

Active Tags: an XML System for Native XML Programming

Philippe Poulard

► **To cite this version:**

Philippe Poulard. Active Tags: an XML System for Native XML Programming. XML 2006, Dec 2006, Boston, United States. inria-00481929

HAL Id: inria-00481929

<https://hal.inria.fr/inria-00481929>

Submitted on 7 May 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Active Tags

an XML System
for Native XML Programming

- A set of specifications (language/platform independant)
- A general-purpose framework
- Batch, Web applications, embedded in an application
- Looks like XSLT/XQuery/Jelly/ASP/PHP/JSP/JSTL/Ant...
- Use XML tags as actions to perform
- XPath-centric
- Can query various data sources (RDBMS, LDAP, XML native databases)
- Several libraries (modules) can be used

<http://disc.inria.fr/perso/philippe.poulard/xml/active-tags/>

Specifications :

Implementation :

<http://reflex.gforge.inria.fr>

The
Active Tags
engine, in Java

Reflex

Unix : everything is a file

OOP : everything is object

Active Tags : everything is XML

→ A powerful and complete system based upon XML technologies

Cohabitation of several tag libraries

Features are not overlapping each others

→ Chosen by INRIA

Used in production

→ Concise/powerful

Modular/extensible/maintenable

Easy to use (if you know XML/XPath ☺)

→ High-level API

General purpose

Mask details (DOM,SAX)

Connexions with the environment (SYSTEM, I/O, SQL, Web...)

master spec → Active Tags

core modules

- Active Catalog
- Active Datatypes
- the Active Schema Language
- the XML Control Language
- an Extension of the XML Processor

- Active Sheet
- Active Document
- Active Material

Standard modules

- I/O module
- SYTEM module
- Web module
- RDBMS module

Application

- the XUnit framework

Some examples...

- A simple example → the basics
- Handling XML documents → mask DOM and SAX differences
- Pipelines → filtering a SAX stream with XPath patterns
- Combining modules → module cohabitation
- A Web application → introduction of X-Operable Objects
- Modularization → macro creation
- XUnit → testing XML

with XCL : the XML Contol Language

a convenient root

```

<?xml version="1.0" encoding="iso-8859-1"?>
<xcl:active-sheet
  xmlns:xcl="http://www.inria.fr/xml/active-tags/xcl">
  <xcl:parse name="myDoc"
    source="file:///path/to/document.xml"/>
  <xcl:parse-stylesheet name="myXslt"
    source="file:///path/to/stylesheet.xsl"/>
  <xcl:transform source="{ $myDoc }"
    stylesheet="{ $myXslt }"
    output="file:///path/to/output.html"/>
</xcl:active-sheet>

```

instruction
that creates
the property
named « myDoc »

XPath expression

Like « AVT » in XSLT
but :

- can occur in text content
- is not cast to string
- can refer to objects

Shell-fashioned references

Comparison with Ant/Jelly/JSTL...

```

<document time="{ $now }">
  Welcome { $user.name } to Jelly!
</document>

```

Can't extend to path expressions ☹️ → { \$now/@year }

Can't compute expressions ☹️, can only refer to values

→ { \$price * \$discount }

the engine "knows" which tag is an action and which tag is a literal
→ an Active Catalog is plugged to the engine

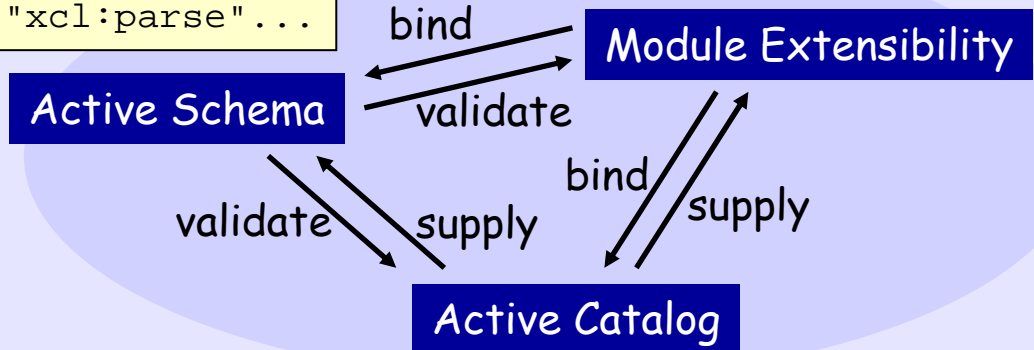
compatible ↪ XML catalog : map URI → URIs
 Active Catalog : map URI + selector → resources
 (URI, module, schema, catalog, active-sheet...)
 resource management facilities (caching policy)

How is it resolved ?

Active tag → `<xcl:parse>`

```
<asl:schema target="xcl">
  <asl:element name="xcl:parse"...
```

```
<exp:module target="xcl">
  <exp:element name="xcl:parse"...
```



```
<cat:catalog>
<exp:module>
<asl:schema>
```

are resolved in the same manner

```
<cat:catalog>
  <cat:uri name="http://www.inria.fr..."
```


`<xcl:parse>` parse an XML document or fragment
`<xcl:document>` create an XML document or fragment

} can be SAX or DOM

- SAX to DOM
- DOM to SAX

```
<xcl:parse name="myDom" type="DOM" />
<xcl:document name="mySax" type="SAX">
  <wrapper>
    { $myDom }
  </wrapper>
</xcl:document>
```

deferred processing

can be accessed with XPath

can be filtered with XPath patterns

```
<xcl:parse-stylesheet name="myXslt" source="file:///path/to/stylesheet.xsl"/>
<xcl:parse-filter name="xinclude" source="http://www.w3.org/2001/XInclude"/>
<xcl:parse name="myDoc" type="SAX" source="file:///path/to/document.xml"/>
<xcl:filter name="included" source="{ $myDoc }" filter="{ $xininclude }"/>
<xcl:transform source="{ $included }" stylesheet="{ $myXslt }"
  output="file:///path/to/output.html"/>
```

Custom filters : XCL filters

- a subset of tags for XPath-based filters
- can process extra-large documents with SAX

```
<xcl:parse-filter>
<xcl:filter>
<xcl:rule> ← XPath pattern
<xcl:forward> ← can specify several "channels"
<xcl:apply-rules>
```

Built-in filters

- XInclude filter
 - Line reader
 - Tokenizer
- text to XML {
- ↑
regex

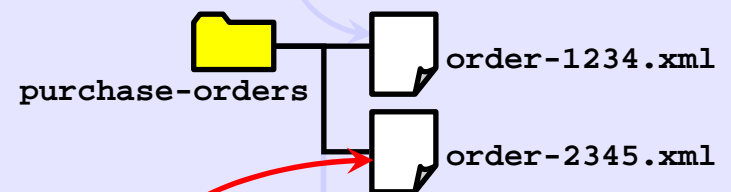
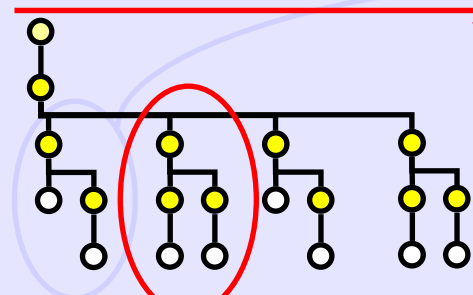
```

<xcl:parse name="allPo"
  source="file:///path/to/purchase-orders.xml" type="SAX"/>
<xcl:filter name="poSplitter" source="{ $allPo }">
  <xcl:rule pattern="/purchase-orders/order">
    <xcl:document name="order" type="SAX">
      <xcl:forward channel="order">
        <xcl:apply-rules/>
      </xcl:forward>
    </xcl:document>
    <xcl:transform
      output="file:///path/to/purchase-orders/order-{ @id }.xml"
      source="{ $order }"/>
  </xcl:rule>
</xcl:filter>
<xcl:transform output="{ $sys:null }" source="{ $poSplitter }"/>
  
```

inline definition

could also be DOM

SAX ("#main" channel)



("order" channel) SAX

serialize

modules URIs

literal

the engine "knows" which tag is active and which tag is a literal

AVT in content

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xcl:active-sheet
  xmlns:sys="http://www.inria.fr/xml/active-tags/sys"
  xmlns:io="http://www.inria.fr/xml/active-tags/io"
  xmlns:xcl="http://www.inria.fr/xml/active-tags/xcl">
  <xcl:set name="dir"
    value="{ io:file('file:///path/to/dir/') }">
  <xcl:document name="mergeAll" type="SAX">
    <mergeAll>
      <xcl:for-each name="myFile"
        select="{ $dir/*[@io:is-file] }">
        <xcl:parse name="myDoc" type="SAX"
          source="{ $myFile }"/>
        { $myDoc }
      </xcl:for-each>
    </mergeAll>
  </xcl:document>
  <xcl:transform
    source="{ $mergeAll }"
    output="{ $sys:out }"/>
</xcl:active-sheet>
```

predefined property the « SYSTEM » module

No stylesheet = copy

Active tags : `<xcl:parse>`

XPath functions : `io:file()`

Predefined properties : `$sys:out`

Foreign attributes : `@xcl:version` ← directives

Data types : `io:x-file` ← x- for « XML friendly » objects

- Can apply XPath expressions
- Can be X-updated

```
$dir//*[@io:is-file]
```

```
$dir/*[@io:is-file][@io:extension='xml']
```



`io:x-file` type definition

In the documentation of the I/O module

<code>name()</code>	<code>xs:QName</code>	The name of the file
<code>parent::</code>	<code>io:x-file</code>	The parent directory of this file
<code>child::</code>	<code>adt:list of io:x-file</code>	The files contained in this directory
<code>@io:length</code>	<code>xs:integer</code>	The length of this file
<code>@io:is-file</code>	<code>xs:boolean</code>	Indicates whether or not this is a file.
<code>@io:last-modified</code>	<code>xs:date</code>	The last date when this file was modified

```
<xcl:update referent="{ $dir/@io:last-modified }"  
    operand="{ $newDate }" />
```

```
<xcl:update>  
<xcl:rename>  
<xcl:append>  
...etc
```

incoming URL : http://www.acme.com/index.xml

```

<?xml version="1.0" encoding="iso-8859-1"?>
<web:service
  xmlns:web="http://www.inria.fr/xml/active-tags/web"
  xmlns:xcl="http://www.inria.fr/xml/active-tags/xcl">
  <web:mapping match="^(.+)\.html$"
    mime-type="text/html">
    <xcl:parse name="myDoc" source="web:/// { $web:match/node()[1] }.xml"/>
    <xcl:transform
      source="{ $myDoc }"
      stylesheet="web:///WEB-INF/transform.xsl"
      output="{ value( $web:response/@web:output ) }"/>
  </web:mapping>
  <web:mapping match="...">
    <!--other web stuff-->
  </web:mapping>
</web:service>

```

regexp

captured group

type : io:output

XML view

```

<web:response
  web:output="[io:output@189c036]"
  web:mime-type="text/html"/>

```

type : web:x-response

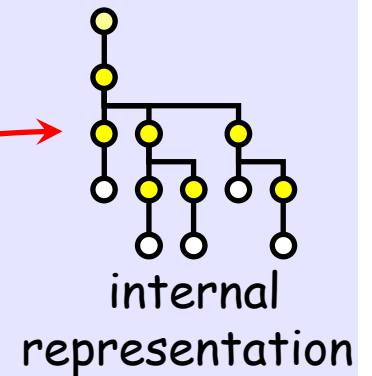
bound to an object

Objects type are identified by QNames like `web:x-response`,
`io:x-file`, `io:output`, `io:input`,
`xs:string`, `xs:integer`, `xml:document`, etc

```
<web:response
  web:output="[io:output@189c036]"
  web:mime-type="text/html">
  <Cache-Control>no-cache</Cache-Control>
  <Date>Tue, 15 Nov 1994 08:12:31 GMT</Date>
</web:response>
```

Not necessary represented with tags

Avoid
round-trip



Objects are not necessary representable in XML :

- what would be the XML representation of an instance of `io:input` or `io:output` ?

(I'm not talking about the content, but about the container, as an object)

Big objects, binary object : high-cost for XML encoding/decoding

Incomplete objects : the content is computed only when it is accessed
 (late binding)

incoming URL : http://www.acme.com/user.html?userName=Poulard

```
<?xml version="1.0" encoding="iso-8859-1"?>
<web:service
  xmlns:web="http://www.inria.fr/xml/active-tags/web"
  xmlns:xcl="http://www.inria.fr/xml/active-tags/xcl">
  <web:mapping match="^/user\.html$"
    <xcl:set name="ldap"
      value="ldap://ldap.acme.org:9009/dc=acme,dc=org
        ??sub?(&(sn~={ $web:request/userName }))/>
    <xcl:parse name="dsml" source="{ $ldap }"/>
    <xcl:document name="userDoc"/>
    <user userName="{ $web:request/userName }">
      { $dsml }
    </user>
  </xcl:document>
  <xcl:transform
    source="{ $userDoc }"
    stylesheet="web:///WEB-INF/us
    output="{ value( $web:response
  </web:mapping>
</web:service>
```

```
<xcl:set name="ldap"
  value="ldap://ldap.acme.org:9009/dc=acme,dc=org
    ??sub?(&(sn~={ $web:request/userName }))/>
<xcl:parse name="dsml" source="{ $ldap }"/>
<xcl:document name="userDoc"/>
<user userName="{ $web:request/userName }">
  { $dsml }
</user>
</xcl:document>
```

Batch script

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xcl:active-sheet
  xmlns:sys="http://www.inria.fr/xml/active-tags/sys"
  xmlns:xcl="http://www.inria.fr/xml/active-tags/xcl">
  <xcl:set name="ldap"
    value="ldap://ldap.acme.org:9009/dc=acme,dc=org
      ??sub?(&(sn~={ $sys:env/userName }))/>
  <xcl:parse name="dsml" source="{ $ldap }"/>
  <xcl:document name="userDoc"/>
  <user userName="{ $sys:env/userName }">
    { $dsml }
  </user>
</xcl:document>
<xcl:transform source="{ $userDoc }"
  stylesheet="file:///path/to/userStylesheet.xsl"
  output="{ $sys:out }"/>
</xcl:active-sheet>
```

```
<xcl:set name="ldap"
  value="ldap://ldap.acme.org:9009/dc=acme,dc=org
    ??sub?(&(sn~={ $sys:env/userName }))/>
<xcl:parse name="dsml" source="{ $ldap }"/>
<xcl:document name="userDoc"/>
<user userName="{ $sys:env/userName }">
  { $dsml }
</user>
</xcl:document>
```

Active Tags

Web application

expose as a single tag
(externalization inside
a custom module)

Use/define a custom module

```
<?xml version="1.0" encoding="iso-8859-1"?>
<web:service
  xmlns:acme="http://tags.acme.org/usersInfo"
  xmlns:web="http://www.inria.fr/xml/active-tags/web"
  xmlns:xcl="http://www.inria.fr/xml/active-tags/xcl">
  <web:mapping match="^/user\.html$"
    <acme:user-info name="{ $web:request/userName }"/>
    <xcl:transform
      source="{ $acme:userInfo }"
      stylesheet="web:///WEB-INF/resources/userStylesheet.xsl"
      output="{ value( $web:response ) }"/>
  </web:mapping>
</web:service>
```

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xcl:active-sheet
  xmlns:acme="http://tags.acme.org/usersInfo"
  xmlns:sys="http://www.inria.fr/xml/active-tags/sys"
  xmlns:xcl="http://www.inria.fr/xml/active-tags/xcl">
  <acme:user-info name="{ $sys:env/userName }"/>
  <xcl:transform source="{ $acme:userInfo }"
    stylesheet="file:///path/to/userStylesheet.xsl"
    output="{ $sys:out }"/>
</xcl:active-sheet>
```

EXP : Extensible XML Processor

implementation of the tag
(« macro tag »)

```
<?xml version="1.0" encoding="iso-8859-1"?>
<exp:module target="acme"
  xmlns:acme="http://tags.acme.org/usersInfo"
  xmlns:exp="http://www.inria.fr/xml/active-tags/exp"
  xmlns:xcl="http://www.inria.fr/xml/active-tags/xcl">
  <exp:element name="acme:user-info">
    <xcl:set name="ldap"
      value="ldap://ldap.acme.org:9009/dc=acme,dc=org
        ??sub?(&(sn~={ $exp:params/@name } ) )"/>
    <xcl:parse name="dsml" source="{ $ldap }"/>
    <xcl:document name="userDoc" />
    <user userName="{ $exp:params/@name }"
      { $dsml }
    </user>
  </xcl:document>
  <exp:exports>
    <exp:export name="acme:userInfo" value="{userDoc}"/>
  </exp:exports>
</exp:element>
</exp:module>
```

exported property

XPath functions and predefined properties can also be defined with macros

A module made of
active tags

```

<xunit:test-case> ← set the boundaries of a test
<xunit:assert-boolean-equals>
<xunit:assert-number-equals> } report assertions
<xunit:assert-node-equals>
<xunit:merge-reports> ← merge test-cases report
...etc

```

Test suites for :

- active sheets
- individual XSLT templates
- Java classes (that are dealing with XML datas)

→ harness for tests suites

```

<xunit:test-case name="acme-test" label="Acme test">
  <!--stuff to test-->
  <acme:foo bar="do-it-like-this"/>
  <!--check the result-->
  <xcl:parse name="result" source="result.xml"/>
  <xcl:parse name="output-expected"
    source="file:///path/to/output-expected.xml"/>
  <!--check if they are equals-->
  <xunit:assert-node-equals result="{ $result }"
    expected="{ $output-expected/some[1]/result[2] }"/>
</xunit:test-case>

```

HTML report :

Acme test	Tests : 82	Errors : 1	Failures : 0
Node expected :	/some[1]/result[2]/foo[3]		
Result node :	/foo[3]		
Attribute existence :	Unexpected attribute bar="BAR"		

- A systemic consideration of XML technologies
 - each component focus on a well-defined problematic
 - component cooperation
- Allow to handle with ease XML datas
 - text to XML
 - SQL to XML
 - LDAP to XML
 - SAX, DOM, XCL filters and pipelines
- Extensible better than ever
 - Custom modules (macro tags, macro XPath functions)
- Viability
 - Lots of runnable examples and tips in RefleX
 - Already used in production at INRIA
 - Could be closer to XPath2/XQuery data model
 - Some features still experimental or incomplete

To go further :

- read carefully the slides !

`http://disc.inria.fr/perso/philippe.poulard/xml/active-tags.pdf`

- email-me

`Philippe.Poulard@sophia.inria.fr`

- discuss about Active Tags on the XML-dev list

- download the engine

`http://reflex.gforge.inria.fr`

- try the tutorials

- send me some money ☺

Questions ?