



Journées Perl 2005



- x **A propos de XML**
- x **Que faire avec du XML ?**
- x **TIMTOWTDI, Perl oblige !**
- x **Parser pour les nuls**
- x **Cuisine & dépendances**
- x **XML::Simple**
- x **XML::LibXML**
- x **XML::SAX**
- x **XML::Twig**
- x **Ecrire du XML**
- x **On ze ouaibe again**



Agenda



- x **A propos de XML**
- x **Que faire avec du XML ?**
- x **TIMTOWTDI, Perl oblige !**
- x **Parser pour les nuls**
- x **Cuisine & dépendances**
- x **XML::Simple**
- x **XML::LibXML**
- x **XML::SAX**
- x **XML::Twig**
- x **On ze ouaibe again**



A propos de XML¹

- x **XML est un *méta-langage* à balises**

- x XML

- x **E**Xtensible **M**arkup **L**anguage

- x Normalisé par le “World Wide Web Consortium”, le [W3C](#)

- x Méta-langage

- x Plus des règles pour construire des langages qu'un langage par lui-même

- x Deux niveaux de définition (obligatoire : “bien formé” + facultatif : “valide”)

- x Balises

- x Eléments (entre 2 balises).....<nom> ... </nom>

- x Instructions de traitement<?cible attribut="valeur" ... ?>

- x Bloc CDATA..... <!CDATA[...]>

- x Commentaires..... <!-- ... -->

- x Entités..... &nom;



A propos de XML²

x **Elément**

- x Défini entre une balise d'ouverture et une de fermeture
 - x Le nom (sensible à la casse) commence par une lettre ou un “_”
 - x Constitué par des lettres (au sens Unicode), chiffres, “.”, “:” ou “_”
- x Contient d'autres éléments ou du texte
 - x Par exemple : `<para>c'est un <emphasis>bon</emphasis> exemple</para>`
 - x Et ainsi de suite de façon hiérarchique
- x Peut faire parti d'un espace de noms (optionnel mais recommandé)
 - x L'espace de nom préfixe le nom de l'élément à l'aide d'un “:”
 - x `<asile xmlns="http://www.nutz.org"><foo> <!-- etc. --> <foo> <asile>`
 - x L'URI (non nécessairement valide) sert de clé
 - x Permet l'unicité des noms d'éléments au travers des différents dialectes
 - x `<gabuzomeu xmlns:asile="http://www.nutz.org"> <ga/> <bu/> <zo/> <meu/>
<asile:foo> <bar nom="toto"> <!-- etc. --> </bar> </asile:foo>
</gabuzomeu>`



A propos de XML³

x **Élément (suite)**

- x Peut posséder des attributs (optionnel)
 - x Par exemple : `<foo bar="fred"> ... </foo>`
 - x Valeur de l'attribut entre cotes ou guillemets
 - x Mêmes règles pour les noms d'attributs que pour les noms d'éléments

x **Instruction de traitement**

- x Donne des informations spécifiques à un processeur XML
 - x La plus connue de toutes (1^{ère} ligne d'un document XML) :
 - x `<?xml version="1.0" encoding="ISO-8859-1" standalone="no">`

x **Bloc CDATA**

- x Totalement ignoré par le parser
 - x `<![CDATA[on y met ce que l'on veut]]>`



A propos de XML⁴

- x **Commentaire**

- x Non imbricable à l'intérieur d'un autre commentaire, hélas !

- x **Entité**

- x Définition

- x Seulement au travers d'une DTD : `<!ENTITY peur "pick-a-boo">`
- x Si pas de DTD (XSD, etc.) en intégrer une réduite aux définitions d'entités

- x Référence

- x A l'aide de `&entité;` : `<bla>Il m'a fait &peur; !</bla>`

- x **Grosso-modo, 2 types de documents XML**

- x Données (DB, structures d'échanges transitoires à la SOAP, etc.)
- x Contenu ("documents" à la DocBook, SVG, etc.)
- x Respectivement "plus de balises que de contenu" et vice-versa



A propos de XML⁵

- x **Un document doit *obligatoirement* être bien formé**
 - x C'est le minimum syndical pour que le parser puisse officier
 - x Voir plus loin à propos des “parsers”
 - x Respecte la syntaxe de la recommandation du W3C
 - x Déclaration de prologue correcte sur la première ligne :
 - x `<?xml version='1.0' encoding='US-ASCII' standalone='no'?>`
 - x Toute balise fermante correspond à une balise ouvrante
 - x Balise vide : `<foo />`
 - x Pas de recouvrement entre balises
 - x Par exemple, le cas suivant est prohibé : `<foo><bar></foo></bar>`
 - x Les noms de balises et d'attributs sont valides
 - x Voir plus haut



A propos de XML⁶

- x **Un document peut éventuellement être valide**
 - x On vérifie que le document est bien conforme à sa DTD
 - x Le validateur peut être un outil externe (e.g., [rxp](#))
 - x Mais une DTD n'est ni exhaustive, ni représentée en XML
 - x Utiliser sinon “XML Schema”, + puissant mais aussi + complexe
 - x Ou Relax-NG, Schematron, etc.
- x **Pourquoi utiliser XML ?**
 - x C'est du texte (Unicode) indépendant de la plateforme
 - x Implique une structure de données, à *priori* décorréllée du rendu
 - x Apporte une palette d'outils (parser, validateur, * <-> XML, etc.)
 - x Et de technologies dérivées (SAX, DOM, XPath, XSLT, etc.)



A propos de XML⁷

x **Plusieurs manières de représenter la même chose**

- x `<bar nom="toto"><fred>bla</fred></bar>`
- x `<bar><nom>toto</nom><fred>bla</fred></bar>`
- x `<bar nom="toto"><fred dedans="bla"/></bar>`
- x Etc.
- x Pour un *décideur pressé*, tout ça “c'est pareil, c'est du XML” !

x **Pourquoi ne pas utiliser XML ?**

- x Dernier “buzzword” à la mode (“HBB platform”)
- x Ne remplit pas les fonctions d'un SGBDR
- x Cf “[lightning talk](#)” de M. Rodriguez



Agenda



- x A propos de XML
- x **Que faire avec du XML ?**
- x **TIMTOWTDI, Perl oblige !**
- x Parser pour les nuls
- x Cuisine & dépendances
- x XML::Simple
- x XML::LibXML
- x XML::SAX
- x XML::Twig
- x Ecrire du XML
- x **On ze ouaibe again**



Que faire avec du XML ?¹

- x **Parser**

- x Transforme un document en une structure (arbre ou flux)

- x **Valider**

- x Croise un document avec une spécification
- x DTD, Schema, Relax-NG, etc.

- x **Traiter**

- x A interpréter dans son sens le plus large...
- x Dépend du type de la structure issue du parser

- x **Ecrire**

- x Dans une chaîne, une filehandle, un fichier, etc.



Document “fil rouge”

- x **Le (trivial) document suivant sert de fil rouge**

```
x <foo>
  <bar nom="toto">
    <fred>bla</fred>
  </bar>
  <bar nom="titi">
    <fred>blabla</fred>
  </bar>
</foo>
```

- x **Attention ! c'est différent de :**

```
x <foo><bar nom="toto"><fred>bla</fred></bar><bar nom="titi"><fred>blabla</fred></bar></foo>
```

- x **Car présence de noeuds typés texte supplémentaires**

- x $E(\text{foo}), T(\backslash n \backslash s+), E(\text{bar}), T(\backslash n \backslash s+), E(\text{fred}), T(\backslash n \backslash s+)$, etc.
- x On fait presque du SAX sans le vouloir ;)
- x Piège classique (mais certains parsers les font sauter)



Agenda



- x A propos de XML
- x Que faire avec du XML ?
- x **TIMTOWTDI, Perl oblige !**
- x Parser pour les nuls
- x Cuisine & dépendances
- x XML::Simple
- x XML::LibXML
- x XML::SAX
- x XML::Twig
- x Ecrire du XML
- x **On ze ouaibe again**



TIMTOWTDI¹

- x **There is more than one way to do it**
 - x On fait du Perl, oui ou non ?
- x **Nous envisagerons 4 types de méthodes**
 - x Oublier XML.....XML::Simple
 - x Arbres DOM.....XML::LibXML
 - x Flux SAX.....XML::SAX
 - x Hybride flux-arbre.....XML::Twig
- x **XSLT ne sera (hélas ?) pas abordé**
 - x Bien que des “hooks” existent au travers de XML::libXSLT
 - x Voir “Perl Cookbook”, chapitre 22, section 7
 - x Transformer du XML avec du XML ([XSL](#)) et du XPath



Agenda

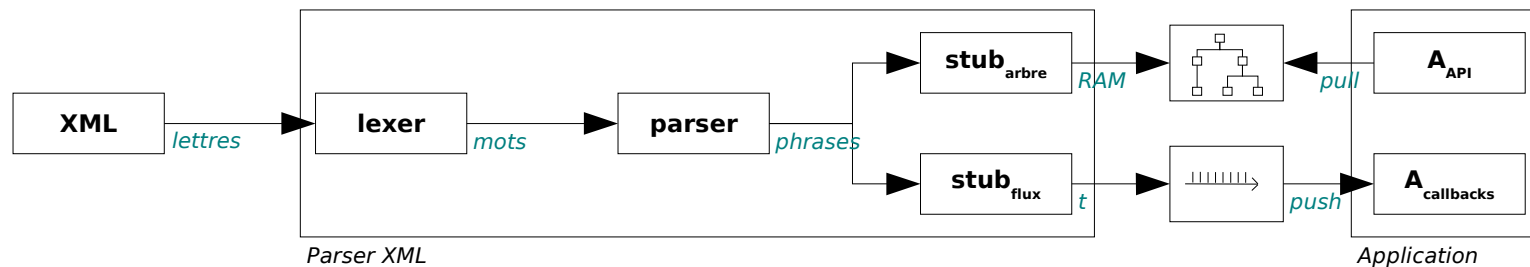


- x A propos de XML
- x Que faire avec du XML ?
- x TIMTOWTDI, Perl oblige !
- x **Parser pour les nuls**
- x Cuisine & dépendances
- x XML::Simple
- x XML::LibXML
- x XML::SAX
- x XML::Twig
- x Ecrire du XML
- x **On ze ouaibe again**



Parser pour les nuls¹

- x **Un document est un flux de caractères**
 - x Sérialisés dans un fichier, une chaîne, etc.
- x **Un parser réside entre le document & l'application**
 - x Il crée la structure attendue par l'application
 - x Parser, c'est bien gentil, mais vers quel type de structure ?
- x **Parser XML = qw(Lexer Parser Stub)**





Parser pour les nuls²

- x **Lexer**

- x Décompose le flux de *caractères* en *mots* (ou “tokens”)
- x Cf `lex(1)`, `flex(1)`, etc.

- x **Parser**

- x Applique une grammaire sur les *mots* pour faire des *phrases*
- x Cf `yacc(1)`, `byacc(1)`, `bison(1)`, `antlr(1)`, etc.

- x **Stub**

- x Construit (grâce aux *phrases*) un arbre en mémoire **ou**
- x Génère (idem) un flux d'évènements



Parser pour les nuls³

- x **Un parser requiert une spécification à appliquer**
 - x Description *formelle* en BNF ou mieux, en EBNF
 - x Une partie concerne le lexer (définition des “tokens”)
 - x L'autre est traduite en une grammaire dédiée à un parser donné
- x **Ecrire un parser en Perl**
 - x Parse::RecDescent.....simple, flexible mais lent
 - x Parse::Yapp..... pas encore essayé... il paraît que c'est bien
 - x Perl-byacc.....un byacc bidouillé pour générer du PurePerl
 - x Lex (C) + Yacc (C) + XS (C)..... compliqué, rigide mais rapide



Parser pour les nuls⁴

x Extrait de la description EBNF de XML

```
x document      ::= prolog element

element         ::= EmptyElemTag | STag content Etag
content        ::= (element | CharData)*

STag            ::= '<' Name (S Attribute)* S? '>'
Attribute       ::= Name Eq AttValue
AttValue        ::= '"' [^%&"] "'" | "'" [^%&'] '"'
ETag            ::= '<' Name S? '>'
EmptyElemTag   ::= '<' Name (S Attribute)* S? '/>'

S               ::= (#x20 | #x9 | #xD | #xA)+
Letter          ::= [a-zA-Z_.:]
Digit           ::= [0-9]
NameChar        ::= Letter | Digit | '.' | '-' | '_' | ':'
Name            ::= (Letter | '_' | ':') NameChar*
Eq              ::= S? '=' S?
```

x Ca descent très bas (et nous n'avons pas tout mis !)

x Description exhaustive sur le site du [W3C](#)



Parser pour les nuls⁵

x **Une petite grammaire pour Parse::RecDescent**

```
x my $grammaire = {
    document    : prolog(?) element
    prolog      : '<?xml' attribute(s?) '?>'
    element     : start content end
    start       : '<' name '>'
    content     : (element|characters)(s?)
    end         : '</' name '>'

    # briques de base
    name        : m/[A-Za-z_]\w*/
    attribute   : identifiant '=' m/[""]*/ '""'
    characters  : m/[^<]+/
};
```

x Le lexer est mélangé au parser (les tokens sont entre cotes)

x Noter l'emploi de RegExp pour définir les tokens

x **Il n'y a plus qu'à insérer le code aux bons endroits**

x Parse::RecDescent recollera les morceaux



Parser pour les nuls⁶

x Exemple de grammaire ~ SAX

```
x $grammaire = q {
  document      : header(?) { $a = {} } element
  header        : '<?xml' attribute(s?) '?>'
  element       : start content end
  start         : '<' identifieur { $a = {} } attribute(s?) '>'
                { push @{$main::id}, "start_element($item[2]," . main::dumper($a) . ")" }
  content       : (element|characters)(s?)
  end           : '</' identifieur '>' { push @{$main::id}, "end_element($item[2])" }
# briques de base
identifieur    : m/[A-Za-z_]\w*/
attribute      : identifieur '=' m/["^"]*/ "'" { $a->{$item[1]} = $item[3] }
characters     : m/[^<]+/ { push @{$main::id}, "characters($item[1])" }
};

x $id = [
  'start_element(foo,{} )',
  'start_element(bar,{nom=>toto})',
  'start_element(fred,{} )',
  'characters(bla)',
  'end_element(fred)',
  'start_element(bar,{nom=>titi})',
  'start_element(fred,{} )',
  'characters(blaba)',
  'end_element(fred)'
];
```



Parser pour les nuls⁷

x Exemple de grammaire ~ DOM

```
x $grammaire = q {
  document      : header(?) element
  header        : '<?xml' attribute(s?) '?>'
  element       : start content end
  start         : '<' identifieur { $main::l = $main::h; push @{$main::h}, ( $item[2],
                                $main::a = {}, $main::t = [], $main::h = [] ) } attribute(s?) '>'
  content       : (element|characters)(s?)
  end           : '</' identifieur '>' { $main::h = $main::l }
# briques de base
identifieur    : m/[A-Za-z_]\w*/
attribute      : identifieur '=' m/["^"]*/ "'" { $main::a->{$item[1]} = $item[3] }
characters     : m/[^<]+/ { push @{$main::t}, $item[1] }
}; # FIXME le deuxième bar est un niveau trop bas

x $id = [ # [ $nom, \%attributs, \@texte, \@enfants ]
  'foo', {}, [], [
    'bar', { 'nom' => 'toto' }, [], [
      'fred', {}, [ 'bla' ], [],
    ],
    'bar', { 'nom' => 'titi' }, [], [
      'fred', {}, [ 'blabla' ], []
    ]
  ]
];
```



Parser pour les nuls⁸

x Utilisation des grammaires de Parse::RecDescent

```
x use Parse::RecDescent;
  $::RD_HINT = 1; # pour des messages d'erreur plus verbeux

  $grammaire = q{
    # bla bla bla
  };

  $id = [];

  $h = $id; # si ~ DOM

  $parser = new Parse::RecDescent ( $grammaire ) or die "Mauvaise grand'mère !\n";

  open $fh, 'foo.xml' or die $?;
  $parser->document( $_ ) while <$fh>;
  close $fh;

  use Data::Dumper::Simple;
  $Data::Dumper::Indent = $Data::Dumper::Sortkeys = 1;

  print Dumper( $id );
```



Agenda



- x A propos de XML
- x Que faire avec du XML ?
- x TIMTOWTDI, Perl oblige !
- x Parser pour les nuls
- x **Cuisine & dépendances**
- x XML::Simple
- x XML::LibXML
- x XML::SAX
- x XML::Twig
- x Ecrire du XML
- x **On ze ouaibe again**



Cuisine & dépendances¹

- x **Notions de front-end et de back-end**
 - x Le front-end est le côté de l'application
 - x Le back-end est le côté du XML
- x **Front-end**
 - x Le module du [CPAN](#) utilisé
- x **Back-end**
 - x Bibliothèque codée en C liée dynamiquement
 - x Encapsulée dans des objets Perl à l'aide de XS
 - x Les 2 principales sont [expat](#) et [libxml2](#)
 - x Certains BE sont du "PurePerl" (portable, "fallback", mais + lent)
 - x Certains modules peuvent utiliser indifféremment plusieurs BE



Cuisine & dépendances²

x **BE sous forme de bibliothèque externe**

x Avantages

- x Plus rapide
- x Plus petite empreinte mémoire (~ facteur d'expansion)
- x On délègue à d'autres codeurs compétents, ne réinvente pas la roue, tout ça...

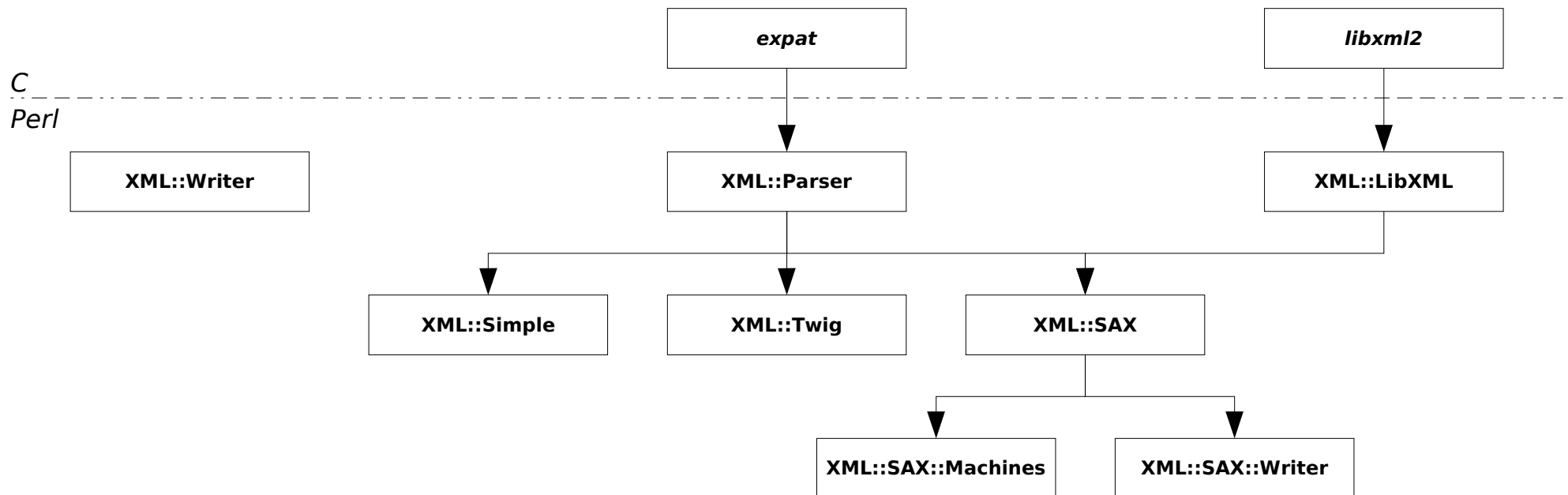
x Inconvénients

- x Compilation de bibliothèques au préalable
- x Pas géré par le module CPAN
- x Crée une dépendance sur la plateforme (e.g., .so)
- x Gestion croisée des versions FE et BE (e.g., libxml2 / XML::LibXML)
- x Peut nécessiter la modification de la variable LD_LIBRARY_PATH (si non root)
- x Impacte le déploiement de l'application (lib/perl5 + lib)
- x Mais la dépendance sur le module du CPAN l'impacte déjà
- x Gestion de paquetages (vendor_perl / site_perl + -devel), fonction de la distro



Cuisine & dépendances²

- x **Certains modules sont le BE d'autres modules**
 - x Par exemple, XML::Twig est basé sur XML::Parser
- x **Photo de famille**





Agenda



- x A propos de XML
- x Que faire avec du XML ?
- x TIMTOWTDI, Perl oblige !
- x Parser pour les nuls
- x Cuisine & dépendances
- x **XML::Simple**
- x XML::LibXML
- x XML::SAX
- x XML::Twig
- x Ecrire du XML
- x **On ze ouaibe again**



XML::Simple¹

- x **Parser vers une structure de données**
 - x Combo de LoL, LoH, HoL, HoH
 - x Utiliser ensuite les accesseurs et itérateurs standards

- x **C'est à priori simple**

```
x use XML::Simple;
  $id = XMLin ( 'foo.xml' );

x $id = {
  'bar' => [
    { 'fred' => 'bla',    nom => 'toto' },
    { 'fred' => 'blabla', nom => 'titi' },
  ],
};
```

- x **Mais besoin de contrôler la structure résultante**

- x Par exemple, on aimerait bien indexer l'élément bar sur son nom
- x De nombreux paramètres existent pour XMLin() (RTFM)



XML::Simple²

x Replier la structure sur un attribut

```
x XMLin ( 'foo.xml', KeyAttr => [ 'nom' ] );  
  
x $id = {  
    'bar' => {  
        'titi' => { 'fred' => 'blabla' },  
        'toto' => { 'fred' => 'bla'   },  
    },  
};
```

x Anticiper les enfants uniques

```
x $id = {  
    'bar' => { 'fred' => 'bla', nom => 'toto' }, # si bar(titi) n'existe pas  
};  
  
x XMLin ( 'foo.xml', KeyAttr => [ 'nom' ], ForceArray => [ 'bar' ] );  
  
x $id = { 'bar' => { 'toto' => { 'fred' => 'bla' } } };
```

x Garder l'élément racine

```
x XMLin ( 'foo.xml', KeyAttr => [ 'nom' ], ForceArray => [ 'bar' ], KeepRoot => 1 );  
  
x $id = { 'foo' => { 'bar' => 'titi' => { # etc.
```



XML::Simple³

- x **Adapté à des documents très structurés**
 - x L'auteur parle de fichiers ~ config (pas de “mixed-content”)
 - x Sinon anticiper et appliquer les bons paramètres à XMLin()
 - x Doc bien faite (*important, handy, seldom, advanced*, etc.)
 - x Attention ! replit par défaut sur ['name', 'key', 'id']
- x **Non standardisé**
 - x Contrairement à DOM ou SAX
- x **Pour info...**
 - x Peut néanmoins générer des évènements SAX
 - x Implémente une fonction XMLout()
- x **Dans le même registre, XML::Smart**



Agenda



- x A propos de XML
- x Que faire avec du XML ?
- x TIMTOWTDI, Perl oblige !
- x Parser pour les nuls
- x Cuisine & dépendances
- x XML::Simple
- x **XML::LibXML**
- x XML::SAX
- x XML::Twig
- x Ecrire du XML
- x On ze ouaibe again



DOM / M.I.¹

- x **Document**

- x `<?xml ... ?>`

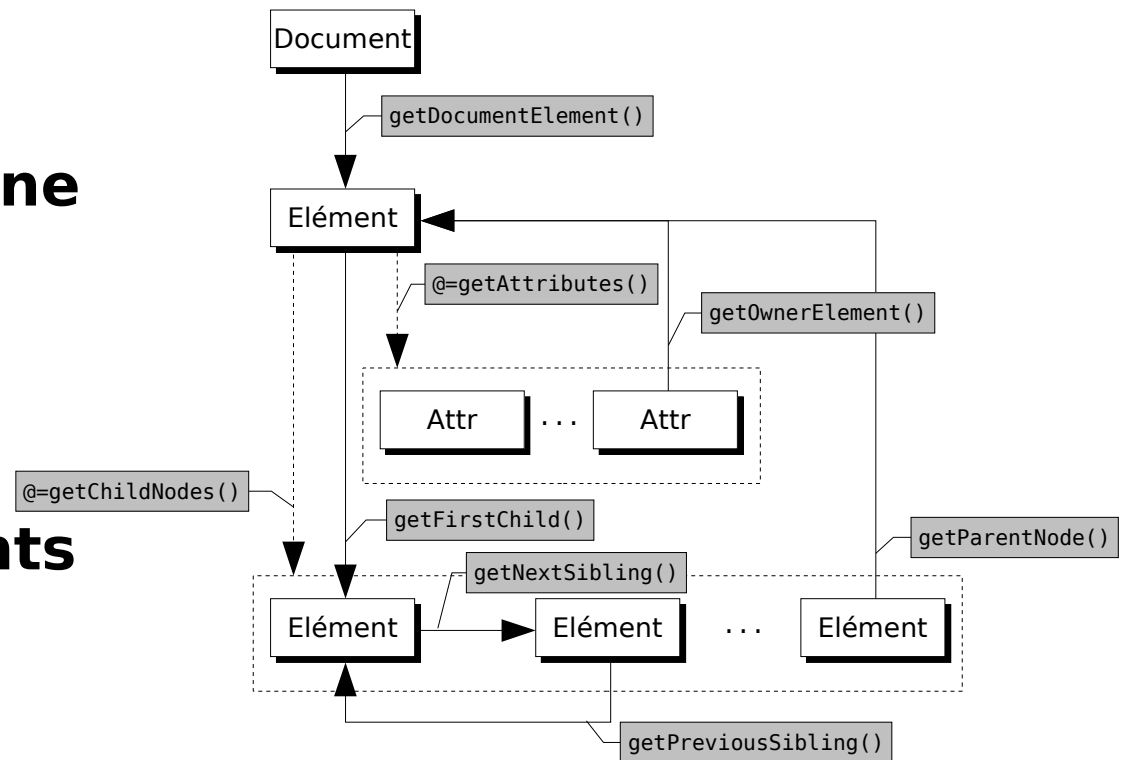
- x **Un seul élément racine**

- x Attributs
 - x Pas de pairs (“siblings”)
 - x Descendants

- x **Puis d'autres éléments**

- x Attributs
 - x Pairs
 - x Descendants

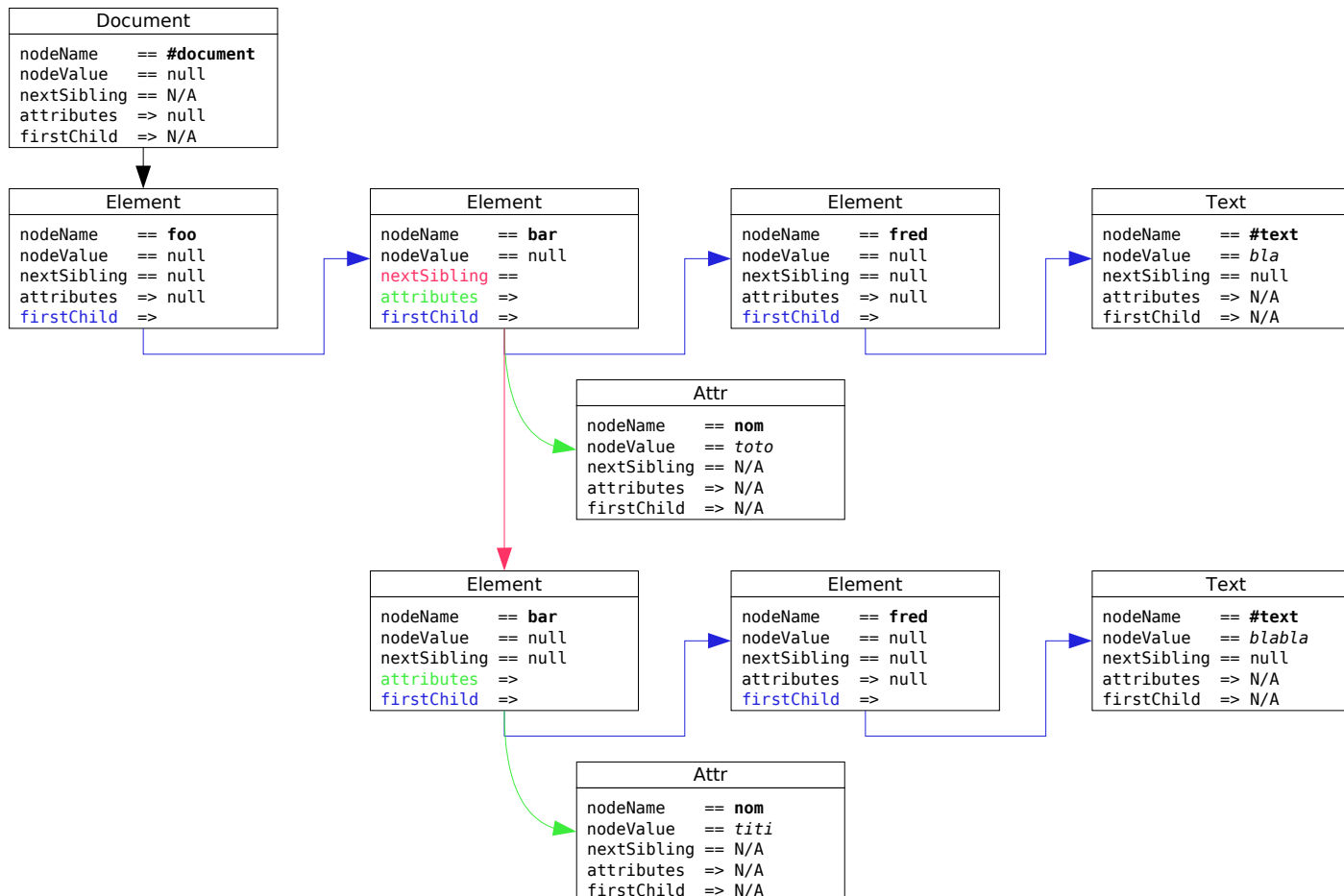
- x **Etc.**





DOM / M.I.²

x Représentation en mémoire du fil rouge





DOM / P.V.T.E.¹

x **Parser**

```
x use XML::LibXML;  
$p = XML::LibXML->new();  
$d = $p->parse_file( 'foo.xml' ); # parse_fh(), parse_string(), etc.
```

x **Valider (DTD)**

```
x $p->validate( 1 ); # avant l'appel à parse_*
```

x **Traiter¹ : se promener sur l'arbre**

x Structure *récurive* (LoL) où l'on va chercher (“pull”) les données

x Mais on ne peut pas utiliser Data::Dumper (à cause du back-end)

```
x $r = $d->getDocumentElement(); # obtenir l'élément racine du document  
dump_node( $r );  
for my $n1 ( $r->getChildNodes ) { # ze itérateur  
    dump_node( $n1, 1 );  
    for my $n2 ( $n1->getChildNodes ) {  
        dump_node( $n2, 2 );  
        # etc.  
    }  
}
```



DOM / P.V.T.E.²

x Traiter¹ (suite)

x Définition de la fonction `dump_node()`

```
x sub dump_node {  
    my( $n, $i ) = @_;  
  
    print + ( ' ' x $i ) . $n->getName . ' [' . $n->nodeType . ' ]';  
    if ( $n->hasAttributes ) {  
        print ' (' .  
            join( ', ', map( { $_->getName . '=' . $_->getValue } $n->getAttributes ) ) .  
            ')';  
    }  
    if ( $n->nodeType == 3 ) { # TEXT_NODE  
        print ' "' . $n->data . '"';  
    }  
    print "\n";  
}
```

x Définition de la fonction récursive `dump_tree()`

```
x sub dump_tree {  
    my( $n, $i ) = @_; # "my $i" puis "our $i" dans la boucle  
  
    dump_tree_loop( @_ );  
}
```



DOM / P.V.T.E.³

x **Traiter¹ (suite)**

x Fonction `dump_tree()` : (suite & fin)

```
x sub dump_tree_loop {  
    my $n = shift;  
    our $i = shift;  
  
    dump_node( $n, $i );  
    if ( my @n = $n->getChildNodes ) {  
        dump_tree_loop( $_, ++$i ) for @n;  
        $i--;  
    }  
}
```

x `dump_tree($r)` donne le résultat suivant

```
x foo [1]  
    bar [1] (nom=toto)  
        fred [1]  
            text [3] "bla"  
    bar [1] (nom=titi)  
        fred [1]  
            text [3] "blabla"
```



DOM / P.V.T.E.⁴

x **Traiter² : chercher des noeuds**

- x Parmi les enfants

- x `dump_node($_) for $r->getChildrenByTagName('bar');` # ne marche pas avec 'fred'

- x Parmi les descendants

- x `dump_node($_) for $r->getElementsByTagName('fred');`

x **Traiter³ : chercher des noeuds avec XPath**

- x Parmi les enfants

- x `dump_node($_) for $r->findnodes('bar');` # =~ getChildrenByTagName

- x Parmi les descendants

- x `dump_node($_) for $r->findnodes('//fred');` # =~ getElementsByTagName

x **Les axes de recherche sont différents**

- x En effet, ("bar" == "/foo/bar" == "/foo/bar/.") != "//fred"



DOM / P.V.T.E.⁵

x **Traiter⁴ : insérer un noeud**

x Par exemple

```
x $n = $r->addChild( '', 'bar' );           # ajoute un fils à la racine
  $n->setAttribute( 'nom', 'tutu' );        # définit un attribut de ce fils
  $n->appendChild( 'fred', 'blablabla' );  # ajoute un fils au fils
```

x **Ecrire**

x Par exemple

```
x print $d->toString( 1 ); # $format = 1
  $state = $d->ToFile( $filename, $format );
  $state = $d->toFH( $fh, $format );
```



DOM / XPath¹

x Chemins et prédicats

- x Choix de noeuds par un *chemin*

- x `/foo/bar/fred`

- x Ils peuvent aussi être choisis par leurs attributs ou contenu

- x `/foo/bar[@nom='toto']/fred` # noeud "fred" fils de bar nommé toto

- x `//fred[.='bla']` # noeud "fred" dont le contenu est "bla"

- x Racine, noeud courant, ancêtre et descendants (cf "axes")

- x `/` (racine) . (noeud courant) .. (père) et `//` (fils, petit-fils, etc.)

- x Jokers

- x `*` (éléments) `@*` (attributs)

- x Les *prédicats* (entre []) filtrent les chemins

- x Définition d'opérateurs booléens et de fonctions diverses

- x Comme `and/or/not`, `=/!=`, `local-name()`, `contains()/substring()`, etc.



DOM / XPath²

- x **Implémenté par la fonction findnodes()**
- x **En pratique, faire attention à q{} et qq{}**
 - x Utiliser \@ dans "" si besoin d'une expression variable
 - x De même :
 - x `'//bar[@nom=\'toto\']'` == `q{//bar[@nom='toto']}`
 - x Requêtes et unilignes, même combat !
 - x Escaper l'inescapable...
- x **En vrac...**
 - x `/foo/bar` == `/foo/bar/.` == `/foo/*[local-name()='bar']`
 - x `//fred` == `descendant-or-self::fred`
 - x `//*[text()]` == tous les noeuds textes du document

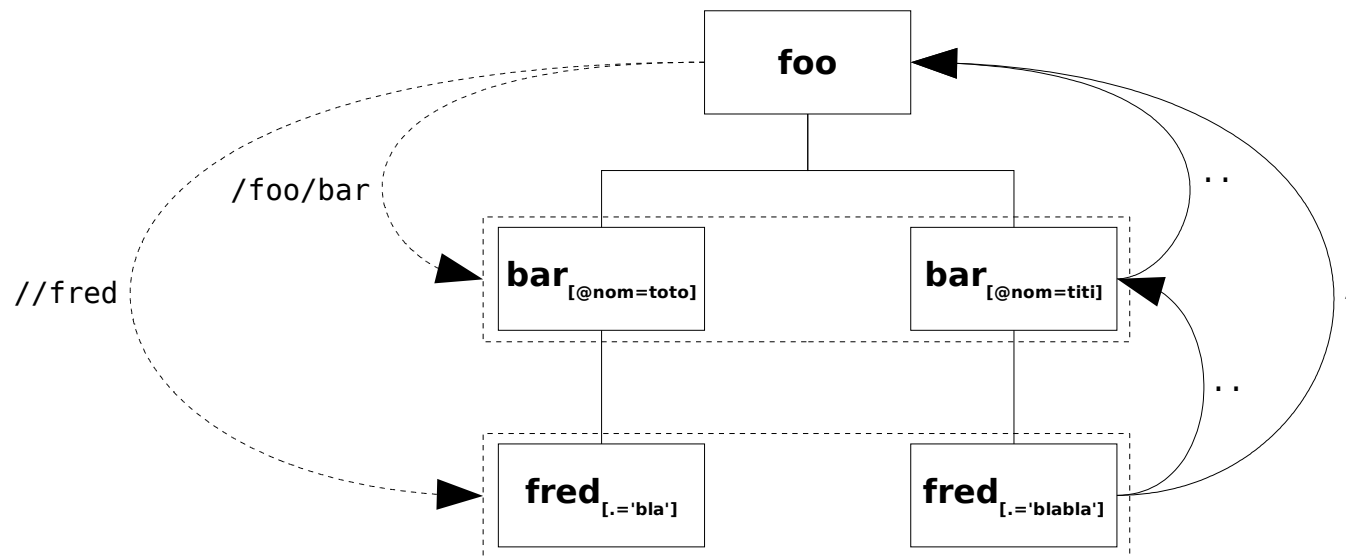


DOM / XPath³

- x **Les axes de recherche**

- x 13 sortes : self (.), child, descendant, descendant-or-self (//), parent (..), ancestor, ancestor-or-self, preceding, following, preceding-sibling, following-sibling, attribute (@) et namespace

- x **Sur notre document**





Agenda



- x A propos de XML
- x Que faire avec du XML ?
- x TIMTOWTDI, Perl oblige !
- x Parser pour les nuls
- x Cuisine & dépendances
- x XML::Simple
- x XML::LibXML
- x **XML::SAX**
- x XML::Twig
- x Ecrire du XML
- x **On ze ouaibe again**



XML::SAX¹

- x **Simple API for XML**
 - x API normée, SAX level 2
- x **Cf présentation des Journées Perl 2004**
 - x Construction de structures de données à partir de flux
- x **Le document devient un flux d'évènements**
 - x Début de balise (+ définition d'attributs)
 - x Caractères
 - x Fin de balise
- x **Déclenchant des fonctions (“callbacks”)**
 - x Les attributs et caractères sont fournis (“pushed”) par les parsers
 - x Sous forme de hash passée en paramètre aux callbacks



XML::SAX²

- x **Avantages**
 - x Normalisé et reconnu par de multiples langages
 - x Faible consommation de mémoire
 - x Un front-end unique mais plusieurs back-end
 - x Mais DOM aussi (dépend en fait de l'implémentation)
 - x Permet de filtrer les données à la volée

- x **Inconvénients**
 - x Pas d'accès aléatoire
 - x Pas de requêtes complexes "à la" XPath
 - x Pas d'accès à la DTD
 - x Peut devenir très compliqué à gérer



XML::SAX³

- x **La norme SAX définit des types d'évènements**

- x start_document, end_document
- x start_element, end_element
- x characters
- x start_cdata, end_cdata
- x processing_instruction
- x Etc. Pour une liste exhaustive :
 - x % perldoc XML::SAX::Base

- x **Noms des méthodes d'une classe "maison"**

- x Sous-classe de XML::SAX::Base
- x

```
package MySAXHandler;  
use XML::SAX::Base;  
@ISA = qw( XML::SAX::Base );
```



XML::SAX⁴

x Exemples de méthodes

```
x sub start_element {
    my( $self, $h ) = @_;
    print "nom : $h->{Name}\n" .
        map { "  $_=$h->{Attributes}->{$_}\n" } for keys %{$h->{Attributes}};
}
sub characters {
    my( $self, $h ) = @_;
    print "texte : $h->{Data}\n";
}
```

x Notion de “parser factory”

x Fabrique un parser avec la classe de callbacks et un back-end

```
x use XML::SAX;      # liste des BE (SAX.ini) : @{$XML::SAX->parsers}
use MySAXHandler; # choix BE : $XML::SAX::ParserPackage = "XML::SAX::Expat (0.72)";

$parser = XML::SAX::ParserFactory->parser( Handler => $id = MySAXHandler->new );
```

x Invoquer le parser déclenche les callbacks le long du flux

```
x open( $fh, 'foo.xml' );
$parser->parsefile( $fh );
close $fh;
```



XML::SAX⁵

x Document

```
x <foo>
x   <bar nom="toto">
x     <fred>
x       bla
x     </fred>
x   </bar>
x   <bar nom="titi">
x     <fred>
x       blabla
x     </fred>
x   </bar>
x </foo>
```

x Evènements

```
x $id->start_element( $h )
x $id->start_element( $h )
x $id->start_element( $h )
x $id->characters( $h )
x $id->end_element( $h )
x $id->end_element( $h )
x $id->start_element( $h )
x $id->start_element( $h )
x $id->characters( $h )
x $id->end_element( $h )
x $id->end_element( $h )
x $id->end_element( $h )
x $id->end_element( $h )
```

\$h->{Name}

\$h->{Name}
%{\$h->{Attributes}}

\$h->{Data}

\$h->{Name}



XML::SAX⁶

- x **Permet de construire des pipelines**

- x Chaque machine est un étage du filtre

- x

```
use XML::SAX::Machines qw( Pipeline );
$machine = Pipeline( MySAXHandler1 => MySAXHandler2 => \$s ); # ou XML::SAX::Writer
$machine->parsefile( 'foo.xml' );
```

- x Permet d'inhiber ou de générer des évènements

- x

```
if ( $h->{Name} eq 'fred' ) {
    $self->SUPER::characters( { Data => "Fred était là !" } );
}
else {
    $self->SUPER::start_element( $h );
}
```

- x Supporte les “taps” (~ moniteur) et plus encore (“manifold”)

- x **RTFM**

- x % perldoc XML::SAX::Intro

- x Articles de K. Hampton sur www.xml.com



Agenda



- x A propos de XML
- x Que faire avec du XML ?
- x TIMTOWTDI, Perl oblige !
- x Parser pour les nuls
- x Cuisine & dépendances
- x XML::Simple
- x XML::LibXML
- x XML::SAX
- x **XML::Twig**
- x Ecrire du XML
- x On ze ouaibe again



XML::Twig¹

- x **3 modes de fonctionnement**
 - x Arbre
 - x Morceaux par morceaux (twig_handlers)
 - x Une section seulement (twig_roots)
- x **Non standard, et alors ?**
 - x Des fonctions utiles : copy/cut/paste/move/replace, etc.
 - x Compatible avec XML::Simple : `$t->simplify()`
- x **La tronçonneuse Suisse du XML**
 - x Imbattable pour l'écriture de filtres “à l'arrache” (outside_roots)



XML::Twig²

- x **Combo flux / arbre roulaize**
 - x Si le document est trop gros pour DOM (expansion ~ 10x)
 - x Ou allergie au SAX (bon d'accord, SAX pas toujours possible)
 - x En fait, permet de gérer un flux de *branches*
 - x Sélection possible avec des expressions *voisines* de XPath
 - x Chaque branche génère une arborescence temporaire
 - x On peut appliquer toute la palette de fonctions sur la branche
 - x Possibilité de travailler sur des branches décorréllées
 - x Mais pas de branches imbriquées, hélas !



XML::Twig³

x Exemple de filtre

x `use XML::Twig;`

```
$twig = XML::Twig::new ( twig_print_outside_roots => 1,  
    twig_roots => { 'bar[@nom="toto"]' => \&echo_bar } );  
$twig->parsefile( 'foo.xml' );
```

```
sub echo_bar {  
    my( $twig, $t ) = @_;  
  
    $t->set_att( nom => $t->att( 'nom' ) x 2 );  
    $t->set_text( $_->text x 2 ) for $t->descendants( 'fred' );  
  
    $t->print;  
}
```

x Remplace `//bar[@nom="toto"]` (seulement) par

x `<bar nom="totototo"><fred>blablabla</fred></bar>`

x Jouer à cache-cache avec les branches

x Extraire une branche : pas de `twig_print_outside_roots => 1`

x Exclure une branche : pas de `$t->print`



Ecrire du XML



- x A propos de XML
- x Que faire avec du XML ?
- x TIMTOWTDI, Perl oblige !
- x Parser pour les nuls
- x Cuisine & dépendances
- x XML::Simple
- x XML::LibXML
- x XML::SAX
- x XML::Twig
- x **Ecrire du XML**
- x On ze ouaibe again



Ecrire du XML¹

- x **Le bon vieux print()**

- x Ne crée pas de dépendance, offre un contrôle + fin de la sortie
- x Pas de vérification de bien-formé à la volée, + dur à maintenir

- x **XML::Writer**

- x L'objet à écrire est construit à l'aide d'une API

- x

```
use XML::Writer;
$w = XML::Writer::new( OUTPUT => $fh, DATA_MODE => 1, DATA_INDENT => 2 );
$w->startTag( 'foo' );
$w->startTag( 'bar', 'nom' => 'toto' );
$w->dataElement( 'fred', 'bla' );
$w->endTag( 'bar' ); # argument facultatif (remplacé par le contexte sinon)
$w->endTag( 'foo' );
# idem pour bar(titi)
$w->end(); # écrit le fichier
```

- x **XML::SAX::Writer**

- x Dernier étage (“exhaust”) d'un pipeline



Agenda



- x A propos de XML
- x Que faire avec du XML ?
- x TIMTOWTDI, Perl oblige !
- x Parser pour les nuls
- x Cuisine & dépendances
- x XML::Simple
- x XML::LibXML
- x XML::SAX
- x XML::Twig
- x Ecrire du XML
- x **On ze ouaibe again**



On ze ouaibe again¹

x **XML**

- x XML..... <http://www.w3.org/XML/>
..... <http://www.xml.com/>
- x DOM..... <http://www.w3schools.org/>
- x SAX..... <http://www.saxproject.org/>
- x XML Schemas..... <http://www.w3schools.com/schema/>
- x XSL..... <http://www.w3.org/Style/XSL/>
- x XPath..... <http://www.w3.org/TR/xpath/>

x **Back-end**

- x <eXpat/>..... <http://expat.sourceforge.net/>
- x Gnome libxml2..... <http://xmlsoft.org/>



On ze ouaibe again²

- x **Perl & XML**

- x Perl & XML..... <http://xml.com/pub/q/perlxml/>
- x The Perl-XML FAQ.....<http://perl-xml.sourceforge.net/faq/>
- x XML::Twig.....<http://xmlltwig.com/>
..... http://www.xmlltwig.com/article/index_wtr.html
..... http://xmlltwig.com/article/ways_to_rome_2/
- x CPAN..... <http://search.cpan.org>

- x **Livres**

- x XML in a nutshell
- x XML précis et concis
- x Perl cookbook (2nd edition)



FIXMEs

- x **DTD et consorts**
- x **Validation, parser validant ou non**
- x **Quid fixodiet ipsos fixodes?**