Mise en œuvre de XML sous APL.

par Bernard Mailhol

1 XML et son contexte
1.1 Ce qu'est XML
1.2 A quoi ça sert ?
1.3 Les DTD
1.4 Pourquoi maintenant ?

2 XML et le monde Internet
2.1 Exemple d'un tableau en HTML
2.2 Exemple de ce même tableau en XML
2.3 Standards complémentaires
2.3.1 Les feuilles de style XSL
2.3.2 SMIL, Synchronized Multimedia Integration Language
2.3.3 Ressource Description Framework

3 Usage de XML en APL
3.1 Représentation en APL
3.2 Outils en APL
3.2.1 Construire des éléments XML
3.2.2 Afficher pour contrôle
3.2.3 Analyser une description XML
3.2.4 Transformer une structure en description XML, exporter.
3.2.5 Modifier dans les structures
3.3 L'exemple du tableau
3.3.1 limitations

4 Sur la Toile, Download et liens

1 XML et son contexte

Depuis quelques mois, on parle beaucoup de XML. Qu'est-ce ? Ceci nous concerne t-il ?.
Pourquoi apprendre encore un système ? Sera-t-il aussi fugace que les autres ?
On peut essayer de découvrir l'intérêt de ce système XML, de comprendre pourquoi il faut s'en imprégner, pour le mettre en œuvre dès que possible.

Très sommairement, XML est un format de fichiers balisés
1 - Standardisés par un organisme non commercial,
2 - Pouvant être créés et lus par un programme que nous pouvons écrire,
3 - être traités par de nombreux autres programmes.
4 - être réutilisables 10 ans après.
Vraiment, on peut s'y intéresser.

1.1 Ce qu'est XML

XML est un langage de description et d'échange de documents structurés.
Il permet de représenter une information simple ou complexe, d'une façon non ambiguë, dans

un fichier composé de caractères ascii imprimables, donc lisible directement par un humain. Lire ... bien sûr, mais ce n'est pas très clair. En voici un exemple; il s'agit de transférer une facture composée exclusivement d'une quantité, et d'un prix total. Pas d'identification, ni rien d'autre .. Mais ce n'est heureusement qu'un exemple.
Simplement, on peut écrire

```
25  320
```

.. mais ce fichier n'est compréhensible qu'en liaison avec une définition très précise, et ne contient aucune identification, sinon le nom du fichier, la date de création du fichier. On peut ensuite essayer la débrouille.
Dans une forme XML, le fichier est plus complexe, mais dispose d'autres propriétés.

```
<?xml version='1.0' ?>
<facture>
  <nombre>25</nombre>
  <prix>320</prix>
</facture>
```

Ce fichier est un fichier de type balisé car les données qu'il contient sont délimitées par des balises. (<nombre> par exemple).
Vous avez l'essentiel de XML. Maintenant, quelques détails.
XML, pour eXtensible Markup Language a été créé par le World Wide Web Consortium (W3C), qui définit aussi le langage HTML, et la façon de s'en servir. XML a été défini en 1997. Le W3C n'émettant que des recommandations. XML est ainsi défini par la recommandation REC-xml-19980210.

1.2 A quoi ça sert ?

Un peu sommaire, cet exemple !
Si on le complète un peu, on peut représenter une vraie facture, avec une définition de client, de fournisseur, plusieurs lignes de commandes, chacune composée de code/désignation/quantité/prix unitaire/taux de TVA/remise ... Enfin, les totaux et modes de règlement. Tout ceci peut être représenté sans ambiguïté.
Une telle définition qui, comme on l'a déjà dit est complètement visible, ne cache rien (donc pas de virus, ni de subtilités cachées vite dépassées par d'autres subtilités cachées), et est comprise aujourd'hui comme dans quelques années par vos programmes, ou des programmes standard.
On constate aujourd'hui l'apparition de produits de plus en plus nombreux reposant sur XML.

Concrètement

•Cette forme de fichiers est de plus en plus utilisée dans les procédures d'EDI.
Quoi de plus facile de transmettre et de recevoir un tel fichier à définition universelle, de le traiter ensuite : ce fichier peut n'être composé que de caractères ascii-7, dont la transmission ne pose aucun problème sur aucun système de transmission; les caractères spéciaux (fin de ligne, tabulation) sont ignorés.
Il s'agit toutefois de définir correctement ce qu'il contient. C'est le rôle de la DTD (p. 101), associée à un fichier XML.

•Cette définition peut bien sûr définir des textes à imprimer. Les balises utilisées peuvent alors être des délimitations de chapitres, sections, paragraphes ... Il existe d'ailleurs quelques DTD publiques utilisables pour ces définitions.

•XML peut convenir pour traiter des affichages Web, en remplacement d'HTML (HTML est le langage d'affichage des pages Web). Cet usage pose deux grands problèmes :

a - Les balises présentes dans un fichier XML ne sont pas connues des fureteurs du marché. Il n'est pas question de définir une DTD "Web" à laquelle devraient se plier tous les développeurs de XML pour affichage par Web.
La logique de XML est que le fichier contient les informations utiles, et non des informations de présentation. Une comparaison sommaire entre les formats XML et HTML est donnée plus loin (p. 103).

b - L'ordre de présentation des informations dans un fichier XML est un ordre logique par rapport à une définition, par rapport à un traitement, alors que l'ordre des informations présentes dans une page HTML respecte les contraintes de visualisation et de mise en page.
Il faut pouvoir réorganiser les informations avant de pouvoir les afficher.
Ces contraintes ont conduit à l'élaboration d'un troisième langage, XSL (p. 106), ayant un double but :

a - Réorganiser l'arbre contenant les informations utiles de XML, pour le présenter correctement en HTML

b - Définir la présentation associée à chaque balise utilisée en XML. (Une feuille de style)

Une fois cette définition XSL établie pour un type de pages, on peut alors publier en XML les données dont on dispose. Quand on a créé des pages HTML dynamiquement, on sait que la quantité de commandes liées à la présentation est énorme, par rapport à la quantité d'information utiles.
Si les données de structuration d'un fichier XML sont importantes, leur préparation se fait au moyen de primitives standard (on en verra un exemple plus loin (p. 107)).
On peut alors publier facilement des données, en les séparant absolument de la charte graphique retenue.

•Les visualisations de graphiques ne sont pas définies en HTML. Pour afficher un graphique, on a alors le choix entre trois solutions :
a - Placer ce graphique dans un fichier d'image, éventuellement animée. Cette solution pose problème si il faut régérérer l'image gif à chaque appel
b - Appeler un programme écrit en Java (cet appel est défini dans la recommandation HTML 3.2), qui se chargera de ce calcul.
c - Utiliser un plug-in, programme disponible pendant un temps limité sur un ensemble réduit de systèmes (mais pensez à la consultation sur
un Web-TV. Si le fureteur intégré ne reconnaît pas ce plug-in, votre page ne sera pas lisible par les possesseurs de ce Web-TV).
Le W3C définit à son tour une DTD (p. 101) permettant de prendre en charge ces graphiques. Dans quelques années, tous les fureteurs, y compris ceux inclus dans votre poste de Télévision, comprendront ces fichiers XML respectant cette DTD spécifique.
On peut ainsi utiliser des DTD personnelles pour créer des fichiers XML; il faut les associer à des définitions de présentations HTML.
L'ensemble de ces fichiers à vocations multiples seront régis par une syntaxe commune, et pourront être pris en charge par votre application, en toute légalité et sans brevets à rémunérer.

1.3 Les DTD

Le but n'est pas de décrire les DTD, mais seulement de montrer ce qu'elles peuvent être.

Disons seulement qu'un fichier XML, après contrôle, peut être :

1 - bien formé ce qui signifie que, globalement, sa syntaxe respecte les formes de XML, mais sans contrôle du nom des balises, de leur organisation, de leur contenu.
2 - valide, pour indiquer que ce fichier est non seulement bien formé, mais que ses balises répondent aux exigences de la DTD.
Voici le fichier précédent (p. 97) agrémenté d'une définition de DTD. Tout d'abord le fichier

facture.xml

```
<?xml version='1.0' encoding='ISO-8859-1' ?>
<!DOCTYPE facture SYSTEM "facture.dtd" >
<facture>
  <nombre>25</nombre>
  <prix>320</prix>
</facture>
```

... puis la DTD correspondante, le fichier facture.dtd, cité dans le fichier précédent.

```
<?xml version='1.0' encoding='ISO-8859-1' ?>
<!-- D,but de la DTD -->
<!ELEMENT facture  (nombre prix) >
<!ELEMENT nombre   (#PCDATA) >
<!ELEMENT prix     (#PCDATA) >
<!-- Fin de la DTD -->
```

On peut lire cette DTD :

•Le fichier XML peut contenir l'élément facture
•Une facture contient un élément nombre et un élément prix, dans cet ordre
•L'élément nombre contient un texte quelconque
•L'élément prix contient un texte quelconque

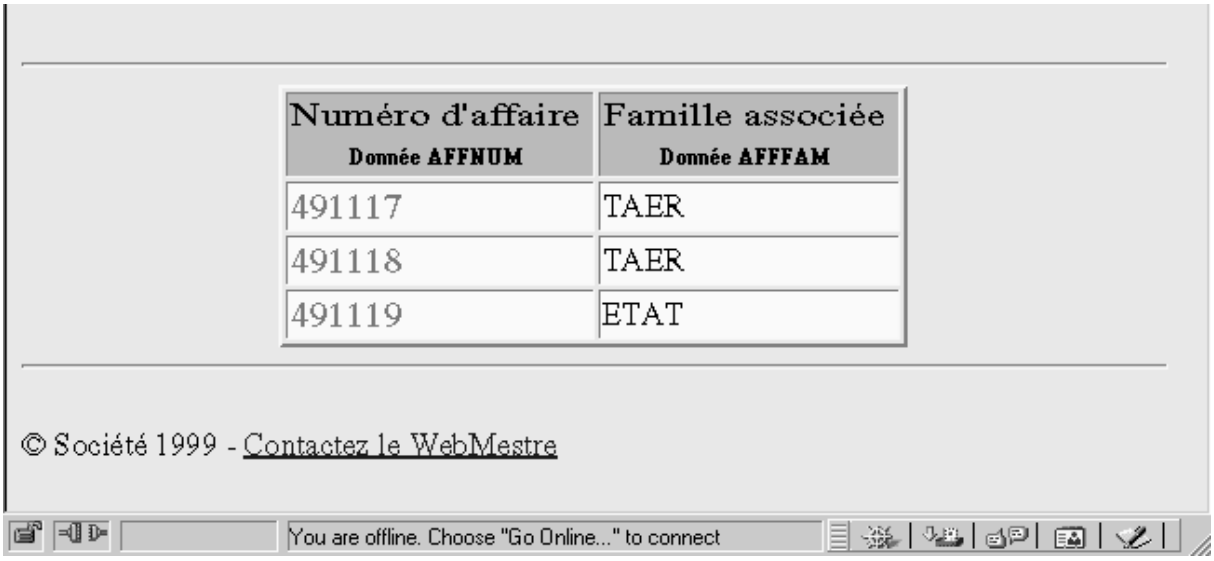
Le fichier facture.xml étant valide, on sait qu'il satisfait aux propriétés indiquées ci-dessus, en plus des propriétés générales des fichiers XML.
Evidemment, cette DTD suit les règles de SGML.

1.4 Pourquoi maintenant ?

XML est une simplification de SGML (norme ISO 8879-1986 de 1996). Bien que perdant de la finesse par rapport à SGML, cette recommandation du W3C permet des usages faciles, à la portée de nombreuses applications.
Nos applications vont aussi utiliser ces formats - simplifiés - de fichiers balisés. Il est temps de savoir utiliser XML en APL.

2 XML et le monde Internet

Ce papier ne cherche ici qu'à donner un exemple d'usage de fichiers XML, dans un contexte d'affichage HTML.
Voici l'affichage d'un tableau via un fureteur HTML :



2.1 Exemple d'un tableau en HTML

Voici la programmation HTML ayant créé cet affichage :

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<html>
<head>
<title>Extraction de donn&eacute;es d'affaire</title>
</head>
<body bgcolor="#E8E8E8">
<center>
<h1>Extraction de donn&eacute;es d'affaire</h1>
</center>
<hr>
<center><table border=2>
<tr bgcolor="#91C4FF">
   <th>Num&eacute;ro d'affaire
   <br><font size="1">Donn&eacute;e AFFNUM</font>
   </th>
   <th>Famille associ&eacute;e
   <br><font size="1">Donn&eacute;e AFFFAM</font>
   </th>
</tr>
<tr bgcolor="#FFFFFF">
   <td><font size="+1"color="FF4444">491117</font></td>
   <td>TAER</td>
</tr>
<tr bgcolor="#FFFFFF">
   <td><font size="+1"color="FF4444">491118</font></td>
   <td>TAER</td>
</tr>
<tr bgcolor="#FFFFFF">
   <td><font size="+1"color="FF4444">491119</font></td>
   <td>ETAT</td>
</tr>
</table></center>
```

```
<hr>
<p>&copy; Soci&eacute;t&eacute; 1999 -
<a href="mailto:webmaster@societe.com">
Contactez le WebMestre</a> </body> </html>
```

Ce texte - qui peut être créé en APL - comporte à la fois des informations de contenu (le titre des colonnes, le code des affaires, la valeur des codes), et une description de présentation. (couleurs utilisées, taille des caractères, épaisseur de traits ...)
La modification de la Charte graphique fait généralement reprendre la programmation de l'application.

2.2 Exemple de ce même tableau en XML

Voici un fichier XML qui, après transformation, pourrait produire le même affichage.

```
<?xml version="1.0" ?>
<!DOCTYPE socextract SYSTEM "socextract.dtd">
<extraction>
  <defdata>
    <code><nom>affnum</nom>
      <lib>Num&#xE9;ro d'affaire</lib>
    </code>
    <code><nom>afffam</nom>
      <lib>Code Famille</lib>
    </code>
  </defdata>
  <affaire>
    <donnee><nom>affnum</nom><val>491117</val></donnee>
    <donnee><nom>afffam</nom><val>TAER  </val></donnee>
  </affaire>
  <affaire>
    <donnee><nom>affnum</nom><val>491118</val></donnee>
    <donnee><nom>afffam</nom><val>TAER  </val></donnee>
  </affaire>
  <affaire>
    <donnee><nom>affnum</nom><val>491119</val></donnee>
    <donnee><nom>afffam</nom><val>ETAT  </val></donnee>
  </affaire>
</extraction>
```

Ce fichier n'est pas plus simple que le précédent. Mais il a d'autres qualités :

1 - Il n'implique aucune présentation particulière. (La mise en forme de ces données fait appel à une définition XSL, non décrite ici en détail
2 - Il peut servir à d'autres usages que le seul affichage HTML
3 - Il peut traverser le temps. Dans quelques années, ces données seront utilisables, plus facilement même qu'aujourd'hui.

2.3 Standards complémentaires

XML permet de définir un texte balisé, de définir des balises, puis de les exploiter; Il peut être plus simple de se ramener à des définitions de balises (DTD) standards. Leur exploitation se fait souvent par des applications écrites en Java, donc accessibles à de nombreux systèmes.

(Attendons une interface entre Java et APL)
En voici quelques exemples.

2.3.1 Les feuilles de style XSL

XSL, Extensible Style Sheet language permet de définir deux traitements consécutifs sur un fichier XSL :

1 - Réorganisation de l'arbre des données, pour placer les données dans l'ordre de leur affichage (de haut en bas, de gauche à droite) (XSLT : http://www.w3.org/TR/XSLT.
2 - Présentation (style) des données obtenues.

XSL est à l'état de proposition de recommandation (9 octobre 1999), les premières propositions datant de 1998. Les principaux fureteurs préparent la reconnaissance de XSL (mais XSL n'est pas encore stabilisé, attendons que la recommandation soit votée).

2.3.2 SMIL, Synchronized Multimedia Integration Language

SMIL permet de décrire - en XML - une présentation multimédia.
Son intérêt est évidemment de ne pas être la propriété d'un éditeur de logiciel, mais d'être utilisable par tous ceux qui le souhaitent. Vous n'êtes plus alors sous la dépendance des choix marketing conjoncturels d'un seul éditeur.
Son autre intérêt est évidemment de se présenter en XML, donc pouvant être calculé par votre programmation ,en APL.
SMIL est à l'état de Recommandation. On peut trouver sa définition en
http://www.w3.org/TR/REC-smil

2.3.3 Ressource Description Framework

RDF permet de définir des ressources, pour faciliter leur recherche.
RDF est à l'état de Recommandation. On peut trouver sa définition en
http://www.w3.org/TR/REC-rdf-syntax
Il est préférable de rechercher une information plus didactique, car cette recommandation est quelque peu abstraite.

3 Usage de XML en APL

Voici un premier exemple de manipulation de structures XML. Nous n'avons pas une grande expérience de ces traitements et les choix de représentation interne APL d'une description XML sont fondamentaux. De ce choix, peut découler le succès ou l'échec d'une application.
Voici donc une tentative de représentation interne d'une description XML.
(On cite description XML et non fichier XML car il s'agit en fait d'un vecteur de caractères, pouvant être placé dans un fichier, ou pouvant être stocké ailleurs, ou bien transmis etc...).

3.1 Représentation en APL

Ainsi, le choix de la représentation est fondamental.
La forme générale d'une description xml est ainsi :
<balise params> contenu </balise>
<nombre FMT="8 3">34.2</nombre>
Le contenu peut contenir d'autres balises.
La forme générale de représentation d'un élément est ainsi un vecteur d'au moins deux

éléments :

1 - Le premier élément est un vecteur alpha, nom de la balise
2 - Le second élément est un second vecteur alpha, les paramètres de la balise
3 - Les éléments suivants contiennent l'information présente entre la balise, et la fin de balise.

Par exemple
*
 est un élément contenant la balise de début, aucun paramètre, et faisant office de balise de fin.
Ceci est représenté par un vecteur de deux éléments
*<paragr>Voici le texte d'un paragraphe</paragr>
Cet exemple est un élément contenant un texte.
Cet exemple est représenté par un vecteur de trois éléments, le troisième élément est un vecteur alphabétique, contenant le texte.
*<facture><nombre>1</nombre><prix>320</prix></facture>
Cet exemple est alors représenté par un vecteur de 4 éléments :

1 - Le premier est le vecteur 'facture'
2 - Le second est un vecteur vide 3 - Le troisième est un vecteur de trois éléments, représentation de <nombre>1</nombre>
4 - Le quatrième élément est aussi un vecteur de trois éléments, représentation de <prix>320</prix>Cette représentation imbriquée conduit ainsi à une programmation récursive.

A titre d'exemple, l'un des outils de base de la boîte à outils XML permet de transformer une description XML en une représentation interne, sous forme de structure APL (cet outil est communément appelé Parser)
Ce parser, donc, extrait 4 parties de <facture><nombre>1</nombre><prix>320</prix></facture>puis analyse à nouveau, par un appel du même programme, l'analyse de <nombre>1</nombre>puis <prix>320</prix>

3.2 Outils en APL

Ces outils permettent la construction de structures internes, et leur usage :
1 - Créer une description XML à partir d'une structure interne
2 - Représenter en interne une description XML (Parser)
3 - Afficher une structure interne
4 - Manipuler une représentation interne

Voici la liste des objets contenus dans l'espace de démonstration

```
   )FNS
ACCUEIL   EXTRACT   INITXML   cOL       eb1
enu       err       gen       r         xmlAFF
xmlBLI    xmlBLT    xmlED0    xmlED1    xmlEXPORT
xmlPARSE  xmlPΔ0    xmlPΔDF   xmlPΔSP
      )VARS
xmlDOC xmlet
Affichages, lors du chargement de l'espace
  Travaux autour de xml
  ITM ←'élem' xmlBLI <contenu> constuire des struc.
individuelles
LITM ← l(c c) xmlBLT  tab      décrire un tableau
    TXT ← DEF xmlEXPORT ITM .. construire une forme d'export
```

```
(RC DEF TXT)← xmlPARSE  TXT    décortiquer un texte XML
              xmlAFF    ITM    afficher un item
exemple
    'demo' 1 xmlEXPORT  'lignes' ( 'col1' 'col2') xmlBLT 3 2 ⍴
⍳2000
    xmlAFF 'global' xmlBLI  'lignes' ( 'col1' 'col2') xmlBLT 3
2 ⍴ ⍳2000
```

3.2.1 Construire des éléments XML

1 - élément individuel
La construction d'un élément XML individuel se fait par la fonction xmlBLI
'nitm' 'param' xmlBLI 'val1' 'val2' ... (val : textes ou ,l,ments)
Cette fonction accepte à droite aussi bien des vecteurs alpha que des descriptions d'autres éléments, à concaténer.
'nombre' 'FMT="8 3"' xmlBLI '34.2'
Crée l'équivalent interne de
 <nombre FMT="8 3">34.2</nombre>
2 - tableau d'éléments
Une fonction permettant de décrire un tableau est aussi définie comme outil de base. Un tableau est créé, avec autant d'éléments lignequ'il y a
de lignes dans le tableau, chaque ligne étant composée d'une suite d'éléments dont les noms sont col1col2 ...

Par exemple
  'ligtarif' ('code' 'prixu') xmlBLT 3 2 æ 331 2, 334 2.5, 338 6
Crée une suite de 3 éléments 'ligtarif', chacun composé d'un élément 'code', et d'un élément 'prixu'
Ces trois éléments sont ensuite assemblés dans un élément simple, appelé 'tarif'
  'tarif' xmlBLI   ...
Nous obtenons ainsi une représentation interne d'un tableau.

```
  <tarif>
    <ligtarif>
      <code>331</code><prix>2</prix>
    </ligtarif>
    <ligtarif>
      <code>334</code><prix>2.5</prix>
    </ligtarif>
    <ligtarif>
      <code>338</code><prix>6</prix>
    </ligtarif>
  </tarif>
```

Ceci est un vecteur de 5 éléments :
a - Code 'tarif'
b - Vecteur vide (sans paramètres)
c - Vecteur de 4 éléments (élément ligtarif, première ligne)
1 - Code 'ligtarif'
2 - Vecteur vide (sans paramètres)
3 - Vecteur de 3 éléments : représentation de <code>331</code>
a - Code 'code'
b - Vecteur vide (sans paramètres)
c - Vecteur de caractères '331'

4 - Vecteur de 3 éléments : représentation de <prix>2</prix>
d - Vecteur de 4 éléments (élément ligtarif, deuxième ligne)
e - Vecteur de 4 éléments (élément ligtarif, troisième ligne)

3.2.2 Afficher pour contrôle

XmlAFF permet de rendre une forme éditée. Elle fait appel à xmlED1, récursive, qui retourne un vecteur de deux éléments : l'information de profondeur et la donnée éditée.

```
[0]    xmlAFF ITM;VNCH
[1]    ⍝ afficher un item constitué
[2]     VNCH←xmlED1 ITM
[3]     (cOL 0⊃VNCH),'|',cOL 1⊃VNCH
    ∇ 1999-09-05 21.31.32 (GMT-1)
```

Cette fonction 'xmlED1' doit aussi bien pouvoir travailler sur un élément vrai (vecteur généralisé) que sur un texte simple.

```
[0]    VNCH←xmlED1 ITM;U;V;VN
[1]    ⍝ rendre une forme éditée : toujours un vecteur de mes
sages
[2]    ⍝ premier cas, un texte seul
[3]     →(1=≡ITM)/L1 ⍝ texte seul, faux item
[4]     →(2=⍴ITM)/L3 ⍝ item début/fin
[5]     →((3≠⍴ITM)∨(1≠≡U←2⊃ITM))/L2 ⍝ item incluant d'autres
items
[6]     →(0≠⍴V←1⊃ITM)/L0
[7]     →0 VNCH←(,⊂,2)(,⊂'<',(↑ITM),'>',U,'</',(↑ITM),'>')
[8]    L0:→0 VNCH←(,⊂,2)(,⊂'<',(↑ITM),' ',V,'>',U,'</',
(↑ITM),'>')
[9]    ⍝ il s'agit d'un texte seul
[10]   L1:→0 VNCH←(,⊂⍳0)(,⊂ITM)
[11]   ⍝ second cas, un ou plusieurs items, rendre un vecteur de
messages
[12]   L2:VN←(,/(2↓⍳⍴ITM),¨⍪0⊃¨VN),(,/1⊃¨VN←xmlED1¨2↓ITM)
[13]    VNCH←(,¨∈¨(⊂,0),(0⊃VN),(⊂,0))((⊂'<',(↑ITM),('
',1⊃ITM),'>'),(' ',¨1⊃VN),(⊂'</',(↑ITM),'>'))
[14]    →0
[15]   ⍝ cas d'une balise isolée
[16]   L3:→0 VNCH←(,⊂0)(⊂'<',(0⊃ITM),' ',(1⊃ITM),'/>')
    ∇ 1999-09-05 21.36.01 (GMT-1)
```

Voici un exemple d'affichage d'une structure décrivant 3 lignes et deux colonnes. Cet exemple montre une structure XML identique à l'exemple du tarif, ci-dessus.

```
xmlAFF 'global' xmlBLI  'lignes' ( 'col1' 'col2') xmlBLT 3 2 ⍴
⍳2000
```

```
0      |<global >
 2 0   | <lignes >
 2 2 2 |  <col1>0</col1>
 2 3 2 |  <col2>1</col2>
```

```
2 0   | </lignes>
3 0   | <lignes >
3 2 2 |  <col1>2</col1>
3 3 2 |  <col2>3</col2>
3 0   | </lignes>
4 0   | <lignes >
4 2 2 |  <col1>4</col1>
4 3 2 |  <col2>5</col2>
4 0   | </lignes>
0     |</global>
```

Quelques exemples (la fonction r analyse un objet APL2, indique le nombre total d'éléments, sa représentation, le rang, la valeur des dimensions).

1 - Vecteur de deux éléments

```
     r K←('nombre' xmlBLI '32.3')('code' xmlBLI 'aaaa')
(2 G0 1 2)(3 G0 1 3)(6 C1 1 6)(0 C1 1 0)(4 C1 1 4)(3 G0 1 3)
      (4 C1 1 4)(0 C1 1 0)(4 C1 1 4)
```

`⋆(2 G0 1 2)` vecteur généralisé de deux objets

`⋆(3 G0 1 3)(6 C1 1 6)(0 C1 1 0)(4 C1 1 4)` vecteur généralisé de trois objets : un vecteur alpha de 6 caractères, un vecteur vide, un vecteur alpha de 4 caractères

`⋆(3 G0 1 3)(4 C1 1 4)(0 C1 1 0)(4 C1 1 4)` vecteur de 3 éléments

2 - Élément contenant deux éléments

```
    r  L ← 'ligne' xmlBLI K
(4 G0 1 4)(5 C1 1 5)(0 C1 1 0)(3 G0 1 3)(6 C1 1 6)(0 C1 1 0)
     (4 C1 1 4)(3 G0 1 3)(4 C1 1 4)(0 C1 1 0)(4 C1 1 4)
      L
ligne   nombre  32.3   code  aaaa

     r xmlAFF L
 0    | <ligne >
 2 2  |  <nombre>32.3</nombre>
 3 2  |  <code>aaaa</code>
 0    | </ligne>
```

La partie gauche indique l'accès à cet élément :
2 2 | <nombre>32.3</nombre>
Indique que le troisième élément de L est un vecteur généralisé dont le troisième élément contient '32.3'. On se servira de cette propriété pour manipuler des représentations internes.

3.2.3 Analyser une description XML

Il s'agit d'un Parser (dans la littérature habituelle) Cette fonction doit transformer une description XML (vecteur de caractères) en une structure APL2.
Cette fonction reconnaît certaines particularités des structures XML, comme les balises uniques de début et de fin, la demande de conservation des espaces, mais ne traite pas les substitutions (notion de entitiesen XML, non évoquée ici), ni les alphabets UNICODE, etc...

3.2.4 Transformer une structure en description XML, exporter.

Il s'agit de reconstruire une description XML, à partir d'une structure APL2.
Cette fonction est semblable à la fonction d'édition, mais n'a nul besoin de créer les informations indiquant l'adressage des objets constitutifs de cette représentation.
L'exportation doit rendre une description xml au moins bien formée sinon valide. Elle ajoute l'en-tête donnant le format XML, éventuellement le nom de la DTD. (par consultation d'une table construite par INITXML dans l'espace de démonstration).

Exemple

'demo' 1 xmlEXPORT î xmlED0 'lignes' xmlBLI 'Voici un texte'
<?xml version="1.0" encoding="ISO8859-1">
<!DOCTYPE demo SYSTEM "./demo.dtd">
<defdemo><lignes>Voici un texte</lignes></defdemo>
Voici un autre exemple d'usage (xmlEXPORT crée un vecteur de caractères, certains de ces caractères contenant des ordres de saut de lignes, ignorés en XML (sauf déclarations spéciales)
   'demo' 1 xmlEXPORT 'lignes' ('col1' 'col2') xmlBLT 3 2 æ ì2000
<?xml version="1.0" encoding="ISO8859-1">
<!DOCTYPE demo SYSTEM "./demo.dtd">
<defdemo >
 <lignes >
  <col1>0</col1>
  <col2>1</col2>
 </lignes>
 <lignes >
  <col1>2</col1>
  <col2>3</col2>
 </lignes>
 <lignes >
  <col1>4</col1>
  <col2>5</col2>
 </lignes>
</defdemo>

3.2.5 Modifier dans les structures

On peut modifier une structure APL, en se servant des primitives standard d'APL. On voit ici l'intérêt de la représentation donnée par xmlAFF :

```
   ZZ← 'demo' xmlBLI 'lignes' ( 'col1' 'col2') xmlBLT 3 2 ρ
ι2000
      xmlAFF ZZ

0       | <demo >
 2 0    |  <lignes >
 2 2 2  |   <col1>0</col1>
 2 3 2  |   <col2>1</col2>
 2 0    |  </lignes>
 3 0    |  <lignes >
 3 2 2  |   <col1>2</col1>
 3 3 2  |   <col2>3</col2>
 3 0    |  </lignes>
```

```
4 0   | <lignes >
4 2 2 |  <col1>4</col1>
4 3 2 |  <col2>5</col2>
4 0   | </lignes>
0     |</demo>
```

Affichage d'une portion de la structure. le quatrième élément de ZZ (en origine 0, nous en reparlerons bientôt) est une structure en elle-même :

```
    xmlAFF 3⊃ZZ

0   |<lignes >
2 2 | <col1>2</col1>
3 2 | <col2>3</col2>
0   |</lignes>
```

On peut modifier un texte

```
(3 3 2 ⊃ZZ) ← 'ABCD'
   xmlAFF 3⊃ZZ

0   |<lignes >
2 2 | <col1>2</col1>
3 2 | <col2>ABCD</col2>
0   |</lignes>

xmlAFF ZZ

0     |<demo >
2 0   | <lignes >
2 2 2 |  <col1>0</col1>
2 3 2 |  <col2>1</col2>
2 0   | </lignes>
3 0   | <lignes >
3 2 2 |  <col1>2</col1>
3 3 2 |  <col2>ABCD</col2>
3 0   | </lignes>
4 0   | <lignes >
4 2 2 |  <col1>4</col1>
4 3 2 |  <col2>5</col2>
4 0   | </lignes>
0     |</demo>
```

On peut aussi remplacer un élément complet :

```
    (3 ⊃ZZ) ← 'ABCD'
       xmlAFF ZZ

0     |<demo >
2 0   | <lignes >
2 2 2 |  <col1>0</col1>
2 3 2 |  <col2>1</col2>
2 0   | </lignes>
```

```
3      | ABCD
4 0    | <lignes >
4 2 2  |  <col1>4</col1>
4 3 2  |  <col2>5</col2>
4 0    | </lignes>
0      |</demo>
```

On peut replacer un élément à la place d'un texte

```
(3 ⊃ZZ) ← 'nouveau' xmlBLI 'Nouveau texte'
      xmlAFF ZZ
```

```
0      |<demo >
2 0    | <lignes >
2 2 2  |  <col1>0</col1>
2 3 2  |  <col2>1</col2>
2 0    | </lignes>
3 2    | <nouveau>Nouveau texte</nouveau>
4 0    | <lignes >
4 2 2  |  <col1>4</col1>
4 3 2  |  <col2>5</col2>
4 0    | </lignes>
0      |</demo>
```

Attention, conserver - par programmation - la conformité avec la description générale, donnée dans la DTD que vous avez choisi de respecter.

3.3 L'exemple du tableau

Cette fonction permet de préparer le fichier XML donné en exemple (p. 97).
Ce tableau ne se construit pas directement : il ne contient pas des balises au nom de la première colonne, au nom de la deuxième colonne, mais des balises capables de donner le nom de la colonne, ce qui remonte d'un niveau le niveau de généralisation de la description. Ce qui augmente d'un niveau la profondeur des traitements.
Ne soyons donc pas étonnés de rencontrer des doiubles each !
•Cette fonction commence, en lignes [3] à [10], par la constitution du tableau des noms de données, du tableau des valeurs.
•Il faut alors (ligne 14) assembler les éléments deux à deux, pour créer des paires nom/lib <nom>affnum</nom><lib>491117</lib>
•En ligne 16, on associe deux paires précédentes (une paire affnum, une paire afffam) en un élément de nom 'affaire'.
•En ligne 18, il ne reste plus qu'à assembler toutes ces 'affaires' en une seule description XML,

```
∇EXTRACT[□]∇
[0]    EXTRACT;DF;VA;IT0;IT1;U;TXT;K
[1]    ⍝ préparer une définition de tableau
[2]    ⍝ préparer la description des données
[3]     DF←0 2⍴0
[4]     DF←DF,[0]'affnum' 'Numéro d''affaire'
[5]     DF←DF,[0]'afffam' 'Code famille'
[6]     IT0←'defdata' xmlBLI 'code'('nom' 'lib')xmlBLT DF
[7]     VA←0 2⍴0
```

```
[8]    VA←VA,[0]'491117' 'TAER'
[9]    VA←VA,[0]'491118' 'TAER'
[10]   VA←VA,[0]'491119' 'ETAT'
[11]  ⍝ s'intéresser à chaque case du tableau (nom/lib)
[12]   K←((⍴VA)⍴⊂¨DF[;0]),¨⊂¨VA ⍝ le nom de la donnée devant
chaque donnée
[13]  ⍝ U tableau 3 ligne 2 col
[14]   U←(⊂'donnee')xmlBLI¨(⊂'nom' 'lib')xmlBLI⍤⍤K
[15]  ⍝ s'intéresser à chaque ligne du tableau
[16]   IT1←(⊂'affaire')xmlBLI¨⊂[1]U
[17]   '************ L''ensemble'
[18]   xmlAFF 'extraction' xmlBLI(⊂IT0),IT1
[19]  ⍝ produire le code
[20]   TXT←'extr' 1 xmlEXPORT(⊂IT0),IT1
[21]   '************ Fichier produit :'(r TXT)
[22]   TXT
    ∇ 1999-09-06 08.58.50 (GMT-1)
```

Voici le résultat de la création d'une chaîne XML, puis son analyse en structure APL et son affichage récursif.

EXTRACT L'ensemble

```
0       |<extraction >
2 0     | <defdata >
2 2 0   |  <code >
2 2 2 2 |   <nom>affnum</nom>
2 2 3 2 |   <lib>Num,ro d'affaire</lib>
2 2 0   |  </code>
2 3 0   |  <code >
2 3 2 2 |   <nom>afffam</nom>
2 3 3 2 |   <lib>Code famille</lib>
2 3 0   |  </code>
2 0     | </defdata>
3 0     | <affaire >
3 2 0   |  <donnee >
3 2 2 2 |   <nom>affnum</nom>
3 2 3 2 |   <lib>491117</lib>
3 2 0   |  </donnee>
3 3 0   |  <donnee >
3 3 2 2 |   <nom>afffam</nom>
3 3 3 2 |   <lib>TAER</lib>
3 3 0   |  </donnee>
3 0     | </affaire>
4 0     | <affaire >
4 2 0   |  <donnee >
4 2 2 2 |   <nom>affnum</nom>
4 2 3 2 |   <lib>491118</lib>
4 2 0   |  </donnee>
4 3 0   |  <donnee >
4 3 2 2 |   <nom>afffam</nom>
4 3 3 2 |   <lib>TAER</lib>
```

```
4 3 0    |  </donnee>
4 0      | </affaire>
5 0      | <affaire >
5 2 0    |  <donnee >
5 2 2 2  |   <nom>affnum</nom>
5 2 3 2  |   <lib>491119</lib>
5 2 0    |  </donnee>
5 3 0    |  <donnee >
5 3 2 2  |   <nom>afffam</nom>
5 3 3 2  |   <lib>ETAT</lib>
5 3 0    |  </donnee>
5 0      | </affaire>
0        |</extraction>
```

Fichier produit : (731 C1 1 731)

```
<?xml version="1.0" encoding="ISO8859-1">
<!DOCTYPE extr SYSTEM "./extr.dtd">
<extraction >
 <defdata >
  <code >
   <nom>affnum</nom>
   <lib>Num,ro d'affaire</lib>
  </code>
  <code >
   <nom>afffam</nom>
   <lib>Code famille</lib>
  </code>
 </defdata>
 <affaire >
  <donnee >
   <nom>affnum</nom>
   <lib>491117</lib>
  </donnee>
  <donnee >
   <nom>afffam</nom>
   <lib>TAER</lib>
  </donnee>
 </affaire>
 <affaire >
  <donnee >
   <nom>affnum</nom>
   <lib>491118</lib>
  </donnee>
  <donnee >
   <nom>afffam</nom>
   <lib>TAER</lib>
  </donnee>
 </affaire>
 <affaire >
  <donnee >
   <nom>affnum</nom>
```

```
   <lib>491119</lib>
  </donnee>
  <donnee >
   <nom>afffam</nom>
   <lib>ETAT</lib>
  </donnee>
 </affaire>
</extraction>
```

Une autre définition XML permettrait d'obtenir une forme conduisant à une programmation plus sommaire. C'est dans le choix des formes de fichiers XML que se tiendra la simplicité ou la difficulté d'un projet.
De fait, si le double each vous est peu familier, on se rend compte que sa pratique devient réflexe, et son usage se lit comme un idiome APL. Le iota-rho semble compliqué la promière fois, c'est maintenant un idiome courant. Il en va de même pour les formes de programmation développées dans cette fonction.

3.3.1 limitations

Au vu de la liste des limitations de cette application, on peut entrevoir l'étendue de la souplesse de XML.
1 - Usage d'entités prédéfinies, permettant des substituion de mots ou de grands textes
2 - Entités (internes et externes)
Accès à des fichiers privés ou des fichiers publics (accessibles à tous via la Toile) comme peuvent l'être les définitions de DTD
3 - Validation par dtd non installée (des produits externes peuvent se charger de ce travail)
4 - Sans transcodages d'alphabets
Un fichier XML précise l'alphabet qu'il utilise.
5 - Sections littérales
Pour retenir tous les caractères d'un élément, y compris les espaces multiples, les sauts de ligne
6 - Sans usage des caractères Unicode (bien qu'APL2 supporte l'alphabet Unicode, via sa primitive □UCS)

4 Sur la Toile, Download et liens

La littérature est abondante à propos de XML. On peut rechercher des informations dans différentes directions :
1 - Site W3C. Ce groupe a défini la version 1.0 de XML. Vous pouvez avoir accès à sa version la plus récente :
http://www.w3c.org/TR/REC-xml

2 - Site W3C, accès général aux recommandations et propositions de recommandations
http://www.w3c.org/TR

3 - Le site IBM des développeurs XML, assez pédagogique
http://www.ibm.com/developer/xml
Concernant ce texte, vous pouvez avoir accès aux exemples et à la programmation évoquée, en APL2. Si vous ne disposez pas de cet interprête, des versions de démonstration sont à votre disposition.

1 - Accès aux fichiers liés à ce texte :
http://www.mailhol.com/pub/afapl/xml

2 - Pour charger une version de démonstration d'APL2 (Windows ou OS2) :
http://www.software.ibm.com/ad/apl/download.html