l'*aire* représentant la colonne de droite.

D'un point de vue typographique, **XSL** définit les objets de formatage nécessaires aux documents, fenêtres, pages, hyperliens, paragraphes, listes, tableaux, images, caractères, etc.

Où en est-on ?

La recommandation a longtemps erré, avant de décider de reprendre comme base, afin de l'industrialiser, la norme [DSSSL](#). Aujourd'hui la recommandation est votée.

Il est intéressant de remarquer que beaucoup de personnes trouvent cette spécification trop lourde et, donc, inapplicable. Ceci est confirmé par le fait qu'il y a, dans cette étape de maturité de la recommandation, peu de logiciels qui implémentent l'existant.

Pourquoi ? Certainement car, d'un part, beaucoup d'acteurs de Web ne reconnaissent pas la nécessité d'une impression sur papier de qualité. Mais, d'autre part et surtout, car il n'existe que peu de logiciels aujourd'hui capables de réaliser, sur des pages un tant soit peu complexes, des formatages *automatiques* de qualité. En effet, tous les logiciels de PAO, même s'ils réalisent certains traitements automatisés (comme pour l'application de feuilles de styles simples), laissent toujours, en dernier ressort, l'utilisateur décider et agir sur la composition finale des pages. Ceci n'est plus possible si l'on souhaite que les documents soient formatés de façon complètement automatisée et, du coup, [XML](#) et les documents structurés nécessitent de nouvelles classes d'outils de composition... Toute la question sera ensuite de savoir *qui* décidera de se lancer sur ce marché.

Exemple

La feuille de styles ci-après, présuppose, dans un document, un élément `date`, contenant les éléments `jour`, `mois` et `an`.

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:fo="http://www.w3.org/1999/XSL/Format">
  ...
  <xsl:template match="date">
    <fo:block text-align="centered" font-size="10pt" line-
      height="12pt" space-before.optimum="3pt" font-style="italic">
      <xsl:text>Paris, le </xsl:text>
      <xsl:value-of select="jour"/>
      <xsl:text>-</xsl:text>
      <xsl:value-of select="mois"/>
      <xsl:text>-</xsl:text>
      <xsl:value-of select="an"/>
    </fo:block>
  </xsl:template>
  ...
</xsl:stylesheet>
```

Cette feuille de styles, appliquée au document suivant :

```
<doc>
  ...
  <date>
    <jour>11</jour>
    <mois>06</mois>
    <an>2000</an>
  </date>
  ...
</doc>
```

... générera le paragraphe ci-après, dans un arbre d'objets de formatage.

```
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
  ...
  <fo:block text-align="centered" font-size="10pt"
    line-height="12pt" space-before.optimum="3pt" font-
    style="italic">Paris, le 11-06-2000</fo:block>
  ...
</fo:root>
```

Recommandations(s)

- ■ *Le langage extensible de feuille de style (XSL)*
Recommandation, version 1.0, du 15-10-2001
Document sur <http://www.yoyodesign.org/doc/w3c/xsl1/Overview.html>
- 🇺🇸 *Extensible Stylesheet Language*
Recommandation, version 1.0, du 15-10-2001
Document sur <http://www.w3.org/TR/xsl/>
- 🇺🇸 *Extensible Stylesheet Language (XSL) Requirements Version 2.0*
Projet en cours, version 2.0, du 26-03-2008
Document sur <http://www.w3.org/TR/xslfo20-req/>
- 🇺🇸 *Extensible Stylesheet Language (XSL) Version 1.1 Requirements*
Projet en cours, version 1.1, du 17-12-2003
Document sur <http://www.w3.org/TR/xsl11-req/>
- 🇺🇸 *XSL Requirements Summary*
Projet en cours, version 19980511, du 11-05-1998
Document sur <http://www.w3.org/TR/WD-XSLReq>

TIRÈME SARL

XML Query, XML Query Language

Rédaction : Pierre Attar

Recommandation(s) liée(s) : XPath

« *XML Query operates on the abstract, logical structure of an XML document, rather than its surface syntax. This logical structure is known as the data model, which is defined in the XML Query 1.0 and XPath 2.0 Data Model document.*

XML Query Version 1.0 is an extension of XPath Version 2.0. Any expression that is syntactically valid and executes successfully in both XPath 2.0 and XML Query 1.0 will return the same result in both languages. Since these languages are so closely related, their grammars and language descriptions are generated from a common source to ensure consistency, and the editors of these specifications work together closely.

XML Query also depends on and is closely related to the following specifications:

The XML Query data model defines the information in an XML document that is available to an XML Query processor. The data model is defined in XML Query 1.0 and XPath 2.0 Data Model.

The static and dynamic semantics of XML Query are formally defined in XML Query 1.0 and XPath 2.0 Formal Semantics. This document is useful for implementors and others who require a rigorous definition of XML Query.

The type system of XML Query is based on Schema.

The default library of functions and operators supported by XML Query is defined in XML Query 1.0 and XPath 2.0 Functions and Operators...

One requirement in XML Query 1.0 Requirements is that an XML query language have both a human-readable syntax and an XML-based syntax. The XML-based syntax for XML Query is described in XML Query: An XML Query Language 1.0.

The current edition of XML Query 1.0. has not incorporated recent language changes; it will be made consistent with this document in its next edition.






This document specifies a grammar for XML Query, using the same Basic EBNF notation used in XML, except that grammar symbols always have initial capital letters. Unless otherwise noted, whitespace is not significant in the grammar. Grammar productions are introduced together with the features that they describe, and a complete grammar is also presented in the appendix (A XQuery Grammar). »


Source. XQuery: An XML Query Language, version 1.0

Objectifs

Information descriptive non finalisée ; n'hésitez pas à nous contacter pour rédiger et/ou maintenir cette information à jour.

Recommandations(s)

-  *XQuery: An XML Query Language*
 Recommandation, version 1.0, du 23-01-2007
 Document sur <http://www.w3.org/TR/xquery/>
-  *XQuery 1.0 and XPath 2.0 Data Model (XDM)*
 Recommandation, version 20070123, du 23-01-2007
 Document sur <http://www.w3.org/TR/xpath-datamodel/>
-  *XML Query Requirements*
 Projet en cours, version 1.1, du 15-12-2009
 Document sur <http://www.w3.org/TR/xquery-requirements>
-  *XQuery Update Facility*
 Recommandation Candidate, version 1.0, du 14-03-2008
 Document sur <http://www.w3.org/TR/xquery-update-10/>
-  *XQuery Update Facility Use Cases*
 Recommandation Candidate, version 1.0, du 14-03-2008
 Document sur <http://www.w3.org/TR/xqupdateusecases/>
-  *XQuery Update Facility Requirements*
 Recommandation Candidate, version 1.0, du 14-03-2008
 Document sur <http://www.w3.org/TR/xquery-update-requirements/>
-  *XML Syntax for XQuery 1.0 (XQueryX)*
 Recommandation, version 1.0, du 23-01-2007
 Document sur <http://www.w3.org/TR/xqueryx>
-  *XML Query (XQuery) 1.1 Use Cases*
 Projet en cours, version 1.1, du 11-07-2008
 Document sur <http://www.w3.org/TR/xquery-11-use-cases/>
-  *XML Query (XQuery) Use Cases*
 Projet en cours, version 20070323, du 27-03-2008
 Document sur <http://www.w3.org/TR/xquery-use-cases>
-  *XQuery 1.0 and XPath 2.0 Formal Semantics*
 Recommandation, version 20070123, du 23-01-2007
 Document sur <http://www.w3.org/TR/xquery-semantics>
-  *XQuery 1.0 and XPath 2.0 Functions and Operators*
 Recommandation, version 1.0, du 23-01-2007
 Document sur <http://www.w3.org/TR/xpath-functions/>
-  *XQuery and XPath Full Text 1.0 Requirements*
 Projet en cours, version 20080516, du 16-05-2008
 Document sur <http://www.w3.org/TR/xpath-full-text-10-requirements/>
-  *XQuery and XPath Full Text 1.0*
 Recommandation Candidate, version 1.0, du 28-01-2010
 Document sur <http://www.w3.org/TR/xpath-full-text-10/>
-  *XQuery and XPath Full Text 1.0 Use Cases*
 Projet en cours, version 1.0, du 28-01-2010
 Document sur <http://www.w3.org/TR/xmlquery-full-text-use-cases>

 *XSLT 2.0 and XQuery 1.0 Serialization*
Recommandation, version 20072301, du 23-01-2007
Document sur <http://www.w3.org/TR/xslt-xquery-serialization/>