XML parsing

Imprimé le 5 mai 2003

Projet	File parsing
Fichier	`xmltools.m`
Version	2.4
Date	mar2003
Auteur	Charles-Albert Lehalle
Mail	`charles.lehalle@miriadtech.com`
Relecteur	***NOT DEFINED***

Table des matières

<xmltools.m>

1 XML TOOLS FOR MATLAB

This is an OCAMAWEB (http ://ocamaweb.sourceforge.net) generated documentation.
This function manage the exchange of data between XML files and a MATLAB structure. The MATLAB structure is this : a **node** is a struct with fields *tag*, *value*, *attribs*, *child*. Where *tag* is the tag name, *value* its contents, *attribs* an array of structure with fields *name* and *value*, and *child* an array of such nodes.
All those fields are always present and can be empty except for the first node (root) that has only children.

The file `file.xml` containing :

```
<?xml version="1.0" ?>
<GUISCRIPT>
<SEQUENCE><NAME VALUE="A@p&gt;B@a"/>
   <MODULE NAME="R6" KEY="A" ACTION="P">
   </MODULE>
   <MODULE NAME="Composante" KEY="B" ACTION="A">
      <PARAM NB="2" NAME="Method">VALUE</PARAM>
   </MODULE>
</SEQUENCE>
</GUISCRIPT>
```

becomes this MATLAB structure :

```
child +-- tag:   '?xml', attribs: name: 'VERSION', value: '1.0', value: '', child: []
      +-- tag:   'GUISCRIPT', attribs: name: '', value: '', value: '',
           child: +-- tag:    'SEQUENCE', attribs: name: '', value: '', value: '',
                        child: +-- tag: 'NAME', attribs: name: 'VALUE', value: 'A@p&gt;B@a',  value: '', child: []
                               +-- tag:     'MODULE',
                               !    attribs: +-- name: 'NAME',   value: 'R6'
                               !             +-- name: 'KEY',    value: 'A'
                               !             +-- name: 'ACTION', value: 'P',
                               !     value: '', child: []
                               +-- tag:     'MODULE',
                                    attribs: +-- name: 'NAME',   value: 'Composante'
                                             +-- name: 'KEY',    value: 'B'
                                             +-- name: 'ACTION', value: 'A',
                                     value: '',
                                     child: tag:     'PARAM',
                                             attribs: +-- name: 'NB',   value: '2'
                                                      +-- name: 'NAME',    value: 'Method'
                                             value:   'VALUE', child: []
```

using `z=xmltools('file.xml');`. And `xmltools(z);` produces :

```
<?xml VERSION="1.0"?>
<GUISCRIPT>
  <SEQUENCE>
    <NAME VALUE="A@p&gt;B@a"/>
    <MODULE NAME="R6" KEY="A" ACTION="P"/>
    <MODULE NAME="Composante" KEY="B" ACTION="A">
      <PARAM NB="2" NAME="Method">
      VALUE
      </PARAM>
    </MODULE>
  </SEQUENCE>
</GUISCRIPT>
```

And for instance, we have :

```
>> z.child(2).child(1).child(3).child.attribs(2)
ans =
    name: 'NAME'
   value: 'Method'
```

◇

```
function z ← xmltools( arg, out_file, varargin)
% XMLTOOLS - tools for managing xml data sets
%      - if arg is a string : arg is an XML file to convert into MATLAB struct
%      - if arg is a variable : it is a MATLAB struct to write into XML, to st-
dout if out_file is not given
% use :
%  z ← xmltools('filename.xml'); read an xml file and store it into z
%  xmltools(z,'filename.xml'); write z into the file
%  xmltools(z,'get','tag-name'); returns only subset of z child which name is tag-
name
%
% project 'File parsing'
% title    'XML parsing'
% author   'Charles-Albert Lehalle'
% mailto   'charles.lehalle@miriadtech.com'
% version '2.4'
% date     'mar2003'

version ← '2.4';
```

1.1 READ AN XML FILE

Utilise ⟨1.1.1⟩ ⟨1.1.2⟩ ⟨1.2⟩ ⟨1.3⟩

◇

```
if isstr(arg)

  ⟨Récupération du fichier dans un string → 1.1.1⟩

  ⟨Parsing → 1.1.2⟩
  return
end

if ¬isstr(arg)

  ⟨SELECT A SUBSET OF z CHILD → 1.2⟩

  ⟨WRITE AN XML STRUCTURE → 1.3⟩
end
```

1.1.1 Récupération du fichier dans un string.

Utilisé par ⟨1.1⟩

◇

```
fid ← fopen(arg, 'r');
F ← fread(fid);
s ← char(F');
fclose(fid);
```

1.1.2 Parsing.

Utilisé par ⟨1.1⟩

◇

```
z ← parse_xml(s);
```

1.2 SELECT A SUBSET OF z CHILD

Utilisé par ⟨1.1⟩

◇

```
if length(varargin) ≡ 1
% warning: I will have to change the value of next at some places
  next ← 'child';

  z ← arg;
  if ¬isfield(z, next)
    error('XMLTOOLS:GET', 'For child selection, structured first argument is nee-
ded');
  end
  tag_name ← varargin{1};

  z ← get_childs(z, next, tag_name);
  return
end
```

1.3 WRITE AN XML STRUCTURE

Utilisé par ⟨1.1⟩ — Utilise ⟨1.3.1⟩ ⟨1.3.2⟩ ⟨1.3.3⟩

◇

⟨Selection de la cible → 1.3.1⟩

⟨Ecriture proprement dite → 1.3.2⟩

⟨Fermeture → 1.3.3⟩

1.3.1 Selection de la cible.

Utilisé par ⟨1.3⟩

◇

```
if nargin < 2
  fid ← 1;
else
  fid ← fopen(out_file, 'w');
end
```

1.3.2 Ecriture proprement dite.

Utilisé par ⟨1.3⟩

◇

```
write_xml(fid, arg);
```

1.3.3 Fermeture.

Utilisé par ⟨1.3⟩

◇

```
if nargin > 1
  fclose(fid);
end
```

2 Fonctions internes

Utilise ⟨2.1⟩ ⟨2.1.8⟩ ⟨2.2⟩ ⟨2.3⟩ ⟨2.3.1⟩ ⟨2.3.2⟩ ⟨2.3.3⟩

◇

⟨parser un string xml → 2.1⟩

⟨Parse attribs → 2.1.8⟩

⟨Ecriture d'une structure xml → 2.2⟩

⟨get childs with a specific tag name → 2.3⟩

⟨udeblank → 2.3.1⟩

⟨emptystruct → 2.3.2⟩

⟨Tokens → 2.3.3⟩

2.1 parser un string xml

Utilisé par ⟨2⟩ — Utilise ⟨2.1.1⟩ ⟨2.1.2⟩ ⟨2.1.3⟩

◇

```
function [z, str] ← parse_xml( str, current_tag, current_value, attribs, idx)

next ← 'child';

if nargin < 2
  current_tag   ← '';
  current_value ← '';
  attribs       ← '';
  idx           ← 0;
end
z ← [];

eot ← 0;

while ¬eot ∧ ¬isempty(udeblank(deblank(str)))

  f_end ← strfind(str, '</');
  f_beg ← strfind(str, '<');

  ⟨Si je n'ai plus de tag dans mon document → 2.1.1⟩

  if isempty(f_end)
    f_end ← length(str)
  else
    f_end ← f_end(1);
  end
  if isempty(f_beg)
    f_beg ← length(str)
  else
    f_beg ← f_beg(1);
  end

  if f_end ≤ f_beg
    ⟨je rencontre une fermeture → 2.1.2⟩
  else
    ⟨je rencontre une ouverture → 2.1.3⟩
end
```

2.1.1 Si je n'ai plus de tag dans mon document.

Utilisé par ⟨2.1⟩

◇

```
    if isempty(f_end) ∧ isempty(f_beg)

      if ¬strcmp(lower(current_tag), '?xml') ∧ ¬isempty(current_tag)
        error('xmltools:parse_xml', 'malformed xml string (current [%s])', current_tag);
      else
        fprintf('end parsing at level %d\n',idx);
        eot ← 1;
        return
      end
    end
```

2.1.2 je rencontre une fermeture.

Utilisé par ⟨2.1⟩

◇

```
      new_tag ← str((f_end+2):end);
      str_t   ← str(1:f_end-1);
      f_end ← strfind(new_tag,'>');
      if isempty(f_end)
        error('xmltools:parse_xml', 'malformed xml string : never ending tag [%s] en-
  countered', current_tag);
      end
      f_end ← f_end(1);
      str     ← new_tag(f_end+1:end); % reste
      new_tag ← new_tag(1:f_end-1);
      if ¬strcmp(new_tag, current_tag)
        error('xmltools:parse_xml', 'malformed xml string : [%s] not properly closed (clo-
  sing [%s] encountered)', current_tag, new_tag);
      end
      fprintf('%sclose [%s]\n', repmat(' ', 2*(idx-1),1), current_tag);
      z.tag     ← upper(current_tag);
      z.attribs ← parse_attribs(attribs);
      z.value   ← udeblank(deblank(sprintf('%s %s',current_value, str_t)));
      eot       ← 1;
```

2.1.3 je rencontre une ouverture.

je vais appeler le même code sur ce qu'il y a après moi

Utilisé par ⟨2.1⟩ — Utilise ⟨2.1.4⟩ ⟨2.1.5⟩ ⟨2.1.6⟩ ⟨2.1.7⟩

◇

```
    current_value ← sprintf('%s %s', current_value, str(1:f_beg-1));
    new_tag    ← str(f_beg+1:end);
    f_end ← strfind(new_tag,'>');
    if isempty(f_end)
      error('xmltools:parse_xml', 'malformed xml string : never ending tag encoun-
tered');
    end
    f_end    ← f_end(1);
    str_t    ← new_tag(f_end+1:end);
    new_tag ← new_tag(1:f_end-1);
    if (new_tag(end)  ≡  '/')∨(new_tag(end)  ≡  '?')
      ⟨Self closing tag → 2.1.4⟩
    end
    ⟨Attributs → 2.1.5⟩
    fprintf('%sopen  [%s]\n', repmat(' ', 2*idx,1), new_tag);

    if eot
      ⟨If self-colsing tag → 2.1.6⟩
    else
      ⟨Appel du même code sur la suite → 2.1.7⟩
    end
  end
```

2.1.4 Self closing tag.

Je met (temporairement !) eot à 1, cela me permet de passer quelques lignes de code tranquilement

Utilisé par ⟨2.1.3⟩

◇

```
      eot ← 1;
```

2.1.5 Attributs.

Utilisé par ⟨2.1.3⟩

◇

```
    f_beg    ← strfind(new_tag, ' ');
    if isempty(f_beg)
      new_attribs ← '';
      if eot
```

```
 new_tag ← new_tag(1:end-1);
      end
    else
      new_attribs ← new_tag(f_beg+1:end);
      if eot
 new_attribs ← new_attribs(1:end-1);
      end
      new_tag     ← new_tag(1:f_beg-1);
    end
```

2.1.6 If self-colsing tag.

Utilisé par ⟨2.1.3⟩

◇

```
      fprintf('%sclose [%s]\n', repmat(' ', 2*idx,1), new_tag);
      new_attribs ← parse_attribs( new_attribs);
      if isfield(z, next)
        nxt ← getfield(z, next);
        nxt(end+1) ← struct( 'tag', new_tag, 'attribs', new_attribs, 'value', '', next, []);
        z   ← setfield(z, next, nxt);
% z.(next)(end+1) = struct( 'tag', new_tag, 'attribs', new_attribs, 'value', '', next, []);
      else
        z ← setfield(z, next, struct( 'tag', new_tag, 'attribs', new_attribs, 'value', '', next,
% z.(next) = struct( 'tag', new_tag, 'attribs', new_attribs, 'value', '', next, []);
      end
      str ← str_t;
      eot ← 0;
```

2.1.7 Appel du même code sur la suite.

Utilisé par ⟨2.1.3⟩

◇

```
%  et stockage du resultat dans mes children.
%  Le code met aussi à jour le string courant str,
%  il en enlève la partie correspondant au string que je viens de trouver.
      [t,str] ← parse_xml(str_t, new_tag, '', new_attribs, 1+idx);
      if isfield(t, next)
 nx ← getfield( t, next);
% nx = t.(next);
      else
 nx ← [];
      end
      if isfield(z, next)
        nxt ← getfield(z, next);
        nxt(end+1) ← struct( 'tag', t.tag, 'attribs', t.attribs, 'value', t.value, next, nx);
        z   ← setfield(z, next, nxt);
% z.(next)(end+1) = struct( 'tag', t.tag, 'attribs', t.attribs, 'value', t.value, next, nx);
      else
 z ← setfield(z, next, struct( 'tag', t.tag, 'attribs', t.attribs, 'value', t.value, next, nx));
% z.(next) = struct( 'tag', t.tag, 'attribs', t.attribs, 'value', t.value, next, nx);
      end
```

2.1.8 Parse attribs.

Utilisé par ⟨2⟩

◇

```
function z ←  parse_attribs( a)
if isempty(a)
  z ← struct( 'name', '', 'value', '');
  return
end
b ← tokens(a, ' ');
j ← 1;
for i←1:length(b)
  if ¬isempty(b{i})
    t ← tokens(b{i}, '←');
    if length(t) ≡ 2
      u ← t{2};
      if u(1) ≡ '"'
 u ← u(2:end);
      end
      if u(end) ≡ '"'
 u ← u(1:end-1);
      end
      z(j) ← struct( 'name', upper(t{1}), 'value', u);
    else
      z(j) ← struct( 'name', upper(a), 'value', '');
    end
    j ← j +1;
  end
end
```

2.2 Ecriture d'une structure xml

Utilisé par ⟨2⟩ — Utilise ⟨2.2.1⟩ ⟨2.2.4⟩ ⟨2.2.5⟩ ⟨2.2.6⟩

◇

```
function z ← write_xml(fid, xml_struct, idx)

next ← 'child';

if nargin < 3
  idx ← 0;
end

margin ← repmat(' ',2*idx,1);

closed_tag ← 1;
⟨Ouverture du tag → 2.2.1⟩

⟨Ecriture de la value → 2.2.4⟩

⟨Ecriture des enfants → 2.2.5⟩

⟨Fermeture du tag → 2.2.6⟩
```

2.2.1 Ouverture du tag.

Utilisé par ⟨2.2⟩ — Utilise ⟨2.2.2⟩ ⟨2.2.3⟩

◇

```
if isfield(xml_struct, 'tag')
  closed_tag ← 0;
  fprintf(fid, '%s<%s', margin, xml_struct.tag);
  ⟨Ecriture des attributs → 2.2.2⟩

  ⟨Gestion des Auto closed tags → 2.2.3⟩
end
```

2.2.2 Ecriture des attributs.

Utilisé par ⟨2.2.1⟩

◇

```
  if ¬isfield(xml_struct, 'attribs')
```

```
    error('xmltools:write_xml', 'malformed MATLAB xml structure : tag without at-
tribs');
  end
  for i←1:length(xml_struct.attribs)
    if ¬isempty(xml_struct.attribs(i).name)
      fprintf(fid, ' %s←"%s"', xml_struct.attribs(i).name, xml_struct.attribs(i).value);
    end
  end
```

2.2.3 Gestion des Auto closed tags.

Si le tag n'est pas auto fermé, alors `closed_tag` est à zéro

Utilisé par ⟨2.2.1⟩

◇

```
  if ¬isfield(xml_struct, next)
    error('xmltools:write_xml', 'malformed MATLAB xml structure : tag without %s', next);
  end
  if ¬isfield(xml_struct, 'value')
    error('xmltools:write_xml', 'malformed MATLAB xml structure : tag without va-
lue');
  end
  if xml_struct.tag(1)  ≡  '?'
    fprintf(fid, '?>\n');
    closed_tag ← 1;
  elseif isempty(getfield(xml_struct, next)) ∧ isempty(xml_struct.value)
% elseif isempty(xml_struct.(next)) & isempty(xml_struct.value)
    fprintf(fid, '/>\n');
    closed_tag ← 1;
  else
    fprintf(fid, '>\n');
  end
```

2.2.4 Ecriture de la value.

Utilisé par ⟨2.2⟩

◇

```
if isfield(xml_struct, 'value')
  if ¬isempty(xml_struct.value)
    fprintf(fid, '%s%s\n', margin, xml_struct.value);
  end
end
```

2.2.5 Ecriture des enfants.

Utilisé par ⟨2.2⟩

◇

```
if ¬isfield(xml_struct, next)
  error('xmltools:write_xml', 'malformed MATLAB xml structure : tag without %s', next);
end
those_children ← getfield(xml_struct, next);
% those_children = xml_struct.(next);
for i←1:length(those_children)
  write_xml(fid, those_children(i), idx+1);
end
```

2.2.6 Fermeture du tag.

Utilisé par ⟨2.2⟩

◇

```
if ¬closed_tag
  fprintf(fid, '%s</%s>\n', margin, xml_struct.tag);
end
```

2.3 get childs with a specific tag name

Utilisé par ⟨2⟩

◇

```
function z ← get_childs(z, next, tag_name);
u ← getfield(z, next);
zo ← [];
for i←1:length(u)
  v ← u(i);
  if strcmp(v.tag, tag_name)
    if isempty(zo)
      zo.anext← v;
    else
      zo.anext(end+1) ← v;
    end
  end
```

```
end
z ← [ zo.anext ];
```

2.3.1 udeblank.

Utilisé par ⟨2⟩

◇

```
function s ← udeblank(str)
s ← deblank(str(end:-1:1));
s ← s(end:-1:1);
if length(s) ≡ 0
  s ← '';
end
```

2.3.2 emptystruct.

Utilisé par ⟨2⟩

◇

```
function z ← emptystruct(next)
z ← struct( 'tag', [], 'value', [], 'attribs', [], next, []);
```

2.3.3 Tokens.

Utilisé par ⟨2⟩

◇

```
function l ← tokens(str,del)
l←{} ;
%  Boucle sur les tokens.
del ← sprintf(del) ;
while ¬isempty(str)
  [tok,str] ← strtok(str,del) ;
  l{end+1} ← tok ;
end
```