



Domteur

and the

XDO text to XML

parser language

by Ulrich Höhne, arxio GmbH

Content of this file is Copyright (c) 2006 arxio GmbH, Karlsruhe, www.arxio.de



Table of contents

Abstract	4
Status of this document	4
Copyrights and Trademarks	4
Introduction	5
Conventions and Styles	5
Terminology	6
Overview of the XDO template language	7
User defined element content	7
Parsing process	7
Using regular expressions on the input text	8
Variables	9
Types	9
Conversion rules	9
Variable elements	9
Built-in variables	10
Variable scopes	11
Expressions	11
Operands	11
Operators in order of precendce	11
side effects	12
List of XDO elements	12
<xdo:attribute>	12
<xdo:autonum>	13
<xdo:branch>	13
<xdo:comment>	14
<xdo:default>	15
<xdo:if>	15
<xdo:output>	16



<xdo:parser>	16
<xdo:pattern>	17
<xdo:process>	18
<xdo:set>	19
<xdo:template>	20
<xdo:text>	20
<xdo:value>	20
<xdo:variable>	22
<xdo:when>	22
<xdo:while>	23
User defined elements.....	24
<any-element>	24
User defined text.....	25
Appendix A: XDO Schema.....	26
Appendix B: Examples.....	26
Appendix C: Bibliography.....	26



Abstract

This document describes a parser and a scripting language, named XDO, to convert text documents into XML documents. It also presents the Domteur software to run XDO templates.

Status of this document

This document is a preliminary description of the XDO parser language. At the time of publishing, XDO is still undergoing changes.

Author: ulrich.hoehne@arxio.de

Revisions:

- | | | |
|-----|------------|--|
| 0.3 | 2006-05-20 | XDO description for version 0.6 of Domteur implementation. |
| 0.2 | 2006-03-10 | XDO description for version 0.5 of Domteur implementation. |
| 0.1 | 2006-02-09 | First draft of XDO for experimental implementation. |

Copyrights and Trademarks

Domteur and XDO is copyright of arxio GmbH

Java is a Trademark of Sun Microsystems

XML and XSL are generic terms of the W3C consortium



Introduction

Domteur is a data conversion tool that takes plain text documents as input and generates XML documents as output. The translation process is described in an XDO template document that must be loaded and compiled by Domteur.

The XDO template language is designed to offer an easy approach for parsing the input text with regular expressions and putting the parser result into XML elements as they are parsed. Complex parsers can be written by combining multiple parser elements and by calling them as needed. XDO offers a certain degree of flexibility in the dynamic generation of XML output elements and content, but its main goal is to build parsers to capture input documents and translate them into XML. If a more elaborated control of the generated XML is necessary, the produced XML can be processed further with XSL stylesheets.

In addition to XML output, Domteur can alter the parsed input text and output it, acting like a 'search and replace' engine. The altered input text may also serve as new input for multiple pass parsers.

For convenience, Domteur offers the possibility to load an XSL stylesheet and process the generated XML document after the parsing process has completed. The XSL translation engine is fed directly from the internal DOM tree generated by the Domteur parsing process.

Conventions and Styles

Usually the text in this document is formatted in Sans Serif (like the text you are reading now). To show actual code, such as XML elements or XDO template examples, a Courier font is used. For example: `<xdo:template>`.



Terminology

Column	Denotes a character position in the input text. The leftmost character is 1.
Current element	The XML element the parser is currently generating.
Cursor	Denotes the current position of the parser engine in the input text. The initial cursor position at the start of the input text is 0.
Domteur	The arxio program that processes XDO scripts.
Generated XML	The XML output of the parser.
Input document	The text of the input document that is parsed to produce the generated XML document. Note that the parser can alter the input text.
Intermediate document	Denotes input text that has been altered by the parser.
Line	Denotes a line in the input text. The first line number is 1.
Output document	The text that is output by the parser after parsing is completed. Same as input text if the parser did not alter the text.
Parser	Set of XDO elements constituting a parser entity. An XDO script may contain several parsers.
Pattern	A regular expression pattern.
XDO	XML language for Domteur parser templates
XDO document	Parser template written in XDO, must be loaded and compiled by Domteur to set up an actual parsing engine.



Overview of the XDO template language

An XDO template contains elements to define the parsing process, usually designed for a specific class of input documents, and elements to define the structure of the XML document to be produced as output. XDO defines its own namespace, `xdo:` for its elements. All other elements, denoted as *user defined elements* in the text, are generated in the XML output document.

User defined element content

Element content is generated by nesting XDO elements like `<xdo:value>` inside user defined elements.

Example

```
<title><xdo:value select="'My title'"/></title>
```

generates an XML element

```
<title>My title</title>
```

Parsing process

The XDO document root element `<xdo:template>` can contain one or more `<xdo:parser>` elements. Each parser element must have its distinct name, set in the name attribute. The parser element usually contains several `<xdo:pattern>` elements to define regular expressions, the parser program flow elements `<xdo:branch>`, `<xdo:if>`, `<xdo:while>` and `<xdo:process>`, that acts like a subroutine call for `<xdo:parser>` elements.

When a XDO template is loaded by Domteur, the parser engine registers and compiles each XDO element it encounters. User defined elements and user defined text nodes are also registered. `<xdo:parser>` elements are extracted from the element tree at the compile stage and registered apart. No parsing of the input text is done at this program stage.

After compiling the XDO template, the parser begins to process the compiled elements in their order of appearance. `<xdo:template>` from top to bottom. Remember that `<xdo:parser>` elements have been extracted from the `<xdo:template>` element at compile stage, so they are not encountered in the process stage. Each time the parser

engine encounters a user-defined element or user defined text, it generates it in the output document. When a `<xdo:process>` element is encountered, the parser engine continues processing the `<xdo:parser>` element set in its `parser` attribute. This means that you need at least one `<xdo:process>` element outside of a `<xdo:parser>` element to enter a parser element and begin parsing the input document.

Using regular expressions on the input text

The parsing engine uses regular expressions defined in `<xdo:pattern>` elements to advance the cursor and extract text from the input text. Patterns can be referenced in any expression. Usually they appear in the `test` expression of the XDO program flow control elements `<xdo:if>`, `<xdo:when>` and `<xdo:while>`. When a pattern matches the input text at the cursor position, it evaluates to `true` and the matched text is made accessible via the `in` in the built-in `_group[]` array. Conforming to standard regular expression usage, `_group[0]` contains the entire matched text, `_group[1]` the first capturing group and so on.

Example

Define a pattern to match the `.SH` formatting element in a Unix man-page:

```
<xdo:pattern name="rx-sh">\.SH (.*)?(?:\n|\z)</xdo:pattern>
```

If the regular expression matches the input text, it evaluates to `true` and the `<xdo:if>` element is entered by the parser engine. The cursor is advanced to the end of the matched text. Then, the text in capturing group 1 is put as content in a `<h2>` element and generated in the XML output document.

```
<xdo:if test="$rx-sh">
<h2><xdo:value select="$_group[1]"/></h2>
</xdo:if>
```

Variables

Domteur offers a small set of user definable and built-in variables to help controlling the parsing process and to provide some flexibility in the dynamic generation of XML output elements and content.

Types

XDO templates can use the following built-in types:

boolean	false true
integer	32bit signed integer
string	16bit Unicode character strings
pattern	Regular expression pattern, evaluates as boolean when used in expressions

Domteur variables accept values of all built-in types (with the exception of the special `pattern` type), and offer automatic type conversion in many cases, like in most scripting languages.

Conversion rules

<i>Source</i>	<i>Target</i>	
boolean	integer	false -> 0, true -> 1
boolean	string	false -> "false", true -> "true"
integer	boolean	0 -> false, not 0 -> true
integer	string	134 -> "134"
string	boolean	"1" "true" -> true, any other -> false
string	integer	"134" -> 123, "true" -> 1, any other -> 0

Variable elements

XDO offers the following elements to contain variables:

<code><xdo:autonum></code>	Read-write, integer variable with auto-increment feature.
<code><xdo:pattern></code>	Read-only <code>pattern</code> type value, defined by a regular expression pattern string.
<code><xdo:variable></code>	Read-write, accepts boolean, integer, string.

Built-in variables

In addition to the XDO variables, the parser engine defines a set of built-in variables that can be used in XDO expressions. All built-in variables are read only, with the exception of the `_group[]` array.

<code>_column</code>	The position of the cursor in the current line. Leftmost position has index 1. Tab ' <code>\t</code> ' characters are counted as one column, as the parser cannot guess the tab width. A tabwidth setting may be added in a future release.
<code>_cursor</code>	The string index of the current cursor position. First character in input text has index 0.
<code>_group[]</code>	Array of <code>string</code> with the text of the last regular expression match. The entire match is always stored at index 0, the capturing group 1 at index 1 etc. Groups are writeable, which means that the original input text matched by a group can be replaced by setting the group content.
<code>_input-path</code>	<code>string</code> containing the full path of the input text document. May be empty if path is not available for the parser engine.
<code>_input-name</code>	<code>string</code> containing the file name of the input text document. May be empty if file name is not available for the parser engine.
<code>_line</code>	The line number of the current cursor position. First line has number 1.
<code>_timestamp</code>	Current system timestamp.
<code>_template-path</code>	<code>string</code> containing the full path of the XDO template document. May be empty if path is not available for the parser engine.
<code>_template-name</code>	<code>string</code> containing the file name of the XDO template document. May be empty if file name is not available for the parser engine.

Variable scopes

The scope of a `<xdo:variable>`, `<xdo:autonum>` or `<xdo:pattern>` variable is the element where it is defined, including the child elements of that element. The variable name must be unique in the variable scope. Note that XDO elements have their own rules regarding the allowance of the specific XDO variable elements as their child elements.

The scope of all built-in variables is `<xdo:template>`.

Expressions

Expressions are accepted in the `test` attribute of the XDO program flow elements `<xdo:if>`, `<xdo:when>`, `<xdo:while>` and in the `select` attributes of `<xdo:attribute>`, `<xdo:set>` and `<xdo:value>`. Expressions contain a sequence of operands (identifiers and literals) and operators.

Operands

Type	Example
Variable identifier	<code>\$identifier</code>
boolean literal	<code>true</code> , <code>false</code>
integer number	<code>134</code>
string literal	<code>'This is a text'</code>
Attribute identifier	<code>@attr-name</code> refers an attributes of the nextmost output XML element
XPath descriptor	<code>/body/my/element@attr-name</code> refers an element/attribute in the output XML document

Operators in order of precedence

<code>()</code>	Subexpression grouping
<code>*</code> <code>/</code> <code>%</code>	Arithmetic
<code>+</code> <code>-</code>	Arithmetic
<code>.</code>	String concatenation
<code><</code> <code><=</code> <code>==</code> <code>!=</code> <code>>=</code> <code>></code>	Comparison
<code>&</code>	Logical AND
<code> </code>	Logical OR

Side effects

When evaluating a `<xdo:autonum>` variable, it is incremented after returning its value. This corresponds to the behavior of the postfix operator `++` in C++ or Java.

When evaluating a `<xdo:pattern>` variable, the pattern is used on the input text at the cursor position. If the pattern matches the input text, the cursor is advanced.

List of XDO elements

`<xdo:attribute>`

Sets an attribute of the current XML output element. If the attribute does not exist, it is added to the element's attribute list. Used for dynamic generation of attributes from parsed input text.

Attributes

Name	Value	Comment
<code>name</code>	<code>identifier</code>	The attribute name.
<code>select_{opt}</code>	<code>expression</code>	Optional expression evaluated to the value to be set. If both a <code>select</code> attribute and element content exist, the <code>select</code> attribute is evaluated first.

Examples

```
<element>
  <xdo:attribute name="type" select="$_group[0]">
</element>
```

Adds an attribute `type` to `<element>` and sets its value to matched input text in `$_group[0]`

```
<element type="none">
  <xdo:attribute name="type">type-name-<xdo:value select="$_group[0]" />
  </xdo:attribute>
</element>
```

Sets attribute `type` of `<element>` to "type-name" followed by matched input text in `$_group[0]`

<xdo:autonum>

Creates a named integer variable with auto numbering feature. This means that after each access, the variable increments itself. If `increment` is set to 0, the variable acts like a normal integer variable.

Attributes

Name	Value	Comment
<code>name</code>	<code>identifier</code>	The variable name.
<code>base_{opt}</code>	<code>integer</code>	The initial value of the variable (defaults to 1 if omitted).
<code>increment_{opt}</code>	<code>integer</code>	The increment value (defaults to 1 if omitted).

Examples

```
<xdo:autonum name="auto" />
```

Defines a autonum variable with name "auto", initial value 1 and increment 1.

```
<xdo:autonum name="count" base="0" increment="10" />
```

Defines a autonum variable with name "count", initial value 0 and increment 10.

```
<xdo:autonum name="countdown" base="10" increment="-1" />
```

Defines a autonum variable with name "auto", initial value 10 and increment -1.

<xdo:branch>

An `<xdo:branch>` element groups one or more `<xdo:when>` elements and a single optional `<xdo:default>` element. The default element may appear at any place in the branch element. After entering the `<xdo:branch>` element the parser processes the `<xdo:when>` elements in their order of appearance. If a `<xdo:when>` element is entered, the branch element is left after processing the child elements of the `<xdo:when>` element. If no `<xdo:when>` element is entered, the parser enters the `<xdo:default>` element, if it finds one.

See `<xdo:when>` `<xdo:default>`

Attributes

Name	Value	Comment
name _{opt}	identifier	Optional element name, displayed by Domteur in 'verbose' mode.

Examples

```
<xdo:branch>
  <xdo:when test="$rx-one">
    Hit one
  </xdo:when>
  <xdo:when test="$rx-two">
    Hit two
  </xdo:when>
  <xdo:default>
    No hits
  </xdo:default>
</xdo:branch>
```

A branch with two `<xdo:when>` elements. If pattern `rx-one` matches the input text at the cursor position, text "Hit one" is added to the XML output element, if pattern `rx-two` matches, text "Hit two" is added, if none of the patterns match, text "No hits" is added.

`<xdo:comment>`

Inserts an XML comment `<!--comment -->` in the current element. Used for dynamic generation of comments from parsed input text.

Examples

```
<body>
  <xdo:comment>Generated on:<xdo:value select="$_timestamp"/>
</comment>
</body>
```

Adds a `<!-- Generated on: 2006-03-06 16:17:01.359 -->` comment with a timestamp to the `<body>` element.

<xdo:default>

The `<xdo:default>` element is entered when no `<xdo:when>` element was entered in an `<xdo:branch>` element. The default element may appear anywhere in a branch element, but a branch element can only have one `<xdo:default>` element.

See `<xdo:branch>` `<xdo:when>`

<xdo:if>

Controls the program flow of the Domteur parser engine. If the expression in the `test` attribute evaluates to true, the element is entered and the parser engine continues processing the `<xdo:if>` children. Otherwise the parser continues with the next sibling element of `<xdo:if>`.

See [Expressions](#).

Attributes

Name	Value	Comment
<code>test</code>	expression	Entry condition. If it evaluates to <code>true</code> , the <code>if</code> element is entered.
<code>name_{opt}</code>	identifier	Optional element name, displayed by Domteur in 'verbose' mode.

Examples

```
<xdo:if test="rx_bigtitle"/>
  <h1><xdo:value select="$_group[0] " /></h1>
</xdo:if>
```

If regular expression `rx-bigtitle` matches the input text at the cursor position, create element `<h1>` in the XML output document and add the matched text to it.

<xdo:output>

This element serves to control the different aspects of the output XML document. It defines the public and system ID used in the `<!DOCTYPE>` element of the generated XML document. If the `doctype-` attributes are defined, the qualified name of the document type is generated from the first user defined element the parser encounters. The remaining attributes control the XML serializer format settings. They apply only when the XML output is written to a file or a stream. If the DOM tree produced by the parser engine is used directly (e.g. as input for a XSLT engine) these settings are ignored.

Attributes

Name	Value	Comment
<code>doctype-public_{opt}</code>	String	PUBLIC ID of the document type.
<code>doctype-system_{opt}</code>	String	SYSTEM ID of the document type.
<code>indent</code>	"yes" "no"	Sets indentation mode of the XML serializer
<code>omit-xml-declaration</code>	"yes" "no"	If set to "yes", tells the XML serializer to omit the <code><?xml...?></code> declaration at the top of the output document. Defaults to "no".
<code>method</code>	"html" "xml"	Sets the output mode of the XML serializer. Default to "xml" if omitted.

<xdo:parser>

Creates a parser element to be processed by the Domteur parser engine. The child elements of `<xdo:parser>` are processed in their order of appearance.

`<xdo:parser>` elements can only appear in the `<xdo:template>` document element. They may not contain nested `<xdo:parser>` elements. To enter a parser it must be called with an `<xdo:process>` element.

See `<xdo:process>`

Attributes

Name	Value	Comment
<code>name</code>	identifier	The name of the parser.

Examples

```

<xdo:parser name="convert-dashes">
  <xdo:pattern name="rx-not-eof">(?!\\z)</xdo:pattern>
  <xdo:pattern name="rx-dash">\\-</xdo:pattern>
  <xdo:pattern name="rx-not-dash">.*?(?=\\-|\\z)</xdo:pattern>

  <xdo:while test="$rx-not-eof" name="not-eof">
    <xdo:branch name="dash">
      <xdo:when test="$rx-dash" name="dash">
        <xdo:set lvalue="$_group[0]">-</xdo:set>
      </xdo:when>
      <xdo:when test="$rx-not-dash" name="not-dash"/>
    </xdo:branch>
  </xdo:while>
</xdo:parser>

```

Defines a parser element to replace escape sequences "\\-" for dashes in man-pages documents to simple "-" characters. The parser is entered when the Domteur parser engine encounters a `<xdo:process parser="convert-dashes"/>` element.

<xdo:pattern>

Creates a named pattern string defining a regular expression. References to pattern elements can appear in expressions in `test` and `select` attributes. The regular expression engine used by the Domteur parser engine depends on the actual implementation of the Domteur program. The first version of Domteur is written in Java, and the Java Platform 5.0 `Pattern` class is used.

See [Expressions](#)

Attributes

Name	Value	Comment
<code>name</code>	<code>identifier</code>	The name of the pattern variable.
<code>select_{opt}</code>	<code>expression</code>	Optional expression evaluated to the pattern string. If both a <code>select</code> attribute and element content exist, the <code>select</code> attribute is evaluated first.

Examples

```
<xdo:pattern name="aword"> [Aa]\w+</xdo:pattern>
```

Create a regular expression pattern named `aword` with pattern string `[Aa]\w+`. The pattern matches words beginning with 'A' or 'a'.

```
<xdo:variable name="keywords" select="'boolean|integer|real'"/>
```

```
<xdo:pattern name="rx-key" select="$keywords"/>
```

Create a regular expression pattern named `rx-key` with pattern string taken from variable `$keywords`.

<xdo:process>

Directs the parser engine to continue processing with the specified `<xdo:parser>` element. After the `<xdo:parser>` element has been processed, the parser engine continues with the next sibling element of `<xdo:process>`.

Attributes

Name	Value	Comment
<code>cursor_{opt}</code>	<code>continue</code>	<code>continue</code> continues parsing at the current cursor position. This is the default
	<code>start</code>	<code>start</code> restart parsing at start of text
<code>parser</code>	<code>identifier</code>	The name of the target parser element.

<xdo:set>

Sets the content of the variable, element or attribute stated in the `lvalue` attribute. If the `lvalue` refers to a pattern group, the matched text in the input text document is replaced. The value used by `<xdo:set>` is a combination of the value obtained by evaluating the `select` expression and the element content.

Attributes

Name	Value	Comment
<code>lvalue</code>	<code>identifier xpath</code>	Refers the variable or XML output element node to be set.
<code>select_{opt}</code>	<code>expression</code>	Optional expression evaluated to the value to be set. If both a <code>select</code> attribute and element content exist, the <code>select</code> attribute is evaluated first.

Examples

```
<xdo:set lvalue="$_group[0]" select="'Replacement text'"/>
```

Replace entire text of last regular expression match with string "Replacement text".

```
<xdo:set lvalue="$varname"/>Found:
<xdo:value select="_group[1]"/></xdo:set>
```

Set value of variable "varname" with string beginning with "Found: " and continuing with the matched input text in capturing group 1.

```
<xdo:set lvalue="/book/title@text" select="$_group[0]"/>
```

Uses an XPath to set the attribute `text` of XML output element `<title>`, which is a child of document root element `<book>`.



<xdo:template>

Root element of an XDO script. Contains the entire XDO template.

Attributes

Name	Value	Comment
Version	string	Currently fixed to "1.0"

Examples

```
<xdo:template version="1.0">
  <!-- the XDO template script goes here ... -->
</xdo:template>
```

<xdo:text>

Inserts its element contents as text node in the current XML output element element or in its XDO parent element if the parent is `<xdo:attribute>`, `<xdo:set>` or `<xdo:value>`. Usually you don't need `<xdo:text>`. User defined text can be entered directly between elements. However, it can be used to insert text with white space characters mixed with `<xdo:value>` elements.

Examples

```
<section>
  <xdo:text>-- Section begin marker --
</xdo:text>
  <value select="$_group[0] " /> <!-- Section content -->
  <xdo:text>
    -- Section end marker --</xdo:text>
</section>
```

<xdo:value>

Used to insert a value in the current XML output element or in its XDO parent element if the parent is a <xdo:attribute> or <xdo:set> element. The value is evaluated from the expression in the `select` attribute.

See [Expressions](#)

Attributes

Name	Value	Comment
<code>select_{opt}</code>	<code>expression</code>	Optional attribute referring a value to be set. If both a <code>select</code> attribute and element content exist, the <code>select</code> attribute is evaluated first.

Examples

```
<xdo:value select="$_group[0]"/>
```

Get entire text of last regular expression match.

```
<xdo:value select="$varname"/>
```

Get value of variable.

```
<xdo:value select="'String literal'"/>
```

Get value of given string literal.

```
<xdo:value select="book/@title"/>
```

Uses Xpath to resolve attribute `title` of element `book` in current XML output element.

```
<xdo:value select="$_line % 2 == 0"/>
```

Evaluates expression (in this example: true, if line is even, otherwise odd)

<xdo:variable>

Creates a named variable. Variables can hold values of type `string` or `integer`. The initial value is evaluated from the expression in the `select` attribute. The element's text content is ignored.

If `select` is omitted, the variable string value is the empty string and the integer value is 0. If the `select` is present, the variable is initialized every time the parser processes the `<xdo:variable>` element. To avoid unnecessary initialization of variables, place them outside of `<xdo:parser>` elements or at the top of a `<xdo:parser>`.

Variables can be referenced with the `select` attribute in `<xdo:attribute>`, `<xdo:set>` and `<xdo:value>` and the `lvalue` attribute in `<xdo:set>`.

See [Variables](#)

Attributes

Name	Value	Comment
<code>name</code>	<code>identifier</code>	The variable name.
<code>select_{opt}</code>	<code>expression</code>	Optional attribute referring a value to be set. If both a <code>select</code> attribute and element content exist, the <code>select</code> attribute is evaluated first.

Examples

```
<xdo:variable name="section-begin" select="'-- Section marker --'"/>
<xdo:variable name="max-expected-lines" select="5"/>
<xdo:variable name="headline" select="$_group[0]"/>
```

Creates 3 variable elements initialized (1) with a literal string, (2) a literal integer value and (3) the result of the last match of a regular expression.

<xdo:when>

<xdo:when> elements can only appear in <xdo:branch> elements. If the expression in the `test` attribute evaluates to true, the element is entered and the parser engine continues processing the <xdo:when> children. Otherwise, the parser continues with the next <xdo:when> or <xdo:default> sibling element, if any of these exist, or leaves the parent <xdo:branch> element.

See [Expressions <xdo:branch> <xdo:default>](#).

Attributes

Name	Value	Comment
<code>test</code>	<code>expression</code>	Entry condition. If it evaluates to true, the case element is entered.
<code>name_{opt}</code>	<code>identifier</code>	Optional element name, displayed by Domteur in 'verbose' mode.

Examples

```
<xdo:branch>
  <xdo:when test="$rx-one">
    Hit one
  </xdo:when>
  <xdo:when test="$rx-two">
    Hit two
  </xdo:when>
  <xdo:default>
    No hits
  </xdo:default>
</xdo:branch>
```

A branch with two <xdo:when> elements. If pattern `rx-one` matches the input text at the cursor position, text "Hit one" is added to the XML output element, if pattern `rx-two` matches, text "Hit two" is added, if none of the patterns match, text "No hits" is added.

<xdo:while>

Controls the program flow of the Domteur parser engine. If the expression in the `test` attribute evaluates to `true`, the element is entered and the parser engine continues processing the `<xdo:while>` children. Otherwise the parser continues with the next sibling element of `<xdo:while>`.

After processing the child elements, the parser engine re-evaluates the test expression and re-enters the `<xdo:while>` element as long as the test expression remains `true`. Using `<xdo:while>` in an XDO script can lead to endless loops if the test expression contains an expression that does not advance the cursor, (e.g. a regular expression pattern with a non-capturing group, that continues matching the same word), and none of the `<xdo:while>` child elements advances the cursor. In this case, the parser engine stops and issues an error message.

Attributes

Name	Value	Comment
<code>test</code>	<code>expression</code>	Entry condition. If it evaluates to <code>true</code> , the <code>while</code> element is entered or re-entered.
<code>name_{opt}</code>	<code>identifier</code>	Optional element name, displayed by Domteur in 'verbose' mode.

Examples

```
<xdo:while test="rx-paragraph">
  <p><xdo:value select="$_group[0]" /></p>
</xdo:while>
```

While pattern `rx-paragraph` matches the input text at cursor position, generate `<p>` elements with the matched text as content text in the XML output document.

User defined elements

<any-element>

Domteur generates a new XML output element with the given tag name and optional attributes whenever it encounters an `<element>` that is not part of the XDO namespace.

Element attributes can also be generated or altered dynamically with `<xdo:attribute>`.

Attributes

Any attributes to generate with the element.

Examples

```
<book title="Fool on the Hill" author="Mat Ruff">
  <xdo:attribute name="price" select="'12,50€'"/>
  <abstract>
    <xdo:value select="_group[0]"/>
  </abstract>
</book>
```

Generates an element `<book>` with a `title` and `price` attribute and sets the content to matched input text in capturing group 0.

User defined text

Domteur generates user defined text if it encounters text content in one of the following elements: `<xdo:attribute>`, `<xdo:set>`, `<xdo:value>`, and in the user defined elements `<any-name>`. User defined text can be mixed with XDO elements to produce a mix of static and parsed output. An alternative is the usage of the `<xdo:text>` element.

Examples

```
<book-critique title="Fool on the Hill" author="Matt Ruff">  
  This book is fun. Well, that's what Eva told me.  
</book-critique>
```

```
<time-schedule>  
  9.00 : <xdo:value select="$rx-gettask"/>  
  11.00 : <xdo:value select="$rx-gettask"/>  
  12.00 : Lunch  
  13.00 : <xdo:value select="$rx-gettask"/>  
</time-schedule>
```



Appendix A: XDO Schema

Appendix B: Examples

Appendix C: Bibliography

- | | |
|--|---|
| [1] XSL Transformations (XSLT) Version 1.0 | http://www.w3.org/TR/xslt |
| [2] XML Path Language (XPath) Version 1.0 | http://www.w3.org/TR/xpath |
| [3] Java2 Platform | http://java.sun.com/ |