



XML::DT - a Perl down translation module



José Carlos Ramalho

jcr@di.uminho.pt

José João Dias de Almeida

jj@di.uminho.pt



Table of Contents

- XML::DT, what?
- Context of use
- Feature description
- Behind the scene
- Examples
- Future developments



What was the motivation?

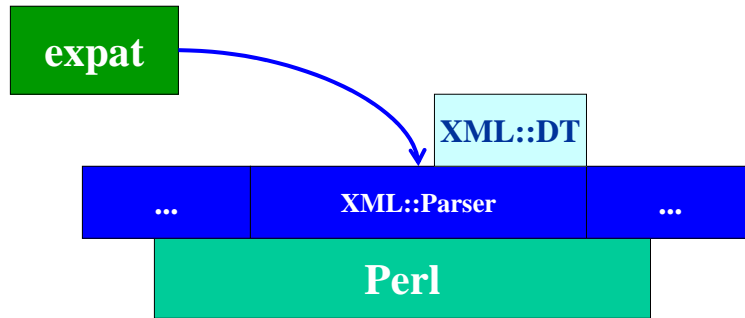
- project GEIRA
 - 30 museums; several libraries; 1 national archive; ...
 - XML documents are produced everyday:
 - Electronic news system
 - Catalog construction
 - Source documents
 - we have to generate several different formats for the same document
- Control over ...



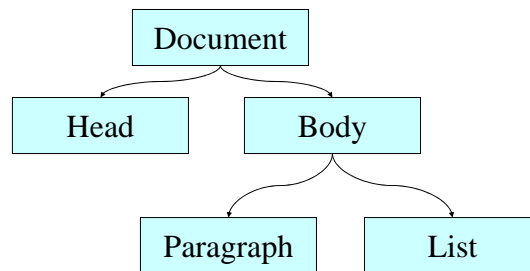
XML::DT, what?

- a simple tool
 - easy to learn
 - with unlimited power !?
- to process down translations
- ... going towards a transformation tool

XML::DT architecture



Processing with XML::DT



$\text{Process}(\text{Document}) = f(\text{Process}(\text{Head}), \text{Process}(g(\text{Process}(\text{Paragraph}), \text{Process}(\text{List}))))$

With XML::DT:

- f, g = concatenation
- Expected result of Process is a string





Example 1: XML file

```
<?xml version="1.0" encoding="ISO-8859-1">
<article>
  <title>The XML Down Translator</title>
  <author>J. João Almeida</author>
  <author>José Carlos Ramalho</author>
  <keyword>XML</keyword>
  <keyword>language processing</keyword>
  <keyword>perl</keyword>
  <abstract>
    Once upon a time ...
  </abstract>
</article>
```



Example 1: DT file

```
1 #!/usr/bin/perl
2 use XML::DT ;
3 my $filename = shift;

4 %handler=(
5   '-outputenc' => 'ISO-8859-1',
6   '-default'   => sub{""},
7   'title'      => sub{"<b>${c}</b>"},
8   'author'     => sub{" <i>${c}</i>"},
9   'article'    => sub{"${c}<br>"}
10 );
11 print dt($filename,%handler);
```

Example 1: DT file

```
perl ex1.pl art.xml
```

```
<b>The XML Down Translator <
  <i> J.J. Almeida </i>
  <i> J.C. Ramalho </i>
<br>
```

\$c - processed content
\$q - element gi
\${AttrName} - value of attribute
dt(arg1, arg2) - main down
translation function

```
5   '-outputenc' => 'utf-8',
6   '-default'   => sub{""},
7   'title'     => sub{"<b>$c</b>"},
8   'author'    => sub{" <i>$c</i>"},
9   'article'   => sub{"$c<br>"}
10  );
11  print dt($filename,%handler);
```



dt(arg1, arg2) function

- **arg1** is the filename of the XML file or a string with XML content
- **arg2** is a structure of the form:
pattern => action
- **patterns** available
 - '-default' - matches every element not matched in any other pattern
 - '-outputenc' - by default is UTF8
 - 'element-gi' - triggers action for elements with that gi
 - '-pcdata' - selects #PCDATA in mixed contents
 - '-end' - processing to be applied to the transformed tree





Is it hard to write? ...skelgen.pl

- Ex: perl skelgen.pl art.xml

```
#!/usr/bin/perl
use XML::DT ;
my $filename = shift;

%handler=(
# '-outputenc' => 'ISO-8859-1',
# '-default' => sub{"<$q>$c</$q>"},
'title' => sub{"$q:$c"},
'author' => sub{"$q:$c"},
'article' => sub{"$q:$c"},
'abstract' => sub{"$q:$c"},
'keyword' => sub{"$q:$c"},
);
print dt($filename,%handler);
```



skelgen.pl

- Generates XML::DT
- It is programmed with
- It shows a different use
- two

Analysis:

```
#!/usr/bin/perl
use XML::DT ;
my $filename = shift;
$xml=( '-default' =>
for (keys %handler) {
dt($filename,%xml);
```

Printing:

```
print <<'END';
#!/usr/bin/perl
use XML::DT ;
my $filename = shift;

%handler=(
# '-outputenc' => 'ISO-8859-1',
# '-default' => sub{"<$q>$c</$q>"},
END

for $name (keys %element){
print " '$name' => sub{"\".$q:.$c\""},";
print '# remember $v{',
join('),$v{',keys % {$att{$name}}),
}' if $att{$name};
print "\n";
}
print <<'END';
);
print dt($filename,%handler);
END
```



skelgen.pl: future developments

- script will become a module function
- It will have a behavior switch to enable incremental processing:
 - if you add something to the structure of your XML file you do not need to hand code the new elements



Ex2: Proceedings List of papers

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<proceedings>
  <title>XML Europe 99</title>
  <chair>Pam</chair>
  <abstract>
    Once upon a time in Granada ...
  </abstract>

  <article file="art1.xml"/>
  <article file="art2.xml"/>
  <article file="art3.xml"/>
</proceedings>
```

Proceedings.pl

```
<H1>Proceedings:
XML Europe 99</H1>
<H2>Chair: Pam</H2>
<b>The XML Down Translator</b>
  <I>J. João Almeida</I><BR>
  <I>JosÁ© Carlos Ramalho</I>
<P>
<b>The XML Parser</b><BR>
  <I>Clark Cooper</I><BR>
  <I>Larry Wall</I><BR>
<P>
<b>The expat library</b><BR>
  <I>James Clark</I><BR>
<P>
```

```
#!/usr/bin/perl
use XML::DT ;
my $filename = shift;

%p_proc=(
'-outputenc' => 'ISO-8859-1',
'-default' => sub{"$c"},
'proceedings' => sub{"<H1>Proceedings: $c"},
'title' => sub{if(inctxt('proceedings')) {"$c</H1>"}
else {""}},
'article' => sub{ dt($v{file}, %p_art) },
'abstract' => sub{""},
'chair' => sub{"<H2>Chair: $c</H2>"},
);

%p_art=(
'-default' => sub{""},
'title' => sub{"<b>$c</b><BR>"},
'author' => sub{" <I>$c</I><BR>"},
'article' => sub{"<P>$c<P>"},
);

print dt($filename,%p_proc);
```



Proceedings.pl

dt can be used in a functional way

- enabling subdocument processing
- nested processing
- processor combination

```
#!/usr/bin/perl
use XML::DT ;
my $filename = shift;

%p_proc=(
'-outputenc' => 'ISO-8859-1',
'-default' => sub{"$c"},
'proceedings' => sub{"<H1>Proceedings: $c"},
'title' => sub{if(inctxt('proceedings')) {"$c</H1>"}
else {""}},
'article' => sub{ dt($v{file}, %p_art) },
'abstract' => sub{""},
'chair' => sub{"<H2>Chair: $c</H2>"},
);

%p_art=(
'-default' => sub{""},
'title' => sub{"<b>$c</b><BR>"},
'author' => sub{" <I>$c</I><BR>"},
'article' => sub{"<P>$c<P>"},
);

print dt($filename,%p_proc);
```



Context dependent processing

- `ctxt(number)`
 - Suppose we have: `CONTEXT = "article/header/title"`
 - `ctxt(0)` matches "title" (\$q)
 - `ctxt(1)` matches "header"
 - `ctxt(2)` matches "article"
- `inctxt(pattern)`
 - `inctxt('art.*')` true if current element is under article and article is the top element
 - `inctxt('proceedings | article')` true if element is son of proceedings or article

← match

LIFO structure

article/header/title



Ex3: Adding a

What do we have? A set

```
<?xml version="1.0" encoding="utf-8" >
<article>
  <title>The XML Down Translator</title>
  <author>J. João Almeida</author>
  <author>José Carlos Ramalho</author>
  <keyword>XML</keyword>
  <keyword>language processing</keyword>
  <keyword>perl</keyword>
  <abstract>
    Once upon a time ...
  </abstract>
</article>
```

What do we want?

Proceedings: XML Europe 99

Chair: Pam

The XML Down Translator

J. João Almeida

JosÁ© Carlos Ramalho

The XML Parser

Clark Cooper

Larry Wall

The expat library

James Clark

Keyword index

C++

The expat library

XML

The XML Down Translator

The XML Parser

The expat library

...

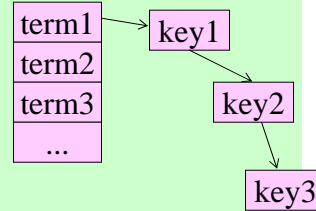


Ex3: How?

```

sub mkKeyInd{
  my $r = "<H2>Keyword index</H2>";
  for $term (sort keys %ind)
  { $r .= "<P> <B>$term</B> $ind{$term}"; }
  $r
}
%p_proc=( ...
  'proceedings' => sub{"<H1>Proceedings: $c" . mkKeyInd()},
... );
%p_art=( ...
  'title' => sub{ $tit = $c; " <b>$c</b><BR>"; },
  'keyword' => sub{ $ind{$c} .= "<BR> $tit"; "" }, ...
);

```



The main algorithm

```

dt( document-handler, processor ) =
  let tree = Parse(document-handler)
  in process( tree, processor)
  || processor{'-end'} (process( tree, processor))

process(PCDATA(p), processor) = p || processor{'-pcdata'}(p)

process( element( e, sons ), processor ) =
  let args = concatenate( [ process( x, processor ) | x ← sons ] )
  in if( e in domain(processor) then processor[e](args)
  else processor['-default'](args)

```





Last example: gcapaper2tex.pl

```
%handler=(...
\ '-pdata' => sub{
  • if(inctxt('(SECTION|SUBSEC1')) { $c =~ s/[s\n]+/ /g; $c }
    $c },
  ...
  • "TITLE" => sub{
    if(inctxt('SECTION')){"\section{$c}"}
    elsif(inctxt('SUBSEC1')){"\subsection{$c}"}
    else {"\title{$c}"} },
  'AUTHOR' => sub{ push @aut, $c ; "" },
  'ABSTRACT' => sub{
    sprintf("author{%s}\maketitle\begin{abstract}%s\end{abstract}",
      join ("and", @aut) , $c ) },
  'XREF' => sub{"\cite{$v{REFLOC}}"},
  ... );
```

N and



New developments

- To Enable dt to create a data structure instead of a string:
 - a mapping: elementgi → content
- for a certain domain of actions
 - dt will generate a DSSSL/XSL specification
- To create the \$utc
 - un-transformed-content

Resources

- Web page
 - <http://www.di.uminho.pt/~jj/perl/XML/DT.html>
 - last version available for download
 - examples
 - documentation

