Høgskolen i Østfold

WXT
from Ostfold College

WXT is: a Web and Xml Tool

WXT handles tasks that involves XML-files in general, and it has some special features that are directed towards XHTML.

Wxt is based on the following simple observation:

"if the Internet is wellformed, you can reach and collect everything with an URL and an XPATH-expression".

The fact that the Internet is not wellformed, may be a concern to those of us with a sceptical inclination, so it may also be usefull to access content that is not wellformed as well.

WXT can assist when you:

- build and maintain websites based on XHTML.
- merge and/or split XML-files and transform them on the way to produce documents of allmost any kind.
- combine the two tasks and build XHTML-pages and sites from any wellformed XML-rawmaterial.
- want to include text that is not wellformed into a wellformed surrounding.
- combine the tasks and build XHTML-pages and sites

WXT is based on a few main principles:

- It does not introduce any hidden or proprietary formats or files. You may at any time supplement WXT with other tools or abandon WXT completely.
- It invites you to separate content, style and structure.
- It is based on a script that competely defines a job, small or big. There is a GUI to monitor scripts and jobs, but WXT works as well from the command line.
- You may use your editor of choice to prepare material.

WXT does not mess things up for you since:

- It never deletes any files
- It never change the content of other files than those you ask it to build.

WXT is

- completely free and distributed "as is"
- written in Java and testet extensively on MS Windows, and to some extent on Linux and MacOS

Halden , Jan 11, 2009
Børre Stenseth
.

Content

Who is WXT for

[Scenario 1][Scenario 2][Scenario 3]

WXT may be used at different levels and may be engaged in different tasks, alone or together with other tools. It is potentially both an administration tool and a production tool. To make a coarse statement of usability, it is fair to say that:

WXT **is not** effective for:

- large heavy duty websites that depends on highly dynamic content management.
- novice publishers that starts FrontPage, type something and hopes for the best.

It **is** mainly targeted at:

- small companies, working groups or indivuals that maintain non-trivial websites with multiple sources with a reasonable need for dynamics.
- authors that want to publish on different media in different formats.

The best way to understand the possibilies and limitations is to read the documentation, the scenarios below or investigate the examples you may download from Download.

You should have som basic knowledge of XML before you start using WXT. At least you should be able to write well-formed XML. The functionality relies on some simple XPATH-expressions. The page XPATH may serve as first-aid. You will find some usefull references on the page References.

.

Scenario 1

The authors story

I am a teacher of computer science and have an ambition of maintaining a few rich websites for the main topics I am teaching. I want, in addition to the website, to produce handouts, printable material, overheads and occasional reports of different sorts. All based on the same source of material. I also want to share parts of my material with collegues and I want to borrow material. An integral part of my educational strategy is to include students works in the source material. As a teacher in Computer Science it is vital to be able to present snippets of programcode in different formats. This code must be up to date, all changes in sourcecode must be reflected in the material.

It is obvious from the description above that the need for dynamics is moderate. I can rebuild everything or the affected parts whenever I do changes in some parts of the material.

I have two basic choices when preparing material with the intentions above.

- I can produce material in a rather strict XML-dialect of my choice and produce web-pages through some transformation. This way I disciplin myselft to a straight format and I have kept backdoors open for alternative transformations of reasonable complexity. DocBook is an obvious candidate for production according to this strategy since tools are availabe for a series of interesting transformations.
- I may consider the pages on the website as the main documents. In this case I will probably produce XHTML in the contentfiles. Not necessary in the same sorroundings and with the same stylesheet as I use in the templates, but it may be browsable. The advantage of this approach is that I have a lot of tools available for production and proofreading, before I submit the source material to WXT.

WXT lets me follow any of the strategies above. It makes no difference if there is one or many authors or if your sourcematerial is in different locations.

The sites below are maintained with WXT, from a lot of different sourcematerial, programcode files, XHTML-pages, text files:

Computer Graphics, in Norwegian.
Markup and Web, in Norwegian.
.Net programming in C#, in Norwegian.

.

Scenario 2

Listbased publishing

Assume that the basic material can be described in a rather simple XML-structure, for instance a fairly homogenous list of items. The items may be orders, images, spare parts, brochure pages or any type of items you would like to sell, produce

or maintain. WXT may help you organise and publish these items in any order or organization you want, on any format you prefer.

.

Scenario 3

Cooperative work

Assume that a document or website should be maintained by any number of contributers on the internet. This document may be a website or a document ment for printing. You can set up a script for WXT that collect material from predefined URI's an produce the latest version of the document. The format of the contributions may be defined specialy for the task at hand or it may be according to an open standard.

.

Documentation

The pages in this section is more or less the same documentation that is distributed with the program. As a matter of fact WXT collects the documentation from the Java-project and produce these pages.

The first pages gives you a general idea of how WXT ticks.

The page **Ajax** introduces a Javascript library that come handy if you want to produce popup-pages or do HTTPRequests to make dynamic pages.

The page **Styling** explains how you must prepare CSS-materiel to get the best of WXT in webpages. A sample stylesheet is available.

The page **XPATH** introduces some simple xpath-expressions that may come handy as a strating ramp if you are not familiar with the concept.

The sections **The Script** and **The PIs** (Processing Instructions) is a detailed documentation of what you can do, and how.

.

WXT-intro

WXT is script-driven. That means that the tasks WXT performs is described in a script. You write the script and WXT performs the actions described in that script. The bulk of these help pages are meant to help you:

- prepare a script, see Script
- mark your pages so WXT can extract and build the documentfragments you want, see Processing Instructions

WXT may be used from the commandline, for instance like this:

```
java -jar c:\wxt\wxt.jar c:\myfiles\script.xml
```

See commandline for details.

WXT does also have a GUI that helps you edit and validate the script and perform parts of the script, see GUI

The GUI has very simple editing support so you may find it usefull to use your favourite text editor to develop a script and use the GUI to experiment with partial builds etc.

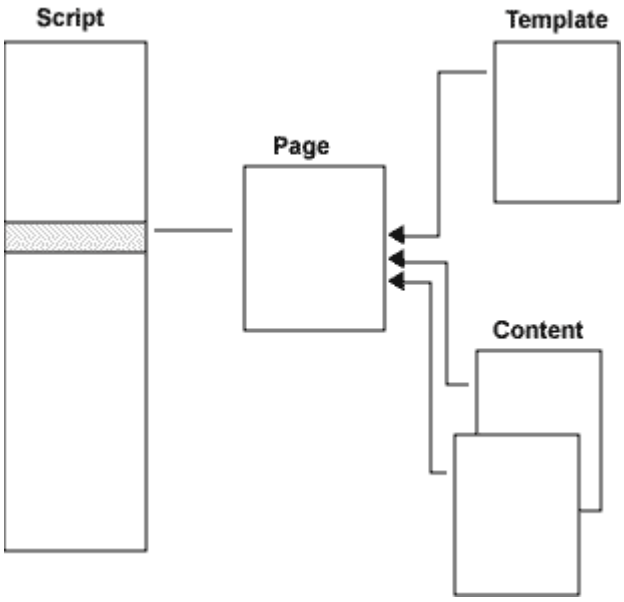
An introductory description of WXT and some examples are found at the website: **http://www.ia.hiof.no/~borres/wxtdoc/**

.

WXT-basics

The basic functionality of WXT is to **build pages**. This does not necessarily mean to build web-pages. It means in general that we may produce a page by extracting from many pages, transforming a page, splitting a page or any combination.

Below we focus on the obvious case where we will build a page from a template and one or more content-files by using a template which will be populated from a set of contentfiles. This is how it works



The script defines a page by telling which template to use and which contentsfiles to fetch material from. A simplified extract from a script may look like:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<script version="1.0">
  <group ...>
    <template name="P" location="templates/p_template.xml"/>
    <page name="page1" template="P" location="index.html">
      <content location="content/b-index.xml"/>
    </page>

    <page name="page2" template="P" location="pub/p1.html">
      <content location="content/b-page1intro.xml"/>
      <content location="content/b-page1.xml"/>
    </page>
  </group>
</script>
```

It is not satisfactory to copy contentfiles completely into the template. We need a mechanisme that gives us the possibility to extract exactly what we want. The answer to this is XPATH and Processing Instructions. XPATH is a general tool for selecting any nodeset from a XML-file and Processing Instruction (PI) is a legal XML-element that may be picked upp by any useragent, of which WXT is an example. Assume for instance that our template has the following line:

```
<_wxt import xpath="//div[@class='main']/*"?>
```

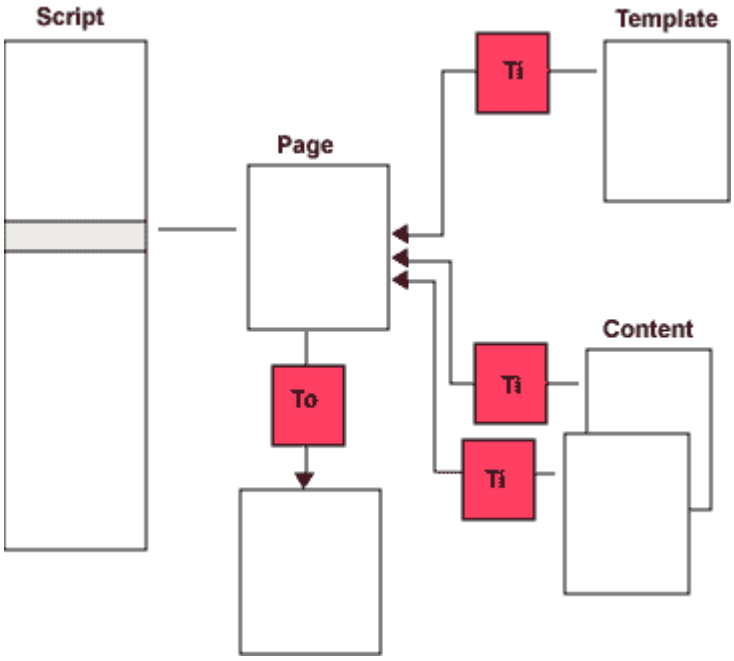
This tells that the PI identifies itself as usefull for WXT (_wxt), it tells that WXT should interpret this as an order to **import** and it says that we should import the children of all div-elements that has an attribute class with the value main. These elements may be scattered around the contentfile, and may even exist in many contentfiles if the script says so.

A templatefile will normally have many such import PI's. This constitutes a very flexible and general mechanism to extract material from the contentfiles at will. If we control the contentfiles, we may realize allmost any construction job we may think of. This combination of the structure as described in the script and the PI's constitute the basic driving mechanisms in WXT.

Please note that even if we have used an example inspired by (X)HTML above, this basic engine in WXT works as well on any wellformed XML-dialect.

Transformations

We may introduce XSLT-transformations as shown:



Ti is XSLT-transformations that is applied before the template or contentfile is used by WXT. The output of these transformations must allways be wellformed XML. All files, templates and contentfiles, may have different transformations. Two files may have the same source and different transformations. We may even control the transformations by parameters which may be different for different files.

To is an output transformation. This is performed as the last operation before a page is written to file. The output of a To-transformation may be anything you may produce by XSLT: text, xml, html, XHTML, rtf or whatever. A simplified extract from a script may be like this:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<script version="1.0">

  <group ...>
    <template name="P" location="templates/p_template.xml"/>
    <transformation name="T1" location="trans/finaltrans.xslt"/>
    <transformation name="T2" location="trans/contenttrans.xslt"/>

    <page name="page1" template="P" location="index.html" transformation="T1">
      <content location="content/b-index.xml"/>
    </page>

    <page name="page2" template="P" location="pub/p1.html">
      <content location="content/b-page1intro.xml"/>
      <content location="content/b-page1.xml" transformation="T2"/>
    </page>

  </group>
</script>
```

Again I would like to stress the point that we still have not introduced any limitation on the type of files we are working with as rawmaterial. Any wellformed XML-construct will do.

.

Command Line

You can run WXT from the commandline. The format of a command is something like this:

```
java -jar wxt.jar script [page]
```

depending on where you have placed wxt.jar.

Page is optional. When omitted, wxt build all pages in the given script. For instance like this on my harddisk:

```
java -jar c:\wxt\wxt.jar c:\project\script.xml
```

You may use the page-argument to restrict building to one page or one group. A group must me identified by name. A page may be identified by name or by a (partial) filepath. Examples:

```
java -jar c:\wxt\wxt.jar c:\project\script.xml page3
java -jar c:\wxt\wxt.jar c:\project\script.xml c:\project\pages\page-3.html
java -jar c:\wxt\wxt.jar c:\project\script.xml pages\page-3.html
java -jar c:\wxt\wxt.jar c:\project\script.xml pagegroup
```

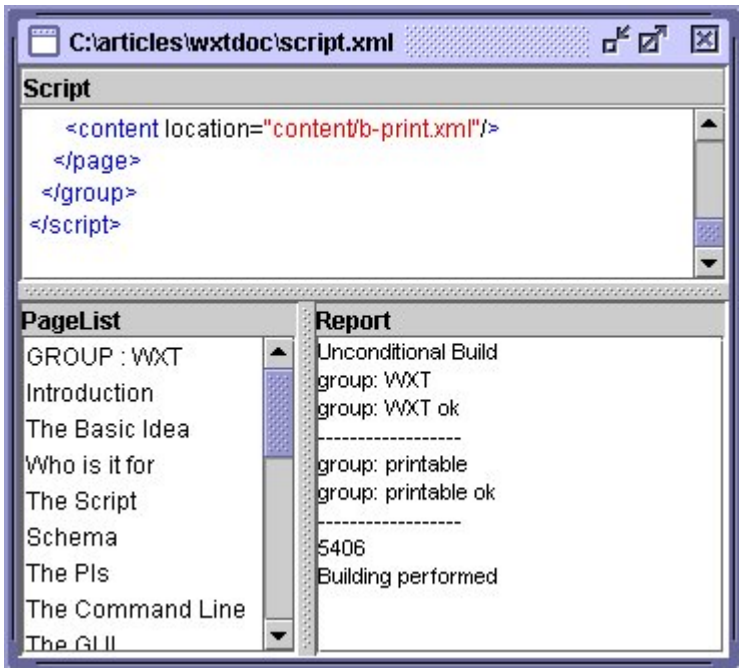
If a filename, or groupname, is ambigiuos, the first match found in a preorder traversal of the page structure is used.

Note that it is not a good idea to use the page-argument with a single page if that page is set up with contents that is based on the content of other pages, for instance an index table (PI: ixtable). Tables of content is ok in this respect since they depend on the structure given i the script, not on the content of pages.

WXT-GUI

WXT comes with a GUI that helps you monitor your operations. The menu and toolbox line is like this:

The menues and the buttons are explained below.



Each document window consists of three components:

- The Script
- A page list
- A report window

The files that you load, save and edit is allways scripts. Pagelist is a simple list of alle pages in the script. It is built once the script is parsed. Report have dynamic content that depends on your actions.

Note that you may right-click on a line in the pagelist to invoke an editor or a browser, if you have specified those in the script, see: program.

Script

The Script menu is as you may expect from a normal file menu and should be self-explanatary.

Edit

The Edit menu is as you may expect and should be self-explanatory.. Remember that we allways edit the script.

Build

The Build menu has the following functions:

Parse Parse the script. All structures are established but no files are loaded. A parse is required every time the script has been changed in this GUI or in another editor.

Build All Pages Do an unconditional build of all pages in all groups. Everything is built from scratch.

Build selected Pages Do an unconditional build of all pages which are selected in the lower right list of files.

Build Updated Pages Build all pages on the condition that one of its contentfiles or its template or its transformation or one of its contentfiles transformations has been changed since last build. (Imported material, except from the contentfiles are not checked for updates.)

The latter two operations reduces processing time when you are working with a limited number of pages in a big script.

View

The functions should be self-explanatory.

Control

The control menu has two items:

- Check links in pages
- Validate script

Check links in pages

This is a posiibility to run a check on links found in the built pages in the current script. You may run the check on relative links or on all links.

Validate

The Validate menu has the following subitems:

Validate script with online schema will attempt to access the validator schema at WXT's homepage.

Validate script with local schema will attempt to access a local validator schema as spesified with the menuitem below.

Set local validator schema ... will let you enter/browse the path to a schema file that you want tu use for validation of the script.

Note that validation according to the standard schema for WXT is strict in the sense that it checks that all elements are in a predefined order. Simply parsing a script is much more tolerant. You will probably want to validate a script fairly seldon, while you will have to parse it often.

.

Ajax

WXT may utilize some javascriptcoding and ajax functionality for webpages. If you use the Processing Instructions **popup** ,**request** or some features of **import-text**, you may want to include the following javascript in your code: http://www.ia.hiof.no/~borres/common/jscripts/std.js. It may be a good idea to include it in your template-pages.

You may of course write your own. It should go something like this:

```
/*
Minialistic library used on my websites
- a simple popup,
     simplepopup(url,wname,wstyle)
- general AJAX with text request/post,
     doRequest(theURL,params,targetId,waiter)
     doGetRequest(theURL,targetId,waiter)
- expand/unexpand text request/get
     expand(address,targetNode)
     unexpand(address,targetNode)
All functions may be used directly, but are also supported by WXT
B.Stenseth 2007.
*/

// ------------  pop ------------------
//popup
  function simplepopup(theURL,wname,wstyle)
  {
          if(wstyle=='*')
                  wstyle='scrollbars=yes,resizable=yes,width=600,height=600,status=no';
          try{
                  newwindow=window.open(theURL, wname, wstyle);
                  if (window.focus) {newwindow.focus()}
          }
          catch(E){
                  // ??
                  alert('If you have blocked pop-ups\n Press Ctrl-Alt when clicking');
          }
  }
//eofpopup

//------------- std ajax ------------------
function establishRequest()
{
    var theRequest=null;
    if (window.XMLHttpRequest)
        theRequest = new XMLHttpRequest();
    else if (window.ActiveXObject)
    {
        try { theRequest = new ActiveXObject("Msxml2.XMLHTTP");
        }
        catch(e) {
            try {theRequest= new ActiveXObject("Microsoft.XMLHTTP");
            }
            catch(e) {theRequest=null;}
        }
    }
    else
        theRequest=null;
    return theRequest;
}

function postRequest(theRequest,params,targetId,theUrl)
{
    theRequest.onreadystatechange =
        function( ) {
                    processRequestChange(theRequest,targetId);
        };
    theRequest.open("POST", theUrl,true);
    theRequest.setRequestHeader("Content-type",
                                "application/x-www-form-urlencoded");
    theRequest.setRequestHeader("Content-length", params.length);
    theRequest.setRequestHeader("Connection", "close");
    theRequest.send(params);
}
```

```
function processRequestChange( aRequest,aTarget )
{
    if (aRequest.readyState == 4) {
        if ((aRequest.status == 200) || (aRequest.status == 304))
            document.getElementById(aTarget).innerHTML=
                aRequest.responseText;
        else
        {
            alert("Problems accessing data:\n" + aRequest.statusText);
            document.getElementById(aTarget).innerHTML=
                aRequest.statusText;
        }

    }
}

//------------------------ general ajax ------------------------
function doRequest(theURL,params,targetId,waiter)
{
    var myRequest=establishRequest();
    if (myRequest)
    {
        if(waiter)
        {
          if(waiter == 'std')
            waiter='http://www.ia.hiof.no/~borres/common/gfx/waiter.gif';
          try{
                  document.getElementById(targetId).innerHTML=
                          '<img src="'+waiter+'" alt=" "/>';}
          catch(E){}
        }
        postRequest(myRequest,params,targetId,theURL);
    }
    else
          {
        alert('Browser will not do a request');
    }
}

function doGetRequest(theURL,targetId,waiter)
{
    var myRequest=establishRequest();
    if (myRequest)
    {
        if(waiter)
        {
          if(waiter == 'std')
            waiter='http://www.ia.hiof.no/~borres/common/gfx/waiter.gif';
          try{
                  document.getElementById(targetId).innerHTML=
                  '<img src="'+waiter+'" alt=" "/>';}
          catch(E){}
        }
        myRequest.onreadystatechange =
          function( ) {
                      processRequestChange(myRequest,targetId);
          };
        myRequest.open("GET", theURL, true);
        myRequest.send(null);
    }
    else
          {
        alert('Browser will not do a request');
    }
}


// ------------------------ expansion ----------------------
//expand
// dependent of style-class: onoff for visible effect
function expand(address,targetNode){
        var myRequest=establishRequest();
        if(myRequest)
        {
                myRequest.onreadystatechange = function( ) {
                    useExpansion(myRequest,address,targetNode);
        };
```

```
                myRequest.open("GET", address, true);
                myRequest.send(null);
        }
}

function useExpansion(theRequest,address,targetNode)
{
    // if the request is complete and successfull
    if (theRequest.readyState == 4) {
        if ((theRequest.status == 200) || (theRequest.status == 304))
        {
            var T=theRequest.responseText;
            var pos1=T.indexOf('<pre');
            var pos2=T.lastIndexOf('</pre');
            if ((pos1 !=-1) && (pos2 > pos1))
                    T=T.substring(pos1,pos2);
            T='<span class="onoff"  '+
               'onclick="unexpand(\''+address+'\',this.parentNode)">-</span>'+T;
            targetNode.innerHTML=T;
        }
        else
          {
            alert("Problem med tilgang til data:\n" + theRequest.statusText);
            targetNode.innerHTML=theRequest.statusText;
          }
    }
}

function unexpand(address,targetNode){
    T='<span class="onoff" onclick="expand(\''
      +address+'\',this.parentNode);">+</span>';
    targetNode.innerHTML=T;
}
//eofexpand
```

Styling

WXT produces some elements which are marked with **class**-attribute. You can, and should, control the appearance of these elements by defining the appropriate styles in a stylesheet.

Element/PI	Styles	Explanation
tocmenu	tocmenu-n , tocmenu-an	The **n** is the level of the page. Level 1 is top. Level 0 is used when the toc is generated with 0 columns (as a line). The **a** is used for the current page.
toctrail	toctrail-0 , toctrail-a0	The **a** is used for the current page.
tocchildren	tocchildren-0	
tocsiblings	tocsiblings-0 , tocsiblings-a0	The **a** is used for the current page.
tocgroup	tocgroup-n ,tocgroup-an	The **n** is the level of the page. Level 1 is top. Level 0 is used when the toc is generated with 0 columns (as a line). The **a** is used for the current page.
tocpage	tocpage-n	The **n** is the level. Level 1 is top. Level 0 is used when the toc is generated with 0 columns (as a line).
ixtable	ixentry , ixword	ixentry is the entry in the indextable. ixword is the word in the indextable.
import-text	nncode, nnword, nnliteral, nncomment	Used when type is applied in import text to "pretty-print" programcode. nn may be: java, c, cpp, csharp, python, xslt, aspx. You should also prepare the anonymous style: .code.
import-text, expand	onoff	Used to format expand and collapse buttons when we have a HTTPRequest on a page.
ref	external	Used to format an a-elemnt that points to an absolute (external) uri.
ref	nolink_span	Used to format an simple "[ix]" construction on a webpage.
reflist	reflist_ul, reflist-li	Used to format an unordered list of references in a referencelist.
reflist	ref_field_n	Used to format fields in a references, n any number [0 ...]
reflist	ref_link, external	ref-link: Used to format all links in a reference external: Used to format absolute links in a reference
demoref	demo-div	Wrapping a demolink
demoref	demo-link	A demolink
demoref	demo-onscreen	A span-element that contains the text seen on screen in a demolink. Should be hidden in media print
demoref	demo-onprint	A span-element that contains the text seen on print in a demolink Should be hidden in media screen
svn	svndiv	Styling the div-fragment that contains the leading text and the svn address
svn	svnaddress	Styling the span-fragment that contains the svn uri.

Part of a sample stylesheet

```
/* ********************************************** */
/* styles hardwired by WXT */
/* ********************************************** */
/* tocmenu PI,  3 levels */
.tocmenu-0,.tocmenu-1,.tocmenu-2,.tocmenu-3,.tocmenu-4,.tocmenu-5,
.tocmenu-a0,.tocmenu-a1,.tocmenu-a2,.tocmenu-a3,.tocmenu-a4,.tocmenu-a5
{
  font-size: 11px;
  font-weight:200;
  background-color:transparent;
  text-decoration:none;
  line-height:120%;
}
.tocmenu-0,.tocmenu-a0{ margin-left:0px;}
.tocmenu-1,.tocmenu-a1{ margin-left:7px;font-weight:200;line-height:150%;}
.tocmenu-2,.tocmenu-a2{ margin-left:14px;}
.tocmenu-3,.tocmenu-a3{ margin-left:21px;}
.tocmenu-4,.tocmenu-a4{ margin-left:28px;}
.tocmenu-5,.tocmenu-a5{ margin-left:34px;}

/* tocgroup PI  3 levels*/
.tocgroup-0,.tocgroup-1,.tocgroup-2,.tocgroup-3,.tocgroup-4,.tocgroup-5,
.tocgroup-a0,.tocgroup-a1,.tocgroup-a2,.tocgroup-a3,.tocgroup-a4,.tocgroup-a4{
  font-size: 12px;
  font-weight:200;
  background-color:transparent;
  text-decoration:none;
  line-height:120%;
}

.tocgroup-0,.tocgroup-a0{ margin-left:0px;}
.tocgroup-1,.tocgroup-a1{ margin-left:30px; font-weight:600; line-height:200%;}
.tocgroup-2,.tocgroup-a2{ margin-left:54px; font-weight:600; line-height:160%;}
.tocgroup-3,.tocgroup-a3{ margin-left:70px;}
.tocgroup-4,.tocgroup-a4{ margin-left:80px;}
.tocgroup-5,.tocgroup-a5{ margin-left:90px;}

/* tocpage PI  5 levels*/
.tocpage-0,.tocpage-1,.tocpage-2,.tocpage-3,
.tocpage-a0,.tocpage-a1,.tocpage-a2,.tocpage-a3
{
  font-size: 12px;
  font-weight:200;
  background-color:transparent;
  text-decoration:none;
  line-height:120%;
}

.tocpage-0,.tocpage-a0{ padding-left:0px;}
.tocpage-1,.tocpage-a1{ padding-left:30px; font-weight:600; line-height:200%;}
.tocpage-2,.tocpage-a2{ padding-left:44px; }
.tocpage-3,.tocpage-a3{ padding-left:50px;}

/* toctrail */
.toctrail-0,.toctrail-a0{
  font-size: 12px;
  font-weight:200;
  background-color:transparent;
  text-decoration:none;
  line-height:160%;
}

/* tocchildren and tocsiblings */
.tocsiblings-0, .tocsiblings-a0, .tocchildren{
  font-size: 12px;
  font-weight:200;
  background-color:transparent;
  text-decoration:none;
  line-height:160%;
}

/* indexes */
.ixentry{margin-left:30px;font-style:italic}
.ixword{margin-left:20px;font-weight:bold}

/* program code */
.cppcode,.ccode,.csharpcode,.javacode,.javascriptcode,.pythoncode,.mlcode,.code,.xsltcode {
        font-size:11px;
        white-space: pre;
        font-family: "Courier new","Courier",  mono-space;
        background-color: #EEEEFF;
        display: block;
        border-style:dashed;
        border-width:1px;
        padding:5px;
}
```

```
.mlcode{background-color: #FFFCFC}

.pythoncode{background-color:#FFFFE7}
.pythonword{color:#FF7700}
.pythonliteral{color:green}
.pythoncomment{color:#DD0000}

.javacode{background-color:#F0F0FA}
.javaword{color:blue}
.javaliteral{color:green}
.javacomment{font-style:italic;color:#DD0000}


.javascriptcode{background-color:#F0F0F0}
.javascriptword{color:blue}
.javascriptliteral{color:green}
.javascriptcomment{font-style:italic;color:#DD0000}


.javacode{background-color:#F0F0FA}
.xsltword{color:blue}
.xsltliteral{color:green}
.xsltcomment{font-style:italic;color:gray}

.cppword,.cword,.csharpword,.javaword{color:blue;font-weight:bolder}
.cppliteral,.cliteral,.csharpliteral{color:green}
.cppcomment,.ccomment,.csharpcomment{color:#DD0000}

/* expansion */
.onoff{
        font-weight:bold;
        background-color:yellow;
        font-size:14px;
        padding-left:3px;
        padding-right:3px;
        cursor:pointer;
        border:solid;
        border-width:1px}


/* end of styles hardwired by WXT */
```

XPATH in short

XPATH was first initiated as a integral, and necessary, part of XSLT, but has been lifted out as a concept of its own that is usable in many connections. XPATH is a general mechanisme to identify items in a treestructured document.

The basic structure in an XPATH-expression is best considered as an analogy to a unix filepath. We allways have to take into account what is called "the context node". In other words we must be aware of where we stand in the tree when we try to formulate an expression that brings us to another place. When you formulate XPATH-expressions in connection with WXT we allways use the root as context node. The task is allways to select a set of nodes somewhere down in the tree (down since trees in the computerworld for some reason allways has the root up). Below you will find some explained examples of expressions that probably is typical for the kind of selections we will use in connection with WXT. The examples are inspired by (X)HTML to support your asossiations, but are of course not limited to this XML-dialect.

//div
All elements of type div in the document
//div/p
All elements of type p which are children of div elements
//div/*
All elements which are children of div elements
//*[@class='mlcode']
All elements which has an attribute with name class with value mlcode
//div/*[@class='mlcode']
All elements which are children of div-elements and has an attribute with name class with value mlcode
//script[@type='text/javascript']
All javascript nodes
//div[@class='clevel1'][position()=2]
The second div-element with an attribute class with value clevel1
//h1 | //p
All elements of type h1 or p
//div[contains(@class,'1')]
All div elements which has an attribute class that contains the letter 1

You may find it usefull to experiment with a tool like XPath Explorer from sourceforge.net.

.

Advices and warnings

A few advices, warnings and observations that may simplify your work with WXT

- Addresscalculations are an expensive job in WXT. If you are not depending on addresscalculations, it is a good idea to turn off the addressing mechanisme in the script:
  ```
  <addressing tagname="_none" attribute="_none"/>
  ```
  See Addressing for detals of this element.
- MSIE does not seem to like XML-extensions on XHTML-pages when the media-type is set to text/xml. The solution seems to be: Allways use mediatype text/html when we make pages which are strict XHTML or transitional. (and we dont make other HTML-variants, do we ?)
- It is expensive to initiate new instances of WXT from a script. It may tempting and practical, but it costs more processing time than to expand the script with another group. An alternative is to use bootstrapping, see Bootstrap.
- It has turned out very efficient to combine all the possibilities of ANT (http://ant.apache.org/) with WXT. ANT may help you copy files and catalogs, producing zip-archives etc. You may call ANT from a WXT-script and you may start WXT from ANT.

The Script

The script is a XML-file. It is possible to describe any job WXT can perform, small or big, by means of a script. An outline of a XML version 1.0 script:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<script version="1.0">
  <group name="..." location="...">
    <template name="P" location="..."/>
    <template name="I" location="..."/>
    <page name="..." template="I" location="...">
      <content location="..."/>
    </page>
    <page name="..." template="P" location="...">
      <content location="..."/>
    </page>
  </group>
  <group name="..." location="...">
    <template name="P" location="..."/>
    <page name="..." template="P" location="...">
      <content location="..."/>
    </page>
  </group>
</script>
```

The script consists of one or more groups. Each group consists of pages.

.

Welcome

WXT is: a Web and Xml Tool

WXT handles tasks that involves XML-files in general, and it has some special features that are directed towards XHTML.

Wxt is based on the following simple observation:

"if the Internet is wellformed, you can reach and collect everything with an URL and an XPATH-expression".

The fact that the Internet is not wellformed, may be a concern to those of us with a sceptical inclination, so it may also be usefull to access content that is not wellformed as well.

WXT can assist when you:

- build and maintain websites based on XHTML.
- merge and/or split XML-files and transform them on the way to produce documents of allmost any kind.
- combine the two tasks and build XHTML-pages and sites from any wellformed XML-rawmaterial.
- want to include text that is not wellformed into a wellformed surrounding.
- combine the tasks and build XHTML-pages and sites

WXT is based on a few main principles:

- It does not introduce any hidden or proprietary formats or files. You may at any time supplement WXT with other tools or abandon WXT completely.
- It invites you to separate content, style and structure.
- It is based on a script that competely defines a job, small or big. There is a GUI to monitor scripts and jobs, but WXT works as well from the command line.
- You may use your editor of choice to prepare material.

WXT does not mess things up for you since:

- It never deletes any files
- It never change the content of other files than those you ask it to build.

WXT is

- completely free and distributed "as is"
- written in Java and testet extensively on MS Windows, and to some extent on Linux and MacOS

.

Script: script-element

The script may contain only one script-element, which must be the root-element.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<script version="1.0">
  ...
</script>
```

The attribute is mandatory

version The actual version of the script. Must be "1.0".

The scriptelement may contain any number of the following elements:

- option
- program
- addressing
- property
- pathfragment
- template
- transformation
- group (at least one)

.

Script: group-element

The script may contain any number of groups. A group encloses any number of pages that are known to each other as members of TOCs. A group element may only be direct child of the script element.

```
<group name="" location="" pubaddress="">
  ...
</group>
```

name(mandatory)	Any string that identifies the group
location(mandatory)	An absolute URI identifying a catalog. This URI will serve as anchor for all locations in the group. You may also use the predefined pathfragment **{_scriptpath}** to identify the catalog which contains the script. The simple form: _scriptpath, without {} is deprecated. See pathfragment
pubaddress(optional)	An absolute URI where this group will be published.

A group element may contain any number of the following elements:

- description (only one)
- commons
- option
- property
- pathfragment
- template
- transformation
- command
- bootstrap
- page

Examples:

```
<group name="MyBook" location="{_scriptpath}">
  ...
</group>
<group name="YourBook" location="c:\books\book1">
  ...
</group>
<group name="MyBook" location="c:\mine\" pubaddress="http://www.library/mybook/">
  ...
</group>
```

.

Script: page-element

A group element may contain any number of page elements and a page element may contain any number of page elements. A page is the main building block in WXT. A page is normally produced by populating a template with material from content files.

```
<group name="" location="" template="" transformation="">
  <description>...</description>
  <content location=""/>
  ...
</group>
```

A page element has the following attributes:

name(mandatory)	Any string that identifies the page. This name will appear in TOCs etc.
location(mandatory)	An URI which may be absolute (file) or a relative to the owning groups location. The page will be written to this location.
template(mandatory)	A template identifier (see: template-element) or a location for a template. The location must be absolute or relative to the groups address.
transformation(optional)	A transformation identifier (see: transformation-element) or a grouprelative or absolute location for a transformationfile. You may use parameters in the transformation. This transformation is applied to the page as the last operation before it is written.
book(optional)	The book(s) we want to associate this page to. A comma-separated list of strings. May be used in PI-collect

A page element may contain any number of the following elements:

- description (only one)
- option
- property
- thumb
- content
- textcontent
- page

Examples:

```
<page name="Audience" template="P" location="pub/who.html">
  <description>Who is WXT made for</description>
  <content location="content/b-who.xml"/>
</page>
...
<page name="All White Wine"
          template="source/allwines.xml"
          location="source/allwhitewine.xml"
          transformation="SELECT(theSelector='white')"/>
...
<page name="Red Italian Wine" template="P"
      location="RedItalianWines.html"
      transformation="NTUT(theCountry='Italia',theType='red')">
  <description>All Italian red wines in XHTML-format</description>
  <content location="source/allwines.xml"/>
</page>
```

Script: content-element

A page page is populated by content. A page may have any number of content-, textcontent- or dbcontent elements.

```
<content location="" transformation=""/>
```

A content element has the following attributes:

location(mandatory)	An URI which may be absolute (file or http) or a relative to the owning groups location. The content will be read from this location.
backup(optional)	An URI which will serve as an alternative location if the URI in location is unavalable.
transformation(optional)	A transformation identifier (see: transformation-element) or a grouprelative or absolute location for a transformationfile. You may use parameters in the transformation. This transformation is applied to the content before is is used to build a page.

A content element can not have any children.

Example:

```
<content location="blocks/b-p2.xml"/>
```

.

Script: textcontent-element

A page page is populated by content. A page may have any number of content-, textcontent- or dbcontent elements.

A textcontent file can be any textfile, and must thus not be wellformed XML. See the processing instruction PI:import-text for a further explanation of how the actual import-command may be taylored.

```
<textcontent location=""/>
```

A textcontent element have the following attributes:

location	An URI which may be absolute (file or http) or a relative to the owning groups location. The textcontent will be read from this location.
backup(optional)	An URI which will serve as an alternative location if the URI in location is unavalable.
leftpar(optional)	A string determining the start of the text that should be imported
rightpar(optional)	A string determining the end of the text that should be imported
select (optional)	Has only meaning when leftpar and rightpar is set. Select describes which of possibly many text fragments that should be included. Possible values are: • all • random • n:m **all** is default. **random** picks one fragment at random fromr the available textfragments. n and m are integers describing low and hi index of the fragments we want to use. Indexing starts at 1. select="1" use the first fragment. Negative numbers start counting in the right end, select="-1" selects the last fragment. select="1:3" select the three first fragments.
replace (optional)	Will do a simple replace in the collected string, before possible codeformatting or parsing. Format is replace="old|new". Will replace "old" with "new". The divider, |, may be changed by option: **replace_divider** in script. | is default. You may have many replaces: replace, replace1,replace2 etc. Not suitable for large replace operations, just for simple masking of addresses, passwords etc.
transformation(optional)	A transformation that should be applied to the source before the text is extracted

leftpar, rightpar and transformation may also be specified in the actual import PI, see: PI:import-text, and overrides the specification given here in the script-element.

A textcontent element can not have any children.

Example:

```
<textcontent location="../../myprograms/prog1.c"/>
```

.

Script: dbcontent-element

This is deprecated and will not work. Wrap your database import in a normal content-file with PI:import-db

.

Script: template-elements

A template element identifies a template, it maps a name to a location. A template element may appear as child of a group or as child of the script. In the latter case it must have an absolute location.

```
<template name="" location="" transformation=""/>
```

The attributes are:

name(mandatory)	Any string that identifies the template. This name may be used when assigning a template to a page.
location(mandatory)	An URI which may be absolute (file or http) or a relative to the owning groups location. The template will read from this location.
backup(optional)	An URI which will serve as an alternative location if the URI in location is unavalable.
transformation (optional)	A transformation identifier (see: transformation-element) or a grouprelative or absolute location for a transformationfile. You may use parameters in the transformation. This transformation is applied to the template before is is used to build a page.

A template element may have no children.

Example:

```
<template name="P" location="templates/p-template.xml"/>
```

Script: transformation-element

A transformation element identifies a transformation, it maps a name to a location. A transformation element may appear as child of a group.

```
<transformation name="" location="">
  ...
</transformation>
```

The attributes are:

name(mandatory)	Any string that identifies the transformation. This name may be used when assigning a transformation to a page, a template or a content element.
location(mandatory)	An URI which may be absolute (file or http) or a relative to the owning groups location. The transformation will be read from this location.
backup(optional)	An URI which will serve as an alternative location if the URI in location is unavalable.

A transformation element may contain any number of the following elements:

- parameter
- buildoption

Examples:

```
<transformation name="ALL" location="transform/all.xslt"/>
<transformation name="SELECT_WHITE" location="transform/all.xslt">
  <parameter name="theSelection" value="white"/>
</transformation>
```

.

Script: command-element

command elements may appear as direct children of a group. A command instructs WXT to execute an external command. commands are executed after pages in the same group, but before bootstraps. Any commands or bootstraps usefull for preprocessing should be placed in a separate group.

```
<command line="" wait="yes"/>
```

The attribute line is equivalent to the command we would issue from the commandline. WXT identifies **_scriptpath** as part of the line. Line is mandatory.

The optional attribute wait may have values yes | no. yes is default. wait="no" instructs WXT to continue without waiting for the command to finish.

A command element may not have any children.

Examples:

```
<command line="c:\fop\fop.bat _scriptpath/wines.fo _scriptpath/wines.pdf"/>
<command wait="no" line="c:\diverse\xslfoexs\show.bat"/>
```

.

Script: bootstrap-element

bootstrap elements may appear as direct children of a group. A bootstrap instructs WXT to execute a WXT-script in the background. This script may be the result of a page-build operation. Bootstraps are executed after commands (which in turn are executed after pages) Any bootstraps or commands usefull for preprocessing should be placed in a separate group. You may start a new instance of WXT with a command in stead of bootstrapping, but bootstrapping is more efficient.

```
<bootstrap location=""/>
```

The attribute location is the location (absolute or relative to the group) of the file that must be a legal WXT-script. Location is mandatory.

A bootstrap element may not have any children.

Examples:

```
<bootstrap location="myscript.xml"/>
```

or combined with a page

```
<page name="p" location="myscript.xml" template="T" transformation="TR"/>
<bootstrap location="myscript.xml"/>
```

.

Script: property-element

Property elements may appear anywhere in a script, and effects all pagebuilding within its scope.

```
<property name="" value="">
   ... any legal XML-construct ...
</property>
```

The attributes are

name (mandatory)	Any string that identifies the property
location (optional)	An uri which is absolute or relative to the script. The uri should contain a wellformed XML-fragment as a string. It should not contain a XML-header nor a property-element

A property may have any wellformed XML-fragment as child(ren). The following are legal property elements:

```
<property name="hitcounter">
  <!--#exec cgi="../cgi-bin/teller.cgi" -->
</property>

<property name="mymail">
  <a href="mailto:b.s@h.no">mail me</a>
</property>

<property name="greetings">
  Hello all !
</property>

<property name="greetings" location="hello.txt"/>

<property name="toppen">
    <div class="printhide" style="text-align:right">
    <hr style="color:#EED9B0;size:1px"/>
    <a href="#TopOfPage">
    <img style="border-style:none" src="gfx/up.gif" alt="beginning of page"/>
    </a>
    </div>
</property>
```

Properties are available in Processing Instruction, see PI:property.

.

Script: commons-element

A group may have one child element of type common. The purpose is to tell WXT where to find common resources like logo's, stylesheets, etc. that normally will appear on pages located at different catalogs.

```
<commons name="" location=""/>
```

The attributes are mandatory

name	Any string that will be used to identify common resources. Default value is "gfx".
location	A grouprelative URI that tells where these resources are acually stored. Default value is "shared/".

Whenever WXT encounters a link that starts with the "gfx", "../gfx", ../../gfx etc., it will calculate the correct address to the catalog shared.

common elements may have no children.

Example:

```
<commons name="abc" location="mystuff/"/>
```

.

Script: addressing-element

Addressing elements may appear anywhere in a script, and effects all pagebuilding within its scope.

```
<addressing tagname="" attribute="">
```

The two attributes are mandatory, and are explained below.

WXT has a mechanisme for recalculating references when necessary. By default the following attributes are recalculated:

- <img **src** ... >
- <a **href** ... >
- <link **href** ... >

You may extend this repetoire by an addressing element, like this:

```
<addressing tagname="applet" attribute="codebase">
```

You may turn off the addressing mechanisme completely (to save processing time) by this:

```
<addressing tagname="_none" attribute="_none">
```

An addressing element may have no children.

.

Script: buildoption-element

Any number of buildoption elements may be children of a transformation element. The buildoption elements may be used to override or supplement the buildoptions which are spesified in the transformation-file itself. The build options match the options available in the <xsl:output/> - element in the transformation file.

```
  <buildoption name="" value=""/>
```

The attributes are mandatory

name A string that matches the available attributes in the <xsl:output/> - element in the transformation file.
value Any legal value, string, that you want to assign to the parameter. The value must reckogizable by the transfromation engine. Consult XSLT-documentation for more information.

A buildoption element may not have any children.

Examples:

```
<transformation name="T" location="trans/t1.xslt">
  <buildoption name="METHOD" value="xml"/>
  <buildoption name="ENCODING" value="UTF-8"/>
  <buildoption name="OMIT_XML_DECLARATION" value="no"/>
  <buildoption name="STANDALONE" value="yes"/>
</transformation>
```

.

Script: description-element

The group or page may contain one description-element. The element has no attributes and should contain a simple string. The description is available as a property in a Processing Instruction.

```
<description>
  ...
</description>
```

.

Script: option-element

An option element may appear anywhere in a script, and effects all pagebuilding within its scope.

```
<option name="" value=""/>
```

At the moment these options are reckognized by WXT:

```
<option name="encoding" value=""/>
<option name="logfile" value=""/>
<option name="fetch_absolutes" value=""/>
<option name="color_code" value=""/>
<option name="replace_divider" value=""/>
<option name="expand_all" value=""/>
```

A option element may have no children.

encoding

Where UTF-8 is the default value.

When WXT sets encoding for pages it tries to determine what encoding is set in the template for that page. There are some situations where this is ambigiuos or impossible. If WXT fails, it will use the value of the encoding option as a last resort. This option will not override encoding found in the files.

Example:

```
<option name="encoding" value="ISO-8859-1"/>
```

logfile

WXT can be set to accumulate reports from each parse, build or control on a logfile. You may set a logfile by the logfile option. The value of a logfile option is a location which is absolute or is interpreted as relative to the script. The logfile is reset, emptied, when it reaches a size of 100KB.

A option/logfile-element must be placed as direct child of the script-element to have any effect.

Example:

```
<option name="logfile" value="local_work_log.txt"/>
```

fetch_absolutes

When WXT encounters an absolute http-reference to any of the resources mentioned in the addressing-element, you may ask it to download a copy of this element to a local catalog and change the reference accordingly. You do this by setting this option to 'yes'. Default is 'no'.

NOTE: that this is implemented for images only at the present.

This may slow down the building process considerably, and is usefull only when the source (content) that refer to the resource is unstable.

Example:

```
<option name="fetch_absolutes" value="yes"/>
```

color_code

When this option is set with value yes, WXT will identify all programcode-segments which are included as: <pre class="nncode">..</pre> and attempt to colorcode the textcontent as programcode. nn may have values: java, c, cpp, csharp, python or xslt. The effect of this is dependant of the exsistence of following style-classes: code, nncode, nnword, nnliteral and nncomment.

When this option is set with value no, WXT will ignore colorcoding. It is still possible to use color coding of programcode as a possibility in the processing instruction: import-text, by using the attribute type. See import-text

Default is no.

Example:

```
<option name="color_code" value="yes"/>
```

replace_divider

Used in attribute **replace** in PI:import-text and in script element textcontent. Default value is |. See import-text and textcontent

Example:

```
<option name="replace_divider" value=";"/>
```

expand_all

All AJAX-expansions are expanded on build. Possible values are yes | no. no is default. See PI's request and import-text

Example:

```
<option name="expand_all" value="yes"/>
```

.

Script: parameter-element

Any number of parameter elements may be children of a transformation-element.

```
  <parameter name="" value=""/>
```

The attributes are mandatory

name	A string that matches the global parameter in the XSLT-file that realizes the transformation.
value	The value, string, that you want to assign to the parameter.

Parameter elements may have no children.

Example:

```
<transformation name="SELECT_WHITE" location="transform/all.xslt">
  <parameter name="theSelection" value="white"/>
</transformation>
```

Note that you may also assign parameters to a transformation when it is used in a page- or content element. Assume that the transformation NTUT has been defined i a transfomation element:

```
<page name="RedItalian" template="P"
  location="RedItalianWines.html"
  transformation="NTUT(theCountry='Italia',theType='red')">
  <description>All Italian red wines </description>
  <content location="source/allwines.xml"/>
</page>
```

.

Script: pathfragment-element

The pathfragment element may contain a part of a path, typically a catalogpath. The defined fragment may be used in location attributes in the script.

```
<pathfragment name="" value=""/>
```

The attributes are mandatory

name The name of the fragment.
value A string which will be used as a fragment of a path.

pathfragment-elements may be children of the script-node or a group-node. Pathfragments may be used in any location-string in the script by enclosing it in {}.

The pathfragment: **_scriptpath** is allways available, and denotes the catalog where the script resides.

The pathfragment element may not contain any elements.

Examples:

```
<pathfragment name="_src" value="c:/myfiles/">
<pathfragment name="_src2" location="C:/myotherfiles/">
```

Examples of usage:

```
    ...
    <group name="mygroup" location="{_scriptpath}">
    ...
    </group>
    ...

    <content location="{_src}file1.xml">
     ...
```

.

Script: program-element

The script element may contain any number of program elements. The program elements identifies helper programs that is triggered by clicking on pages in the GUI. Thus they have no other mission than to make it easy for the user to inspect a file by calling the browser or edit a file by calling the editor.

```
<program name="" location=""/>
```

The attributes are mandatory

name	The name of a program. The only effective names at the moment are: **_editor** and **_browser**.
location	An absolute URI to the program.

The programelement may not contain any elements.

Examples:

```
<program name="_editor" location="c:\Program Files\UltraEdit\uedit32.exe"/>
<program name="_browser" location="C:\Program Files\mozilla.org\Mozilla\mozilla.exe"/>
```

.

Script: thumb-element

A thumb element can only exist as a direct child of a page element.

```
<thumb location=""/>
```

A thumb element must have the following attribute:

location An URI whitch may be absolute (file or http) or relative to the owning groups location. WXT will assume that the URI locates an image.

A thumb element can have no children

.

Validation

You may validate your script according to the scheme quoted below. This scheme is distributed with wxt and may be used from the GUI, or you may use any other tool you want to validate your script.

Note that the validation in some respects perform a more strict control than wxt. WXT is more relaxed when it comes to the sequence of elements. You may thus live happily with a script that does not validate, but validation is an assurance that you have a usefull script.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="commons">
    <xs:complexType>
      <xs:attribute name="name" type="xs:string" use="required"/>
      <xs:attribute name="location" type="xs:anyURI" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="addressing">
    <xs:complexType>
      <xs:attribute name="tagname" type="xs:string" use="required"/>
      <xs:attribute name="attribute" type="xs:string" use="optional"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="content">
    <xs:complexType>
      <xs:attribute name="location" type="xs:string" use="required"/>
      <xs:attribute name="transformation" type="xs:string" use="optional"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="command">
    <xs:complexType>
      <xs:attribute name="line" type="xs:string" use="required"/>
      <xs:attribute name="wait" type="xs:string" use="optional"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="textcontent">
    <xs:complexType>
      <xs:attribute name="location" type="xs:string" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="description" type="xs:string"/>
  <xs:element name="group">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="description" minOccurs="0"/>
        <xs:element ref="commons" minOccurs="0"/>
        <xs:element ref="option" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="property" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="template" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="transformation" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="command" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="page" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="name" type="xs:string" use="required"/>
      <xs:attribute name="location" type="xs:anyURI" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="option">
    <xs:complexType>
      <xs:attribute name="name" type="xs:string" use="required"/>
      <xs:attribute name="value" type="xs:string" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="page">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="description" minOccurs="0"/>
        <xs:element ref="option" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="property" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="thumb" minOccurs="0"/>
        <xs:element ref="content" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="textcontent" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="page" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="name" type="xs:string" use="required"/>
      <xs:attribute name="template" type="xs:string" use="required"/>
      <xs:attribute name="location" type="xs:anyURI" use="required"/>
      <xs:attribute name="transformation" type="xs:string" use="optional"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="program">
    <xs:complexType>
      <xs:attribute name="name" type="xs:string" use="required"/>
      <xs:attribute name="location" type="xs:anyURI" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="script">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="option" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="program" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="addressing" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="property" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="template" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="transformation" minOccurs="0" maxOccurs="unbounded"/>
```

```
        <xs:element ref="group" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="version" type="xs:decimal" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="template">
    <xs:complexType>
      <xs:attribute name="name" type="xs:string" use="required"/>
      <xs:attribute name="location" type="xs:anyURI" use="required"/>
      <xs:attribute name="transformation" type="xs:string" use="optional"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="transformation">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="parameter" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="buildoption" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="name" type="xs:string" use="required"/>
      <xs:attribute name="location" type="xs:anyURI" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="thumb">
    <xs:complexType>
      <xs:attribute name="location" type="xs:anyURI" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="parameter">
    <xs:complexType>
      <xs:attribute name="name" type="xs:string" use="required"/>
      <xs:attribute name="value" type="xs:string" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="buildoption">
    <xs:complexType>
      <xs:attribute name="name" type="xs:string" use="required"/>
      <xs:attribute name="value" type="xs:string" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="property">
    <xs:complexType mixed="true">
      <xs:sequence minOccurs="0">
        <xs:any processContents="skip" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="name" type="xs:string" use="required"/>
      <xs:attribute name="value" type="xs:string" use="optional"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

.

Dating elements in the script

All elements in the script may have a limited lifespan. You can spesify the first and last date for the element to be used. This is controlled by two attributes that are not documented for each element.

Optional attributes for all elements

firstdate	The first day this element will be used
lastdate	The last day this element will be used

The form of the attributes are: yyyy-mm-dd. You may use one, both or none. If none of them is present the element is valid at all times.

Examples:

```
<page name="Solutions" firstdate="20050401">
  ...
</page>
...
<property name="greetings" firstdate="2005-12-24" lastdate="2006-01-01">
  Merry Xmas
</property>
```

Processing Instructions, PI

Processing Instructions (PI) are legal XML elements. PIs have the general form:

```
<?name ..any text interpreted by the interested program ...?>
```

WXT has defined its own PIs which has the name **_wxt**. PIs may occur in template files and content files. Typically we embed PIs in the template to populate it with extracts from content.

Some PI's are general and will work on any wellformed XML-file. Some is by their intention related to XHTML. Some of the latter are also XHTML-spesific in the sense that they introduce XHTML-elements into the page. Typically a table of content will introduce (X)HTML elements like a, div, span and even table.

The text in a _wxt PI is organized as name - value pairs, like attributes. This document use the term parameter. This example instructs WXT to import all div elements with attribute id main from myfile.xml:

```
<?_wxt import uri="myfile.xml" xpath="//div[@id='main']"?>
```

PI's may be dated, just like script elements, see dating. You can spesify a first and/or last date for a PI to be expanded. The dateform is yyyymmdd, or yyyy-mm-dd

```
<?_wxt import uri="myfile.xml" xpath="//div[@id='xmas']" firstdate="2005-12-24" lastdate"2006-01-01"?>
```

.

PI: collect

The purpose is to collect fragments from other pages.

```
<?_wxt collect pages="" xpath=""?>
```

The parameters are:

pagepath (optional)	Identifies the pages we will collect from by an xpath which is applied on the script. For instance "//page" which means all pages in the script. No default.
pages (optional)	Identifies the pages we will collect from. Pages may be identified by a comma-separated list. Each entry in the list may be a pagename or a (partial) path to the pages file. Each entry may have appendix "+" which indicates that all children of the page is included. Pages from an other group may be included with groupname# as prefix. No default.
book (optional)	Identifies the pages we will collect from by attribute book. May be a commaseparated list of book-id's. Pages are identified across groups. No default.
xpath (mandatory)	An xpath that identifies the fragments we want to collect from each of the selected pages.

The three selection methods (book, pagepath, pages) may be combined, but a page is selcted only once.

A few examples using pagepath (remember that the path must select elements of type **page**):

```
<?_wxt collect pagepath="//group[@name='Group4']/page"
           xpath="//div[@id='collectable']"?>
<?_wxt collect pagepath="//group[@name="Chapter2"]/descendant-or-self::page"
           xpath="//div[@class='collectable']"?>
<?_wxt collect pagepath="//group[@name="Chapter2" or @name="Chapter3"]/descendant-or-self::page"
           xpath="//div[@class='collectable']"?>
```

A few examples using pages:

```
<?_wxt collect pages="A,pages/c.xml+,ref"
           xpath="//div[@class='collectable']/*" ?>
<?_wxt collect pages="Group4#_all"
           xpath="//div[@class='collectable']"?>
```

.

PI: date

Purpose is to set a date mark.

```
<?_wxt date language="" form=""?>
```

The parameters are:

language (optional)	According to locale spesification. "no" is norwegian, "en" is english etc.
form (optional)	Values are: SHORT , MEDIUM , LONG, FULL. MEDIUM is default.

PI: daydiff

Purpose is to produce difference between two dates

```
<?_wxt daydiff firstday="" lastday=""?>
```

The parameters are:

firstday (optional)	A date in the form: yyyy-mm-dd. If the year-part is set to xxxx, the date is considered a repeating date, repeated each year.
lastday (optional)	A date in the form: yyyy-mm-dd. If the year-part is set to xxxx, the date is considered a repeating date, repeated each year.
absolute (optional)	The PI will allways return a positive integer, absolute value.

If one of the day-attributes is unspesified, the date is set to the date of building. At least one of the day-attributes must be set.

A few examples:

```
<?_wxt daydiff firstday="xxxx-12-24"?>
<?_wxt daydiff lastday="xxxx-12-24"?>
```

will in trurn produce the days since christmas and the days until christmas.

.

PI: import-db

The purpose of an import-db PI is to produce an XML-fragment from a databasequery result and replace the PI with this fragment.

```
<?_wxt import-db
    host=""
    database=""
    databasetype=""
    driver=""
    user=""
    password=""
    sql=""
    transformation=""
    xpath=""
    uri="" ?>
```

Alternative form

```
<?_wxt import-db
    connectionstring=""
    sql=""
    transformation=""
    xpath=""
    uri="" ?>
```

The parameters are:

uri(optional and recommanded)	A file-URI which may be absolute or relative to the owning groups location. The result of a database transaction will be **written** to this location. The result may be a support when writing transformations and will serve as a backup if the database is temporary unavailable.
transformation(optional)	A transformation identifier (see: transformation-element) or a grouprelative or absolute location for a transformationfile. You may use parameters in the transformation. This transformation is applied to the content before it is imported. You may use the word: **_table** for the transformation. This will produce a set of HTML tables describing the result of the database transactions in a simple way. Omitting the transformation will simply return all result-rows from the database transaction(s).
xpathoptional	Identifies what we want to extract from the (transformed) databasequery result.
database(mandatory or in connectionstring)	A string which identifies the database.
databasetype(mandatory or in connectionstring)	A string which identifies the databasetype.
host(mandatory or in connectionstring)	A string which identifies the host.
user(mandatory or in connectionstring)	The user of the database.
password(mandatory or in connectionstring)	The users password.
driver(optional)	The databsedriver to be used. WXT reckognize the shortforms **psql**(postgres) and **mysql**, and attempts to find a driver for those databases. You may spesify any driver as long as you are sure that the driver is available in the classpath.
sql(optional)	SQL-statements of type SELECT, separated by semicolon. Overrides sqlfile
sqlfile(optional)	A file which contains SQL-statements of type SELECT, separated by semicolon.

Some examples:

```
<dbcontent
    uri="frascript/contentproduced.xml"
    database="db1"
    host="localhost"
    driver="org.gjt.mm.mysql.Driver"
    user="bs"
    password="thepassword"
    sql="SELECT * FROM perstab"
    transformation="_table"/>
<dbcontent
    connectionstring="jdbc:mysql://localhost/db1?user=bs&password=thepassword"
    uri="frascript/contentproduced.xml"
    sql="SELECT * FROM perstab"
    transformation="T"
    xpath="first_name"/>
```

.

PI: request

The purpose is to produce a simple expand of a html-fragment on mouseclick. This involves javascript function, see AJAX. This is a simplified version of the mechanismes available in import-text, see import-text. This simple version is faster when the fragment that is expanded is readymade.

```
<?_wxt request request_file="" request_function="" expanded=""?>
```

The parameter are:

request_file (mandatory)	The URI of the page to show. Relative to the page or the siteroot.
request_function (optional)	The name of the javascript function that are called to expand the fragment. Default value is "expand", relying of the library as desribed in AJAX. The parameterlist for your function must be as in default function "expand".
expanded (optional)	Values are yes | no. no is default. If yes: The fragment is showed initially, with unexpand-functionality.

NOTE also that this mechanisme uses the style-class "onoff", which styles the expand and collapse nodes (button-lookalikes).

Some examples:

```
<?_wxt request request_file="sometext.txt"?>
```

.

PI: import

The purpose of an import PI is to produce an XML-fragment and replace the PI with this fragment.

```
<?_wxt import xpath="" uri="" transformation=""?>
```

The parameters are:

xpath (mandatory)	Any xpath expression that identifies a nodeset that will be treated like a XML-fragment.
uri (optional in templates)	The URI of the file we want to import from. In templates the uri parameter is usually skipped. In this case all content files listed in the script are searched for appropriate content.
transformation (optional)	A transformation identifier (see: transformation-element) or a grouprelative or absolute location for a transformationfile. This transformation may have parameters.

Some examples:

```
<?_wxt import xpath="//h1"?>

<?_wxt import xpath="//div[@class='main']/*"?>

<?_wxt import
        uri="http://www.ia.hiof.no/~borres/ml/index.shtml"
        xpath="//div[@class='main']/*"?>

<?_wxt import uri="../quotes/q.xml"
        xpath="//p[@class='quot']/*"?>

<?_wxt import xpath="//div[@id='main']"
        uri="b.xml"
        transformation="myT(para='hello',parb='goodbye')"?>
```

.

PI: import-text

The purpose is to import text that is not necessarily wellformed XML. The imported text may be modified in several ways and is

- inserrted as a textnode
- inserted as an XML-fragment
- saved to file for possible HTTP Request

```
<?_wxt import-text uri="" leftpar="" rightpar="" select= "" parse="" type="" request_file="" request_function="" expan
```

The parameters are:

uri (optional in templates)	The URI of the file we want to import from. In templates the uri parameter is usually skipped. In this case all content files listed in the script are searched for appropriate content.
leftpar (optional)	Any string in the text that may serve as a left bracket for a selection.
rightpar (optional)	Any string in the text that may serve as a right bracket for a selection.
select (optional)	Has only meaning when leftpar and rightpar is set. Select describes which of possibly many text fragments that should be included. Possible values are: • all • random • n:m **all** is default. **random** picks one fragment at random fromr the available textfragments. n and m are integers describing low and hi index of the fragments we want to use. Indexing starts at 1. select="1" use the first fragment. Negative numbers start counting in the right end, select="-1" selects the last fragment. select="1:3" select the three first fragments.
replace (optional)	Will do a simple replace in the collected string, before possible codeformatting or parsing. Format is replace="old|new". Will replace "old" with "new". The divider, |, may be changed by option: **replace_divider** in script. | is default. You may have many replaces: replace, replace1,replace2 etc. Not suitable for large replace operations, just for simple masking of addresses, passwords etc.
parse (optional)	Possible values are "yes" or "no" with "no" as default. Set this parameter to "yes" if you want WXT to attempt to parse the selected text as an XML-fragment.
type (optional)	Reckognized values are: **javacode**, **javascriptcode**, **cppcode**, **ccode**, **csharpcode**, **xsltcode**, **aspxcode**, **pythoncode**, **mlcode**. If either of these is found WXT attempts to produce programcode with hilited keywords, stringliterals and comments. To get the full effect from this you must supply a stylesheet with the following styles: .nncode, .nnword, .nnliteral and .nncomment, where nn is ml, java, javascript, c, cpp, csharp, xslt, aspx or python. See Styling **Note** that this is not a fullblown language parser, it simply attempts to identify and hilite elements in a codefragment. If the parser fails, it will simply wrap the supposed code in a pre-tag: <pre class="code">..</pre>. You should therefore also prepare the .code - style to meet this possibility.
request_file (optional)	A simple filename. The file will be produced in the catalog of the owning page and an expansion-button will be placed in the page. Relies on a Javascript for a HTTPRequest, see AJAX. The CSS class "onoff" should be set to taylor the look of this expansion-button.
request_function (optional)	A name of the Javascript function that will be involved when we try to expand the fragment. Default value is "expand", see AJAX.
expanded (optional)	Values are yes | no. no is default. Has only meaning when request-file is set. If yes: The fragment is showed initially, with unexpand-functionality.

If either or both of leftpar and rightpar is not set, the whole file is imported. If parse is set and fails, nothing is imported. If type is set, parse is ignored.

If request_file is set, only a simple button-like reference is set in the page.

A few examples:

```
<?_wxt import-text uri="../p/main.c"
          leftpar="//inlude"
          rightpar="//end-include"?>
<?_wxt import-text
         leftpar="<!-- begin -->"
         rightpar="<!-- end -->"
         parse="yes"?>
<?_wxt import-text uri="../code/test.java"
         leftpar="/*START_PART*/"
         rightpar="/*END_PART*/"
         replace="hemmelig|mypassword"
         type="javacode"?>
<?_wxt import-text uri="../code/test.java"
         type="javacode"?
         request_file="test_java.txt">
<?_wxt import-text uri="../code/test.java"
         type="javacode"?
```

```
        request_file="test_java.txt"
        expanded="yes">
```

.

PI: ixword

ixword marks all words that will be included in an indextable.

See also: ixtable

```
<?_wxt ixword word="" indexname=""?>
```

The parameters are:

word (mandatory)	The word we want to index.
indexname (optional)	The name of the indextable we want to include the word in. Default is "_main".

The involved style-classes in inexword and indextable are: **ixword** and **ixentry**. See Styling

.

PI: ixtable

ixtable produced an indextable.

See also: ixword

```
<?_wxt ixtable indexname="" cols=""?>
```

The parameters are:

indexname (optional)	The name of the indextable. Default is "_main".
cols (mandatory)	The number of columns we want in the indextable.

The involved style-classes in indexword and indextable are: **ixword** and **ixentry**. See Styling

.

PI: popup

The purpose is to make a html-fragment that generates a popup-page and replace the PI with this fragment. It involves a javascript function, see below.

```
<?_wxt popup uri="" text="" title="" style="" image=""?>
```

The parameters are:

uri (mandatory)	The URI of the page to show. This URI should be absolute (http://) or should be a correct relative URI. The only correction done is correction according to commons (commons)
text (optional)	The text we want to show to identify the popup. Default is the URI.
title (optional)	Works as title in HTML-hrefs. Default is 'popup'.
style (optional)	The CSS-style we want to use on the popup-element. Default is no style.
image (optional)	An image that we want to represent the popup-element. Will override text. The image-uri should be absolute (http://) or should be a correct relative URI. The only correction done is correction according to commons (commons

NOTE that this mechanisme relies of the existence of a javascript function called 'simplepopup'. This function is part of a very simple JavaScript-library which is available at: http://www.ia.hiof.no/~borres/common/jscripts/std.js.

You may as well write your own. It should take three parameters as should be clear from the sample definition below (which you could copy and work on):

```
function simplepopup(url,wname,wstyle)
  {
          if(wstyle=='*')
                  wstyle='scrollbars=yes,resizable=yes,width=600,height=600,status=0';
          try{
                  newwindow=window.open(url, wname, wstyle);
                  if (window.focus) {newwindow.focus()}
          }
          catch(E){
                  alert('If you have blocked pop-ups on this page\n Press Ctrl-key when clicking');
          }
  }
```

Or you may copy the javascript library at AJAX

Some examples:

```
<?_wxt popup uri="content/pop.html"?>

<?_wxt popup text="Click" title="En enkel popup" uri="content/pop.html" style="popup"?>

<?_wxt popup uri="content/pop.html" image="gfx/home.gif"?>
```

.

PI: property

The purpose is to replace this PI with a property set in the script.

```
<?_wxt property name=""?>
```

A few default properties are available:

- _pagename
- _pagedescription
- _pagefilename, short name (no path) of the pages filename
- _pagethumb
- _groupname
- _groupdescription
- _pageno, no of this page according to a preorder traversal within the group
- _pagecount, number of pages in the group
- _book, the book we want to associate this page to

A property PI will expand the properety as it is defined in the script. Assume:

```
    ...
    <property name="theAuthor">
    <author>
      <name>John Smith</name>
      <street>Elm st. 1</street>
    </author>
    </property>
    ...
<?_wxt property name="theAuthor"?>
```

will expand to:

```
<author>
  <name>John Smith</name>
  <street>Elm st. 1</street>
</author>
```

PI: stamp

Purpose is to set a stamp and a reference back to WXT's homepage

```
<?_wxt stamp version="" icon=""?>
```

The optional parameters are:

version	"yes" displays the buildversion of WXT. "no" displays only the string "WXT" with a reference back to WXT's homepage. Default is "no".
icon	"yes" displays an icon for WXT fetched from WXT's homepage, instead of string "WXT" and version. Default is "no".

icon will suppress version.

.

PI: time

Purpose is to set a time mark.

```
<?_wxt time language="" form=""?>
```

The parameters are:

language (optional)	According to locale spesification. "no" is norwegian, "en" is english etc.
form (optional)	Values are: SHORT , MEDIUM , LONG, FULL. MEDIUM is default.

PI: tocchildren

The purpose is to set up a TOC of all children.

```
<?_wxt tocchildren cols="" divider="" left="" right="" thumb=""?>
```

The parameters are:

cols (optional)	Number of columns. Default is "0", which sets up a line.
left (optional)	A string that will appear left of all entries. Default is the empty string.
right (optional)	A string that will appear right of all entries. Default is the empty string.
divider (optional)	A string that will appear between all entries. Default is the empty string.
thumb (optional)	If the value is "yes", the children are represented by their thumbs as defined in the script.
target (optional)	To facilitate use of frames on a webpage. The refreneced entries will be opened in this frame.

A typical tocchildren PI will look like:

```
<?_wxt tocchildren left="[" right="]"?>
```

The only involved style-classe is: **tocchildren-0**. See Styling

.

PI: tocgroup

The purpose is to set up a TOC of all, or some of the, pages in a group.

```
<?_wxt tocgroup cols="" lo="" hi="" left="" right="" divider=""?>
```

The parameters are:

cols (optional)	Number of columns. Default is "3".
lo (optional)	A numeric value that defines the lowest level of the pages that should be included. Default is "1".
hi (optional)	A numeric value that defines the greatest level of the pages that should be included. Default is "3".
left (optional)	A string that will appear left of all entries. Default is the empty string.
right (optional)	A string that will appear right of all entries. Default is the empty string.
divider (optional)	A string that will appear between all entries. Default is the empty string.
target (optional)	To facilitate use of frames on a webpage. The refreneced entries will be opened in this frame.

A typical tocgroup PI will look like:

```
<?_wxt tocgroup cols="2" lo="1" hi="4"?>
```

The involved style-classes are of the form: **tocgroup-n** and **tocgroup-an**. n is a number greater or equal to 0, and indicates the level of the page. 0 is used when parameter cols is 0. a is used for the active, current page. See Styling

.

PI: tocpage

PI: tocpagefinal

The purpose is to set up a TOC for one page on that page. The two PI's works exactly the same, but they are invoked at different phases in a pagebuild. **tocpagefinal** is invoked after all pages have been built and all collections and indextables have been effectuated, while **tocpage** is invoked during normal pagebuild.

```
<?_wxt tocpage cols="1" hi="3" lo="1" left="" right="" divider="" header="h" ?>
```

The parameters are:

cols (optional)	Number of columns. Default is "1".
hi (optional)	Highest toc-level included. Default is "1".
lo (optional)	Lowest toc-level included. Default is "3".
left (optional)	A string that will appear left of all entries. Default is the empty string.
right (optional)	A string that will appear right of all entries. Default is the empty string.
divider (optional)	A string that will appear between all entries. Default is the empty string.
header (optional)	A string that identifies a tag that describes levels in the page. Default is **h**, which will identify tags: h1, h2, h3 , h4 etc.

A typical tocpage PI may look like:

```
<?_wxt tocpage cols="0" lo="2" hi="2" divider=" ][ "?>
```

The involved style-classes are of the form: **tocpage-n**. n is a number greater or equal to 0, and indicates the level. 0 is used when parameter cols is 0. See Styling

.

PI: tocsiblings

The purpose is to set up a TOC of all siblings.

```
<?_wxt tocsiblings cols="" divider="" left="" right="" thumb=""?>
```

The parameters are:

cols (optional)	Number of columns. Default is "0", which sets up a line.
left (optional)	A string that will appear left of all entries. Default is the empty string.
right (optional)	A string that will appear right of all entries. Default is the empty string.
divider (optional)	A string that will appear between all entries. Default is the empty string.
thumb (optional)	If the value is "yes", the children are represented by their thumbs as defined in the script.
target (optional)	To facilitate use of frames on a webpage. The refreneced entries will be opened in this frame.

A typical toctchildren PI will look like:

```
<?_wxt tocsiblings left="[" right="]"?>
```

The only two involved style-classes are: **tocsiblings-0** and **tocsiblings-a0**, where a is for the current page. See Styling

.

PI: toctrail

The purpose is to set up a trail of references showing the ancestors of the current page.

```
<?_wxt toctrail cols="" divider="" left="" right=""?>
```

The parameters are:

cols (optional)	Number of columns. Default is "0", which sets up a line.
left (optional)	A string that will appear left of all entries. Default is the empty string.
right (optional)	A string that will appear right of all entries. Default is the empty string.
divider (optional)	A string that will appear between all entries. Default is the empty string.

A typical toctrail PI will look like:

```
<?_wxt toctrail divider=" > "?>
```

The only two involved style-classes are: **toctrail-0** and **toctrail-a0**. a is used for the active, current page. See Styling

.

PI: xref

The purpose is to produce a reference to an other page. xref introduces (X)HTML-elements: **a** and optionally **img**.

```
<?_wxt xref pageid="" text="" image=""?>
```

The parameters are:

pageid (mandatory)	Identification of the page we want to refer to. A page may be identified either by name as given in the script, or by a partial path. If the pagid is ambigiuos, the first match in a preorder traversal of the pages is used.
text (optional)	A text that will appear in the produced a-element. If this parameter is omitted, the pages name is used.
image (optional)	The URI to an image that will show the link. The text-parameter, if submitted is used as alt-attribute in the produced img-element. WXT reckognize the value "_thumb" as the thumb specified in the script for the referenced page.
target (optional)	To facilitate use of frames on a webpage. The refreneced page will be opened in the targeted frame.
listname (optional)	Will add this reference to a referencelist. May be a commaseparated list of listnames. See reflist

A few generic page identificators are reckognized:

- _next : The next page in a preorder traversal of the pagestructure, or this page if last
- _prev : The previuos page in a preorder traversal of the pagestructure, or this page if first
- _home : The first child of the group root in a preorder traversal of the pagestructure
- _parent: The parent in the pagestructure, or this page if parent is root
- _firstchild: The first child of the page, or this page if no children
- _nextsibling: The next sibling of the page, or this page if no nest sibling
- _prevsibling: The previous sibling of the page, or this page if no previous sibling

You may set up an xref to a page in an other group by prefixing the pagid by groupname#.

A few examples:

```
<?_wxt xref pageid="_next" image="gfx/next.gif" text="next"?>
<?_wxt xref pageid="_next" image="_thumb" text="next"?>
<?_wxt xref pageid="page4.html"?>
<?_wxt xref pageid="Masterpages#index.html"?>
```

.

PI: ref

ref produce a simple ref-element and introduce it to a reflist.

See also: reflist

```
<?_wxt ref uri="" fields="" listname="" category="" display="" delimiter=""?>
```

The parameters are:

delimiter (optional)	Alternative to | as a list separator in lists in this element. Default is |
fields (mandatory)	A delimiter-separated list of simple text strings.
uri (optional)	An uri to the resource we want to link to, if any.
listname (optional)	A delimiter-separated list of names of the lists we want to include this reference in. The name **_footnote** is reckognized as the name of the page, and should be matched with listname **_footnote** in a reflist-PI on the same page, if you want to collect all references at a page in a list. Default is "_main".
category (optional)	The category of the ref. You are free to name categories at will. Default is "*".
display (optional)	This controls the way the reference is displayed. Possible values are: • **none**. The reference is not displayed • **simple**. The refrenece is displayed as a "ref to footnote way": [n]. • **full**. With link, if any or first field, if no link. • **prefix**. First field is prefixing link, no effect if no link. . Default is simple.

Some examples:

```
<?_wxt ref uri="http://www.w3schools.com/htmldom/" fields="W3org|HTML DOM Tutorial" category="DOM"?>

<?_wxt reflist listname="_main" category="DOM"?>

<?_wxt ref uri="http://www.w3schools.com/htmldom/" listname="_footnote|_main" fields="HTML DOM Tutorial" category="DOM

<?_wxt reflist listname="_footnote" category="DOM"?>
<?_wxt ref fields="Håkon Wium Lie & Bert Bos|
                   Cascading Style Sheets|
                   Addison Wesley|
                   1999|
                   0-201-59625-3|
                   Cover CSS1 and CSS2"
              category="book"
              display="simple"
              listname="_footNote|_main"?>
```

The involved style-classes in a reference and a referenselist are:

- **external** mark a reference as external
- **reflist_ul** styling the ul-element in a reflist.
- **reflist_li** styling the li-element in a reflist.
- **nolink_span** styling a simple [] .
- **ref_field_n** where **n** is a number in the range [0..m], m is the maximum number of fields you plan to use in a ref field.
- **ref_link** styling all links in a reference list.
- **external** styling an absolute link in a reference list.

See Styling

.

PI: reflist

reflist produced an unordered list of external references.

See also: ref

```
<?_wxt reflist listname="" category=""?>
```

The parameters are:

listname (optional)	The name of the referencelist. Note the usage of listname="_footnote" to make a fotnote-list of references. Will collect all references (PI:ref or PI:xref) on the same page with _footnote as (one of the names in) listname. Default is "_main".
category (optional)	A commaseparated list of categories we want to include in the list. Default is all categories: "*".

Some examples:

```
<?_wxt ref uri="http://www.w3schools.com/htmldom/" fields="W3org|HTML DOM Tutorial" category="DOM"?>

<?_wxt reflist listname="_main" category="DOM"?>

<?_wxt ref uri="http://www.w3schools.com/htmldom/" listname="_footnote|_main" fields="HTML DOM Tutorial" category="DOM

<?_wxt reflist listname="_footnote" category="DOM"?>
<?_wxt ref fields="Håkon Wium Lie & Bert Bos|
                   Cascading Style Sheets|
                   Addison Wesley|
                   1999|
                   0-201-59625-3|
                   Cover CSS1 and CSS2"
               category="book"
               display="simple"
               listname="_footNote|_main"?>
```

The involved style-classes in a reference and a referenselist are:

- **external** mark a reference as external
- **reflist_ul** styling the ul-element in a reflist.
- **reflist_li** styling the li-element in a reflist.
- **nolink_span** styling a simple [] .
- **ref_field_n** where **n** is a number in the range [0..m], m is the maximum number of fields you plan to use in a ref field.
- **ref_link** styling all links in a reference list.
- **external** styling an absolute link in a reference list.

See Styling

.

PI: demoref

demoref produce a refrence to a file. This reference will lokk different on screen than on paper. On paper it will an absolute reference. This element is handy when you want to make links to demopages of som ekind. Sort of "sidebar"-material.

```
<?_wxt demoref uri="" text="" demostart=""?>
```

The parameters are:

uri (mandatory)	An absolute href or a href relative to the group, the page or a contentfile
text (optional)	The text that will appear on the screen. If omitted the URI will be shown.
demostart (optional)	The text that will appear before the URL on print. Default value is "DEMO: ".

Expected behaviour is depending on the pubaddress-attribute in the group - element.

Some examples:

```
<?_wxt demoref uri="myfile.html" text="See this example" ?>

<?_wxt demoref uri="yourfile.html" text="See this example" demostart="Link to demo" ?>

<?_wxt demoref uri="http://www.ia.hiof.no/~borres/self/bs.gif" text="Nice guy"?>
```

The involved style-classes in a reference and a referenselist are:

- **demo-div** wrapping the element
- **demo-link** the link
- **demo-screen** the text associated with the link that will appear on screen
- **demo-print** the text associated with the link that will appear on print

You must define media screen and media print in the stylesheet and turn visibility on and off on the two: demo-screen and demo-print.

See Styling

.

Download and installation

[Download WXT][Download demos][History]

WXT is written in Java. The XPATH 1.0 and XSLT 1.0 functionality is realized with XALAN 2.6.0. WXT has been tested thoroughly on Windows and to some extent on Linux and MacOs.

.

Download WXT

If you have not installed a new version of JRE, you should do that first: 

Then:

- Download this zip-file: downloadWxt.zip, and unzip it in a catalog of your choice. Leave all files in that catalog.
- Inspect and edit the file **run.bat** to match your installation path.
- Run it.

... or any other procedure necessary to run the jar-file **wxt.jar**

.

Download demos

You can download a small set of simple demos. Download, unpack and open the script.xml -files i WXT: demozips.zip

.

History

Jan 04. 2008

Build: 1.0 - 20080104

Introducing PI's: **demoref**. This makes it possible to produce refrences that look different on screen than on paper.

Introducing attribute **book** in page-elements in the script. May be used in PI-collect and may be used by other programs to collect pages and pageinformation from a WXT-script. Available as property "_book".

Introducing attribute **pubaddress** in group-elements in the script. Available as property "_pubaddress".

Nov 22. 2007

Build: 1.0 - 20071122

Introducing PI's: **ref** and **reflist**. This makes it possible to collect and display references of any type in a simple way. It also facilitates a crude footnote-implementation.

Note that PI **xref** is also updated with an attribute, listname, which makes it possible to include crossrefrences in reference lists.

Sept 9. 2007

Build: 1.0 - 20070909

Fixing addressing problem for popup pages.

June 25. 2007

Build: 1.0 - 20070625

Introducing flexible expand/noexpand control of AJAX-requests.

Involves script option: expand_all and attribute expand on PIs: request and import-text.

May 22. 2007

Build: 1.0 - 20070522

Correcting pagecollection in PI collect

May 19. 2007

Build: 1.0 - 20070519

Minor updates of help files

Dec. 29. 2006

Build: 1.0 - 20061229

HTTPRequest included:
PI:request for simple fileaccess, see Request
PI:import-text for more sophisticated datapreparation, see TextImport
See also Ajax

Oct. 22. 2006

Build: 1.0 - 20061022

Minor corrections in codeformatting.
Added attribute replace in text import

Oct. 11. 2006

Build: 1.0 - 20061011

Minor corrections in codeformatting.
Selective text import from a candidate list is possible.

Sept. 04. 2006

Build: 1.0 - 20060904

Minor corrections.
References to page-fragments are simplified in form.
Pagedescritions are included as "title"-attributes in indextables.

July. 15. 2006

Build: 1.0 - 20060715

The property element in the script is changed. You may now use an attribute location to identify a property. The content is supposed to be a wellformed XML-fragment (no XML-header and no property tag). The location should be absolute or relative to the script.

The value-attribute is deprecated.

June. 06. 2006

Build: 1.0 - 20060606

Databaseaccess is changed. It is no longer legal to specify databaseaccess directly in the script. You must wrap the databaseaccess in a normal contentpage by a processing instruction.

One features introduced, see WXT-help:

- daydiff - You may now calculate and display the difference between two dates. Accessible as a processing instruction.

Jan. 08. 2006

Build: 1.0 - 20060108

Refactoring of addresscalculation. A more robust solution for Linux.

Two features introduced, see WXT-help:

- pathfragment - You may now define fragments of paths and use these in the script.
- backup - content, textcontent, template and transformation elements ha an attribute backup which will be used if regular location is unavailabe.

Nov. 21. 2005

Build: 1.0 - 20051120

Added possibility for bootstrapping. That is a script may contain a bootstrap-element that instructs WXT to parse and build an other script in background. Faster than using external command.
<bootstrap location="myfile.xml"/>

Sequence is changed. Pages are now built before commands, which in turn are executed before bootstraps. You may as before change the sequence by using groups.

See WXT-help or The Script

Oct. 21. 2005

Build: 1.0 - 20051030

Databaseaccess added.

Sep. 01. 2005

Build: 1.0 - 20050901

Printing of script added.

Aug. 27. 2005

Build: 1.0 - 20050827

Error corrected in colorcoding, see below.

Aug. 11. 2005

Build: 1.0 - 20050811

Colorcoding of program fragments are improved.
See options in The Script and import-text in The PIs.

Aug. 01. 2005

Build: 1.0 - 20050801

Introducing colorcoded programcode as a possible import-text.

Apr. 04. 2005

Build: 1.0 - 20050404

Introducing threads in the GUI.
Major restructuring of memory strategy.
More robust addresscalculations.

Mar. 31. 2005

Build: 1.0 - 20050331

Simplify the logfile-mechanisme.
Fixing minor problems.

Mar. 28. 2005

Build: 1.0 - 20050328

Added dating of elements.
New attribute in collect.

The version/build string appearing in the about-box is erronous in builds older than this one. The build appears to be up to date at all times. You should download latest version to get it right.

Mar. 23. 2005

Build: 1.0 - 20050323

Added Popup (Processing Instruction)
Minor restructuring of code.

Mar. 13. 2005

Build: 1.0 - 20050313

Added a possibility for a log file that accumulates reports from parse, build and control actions. Available as OPTION in script and can be turned on/off from GUI.
Removed the annoying delay (due to syntax hilighting)when loading a script.
Minor restructuring of code.

Dec. 12. 2004

Build: 1.0 - 200411212

GUI modified
Added possibility to control links.

Nov. 28. 2004

Build: 1.0 - 20041128

Added target as an optional parameter to some TOC PI's and to the XREF PI.
Facilitating use of frames.

Oct. 02. 2004

Some adjustments in reporting.
Introduction of attribute wait in element command.
Schema is uppdated accordingly.

Sep. 17. 2004

WXT has been in production on several projects under MSWindows for some time. It seems to work according to expectations.

No serious testing done under Linux and MacOs.

.

References

- World Wide Web Concortium, W2C. Links to a lot of definitions and tutorials. You will find sources for information on XML, XSLT, XSL-FO, XPATH, CSS etc. here.
  http://www.w3.org/

- Xalan. The transformation engine used in WXT.
  http://xml.apache.org/xalan-j/

- FOP. Utility to tranform .fo-files to a lot of other formats.
  http://xml.apache.org/fop/

- DocBook. A definition of XML-documents, mainly for technical documentation. A number of transformation utilities are available for this format.
  http://www.docbook.org/ or http://www.oasis-open.org/home/index.php

- SiteLite. A textbased sitebuilder written in VC++. Fast and insensible to bad XML/HTML-syntax, but requires eksplisitt markup. SiteLite is not maintained.
  http://www.ia.hiof.no/~borres/sitelite/ver30/

- Ant. Ant works well together with WXT. You may call ANT-scripts from WXT and you may start WXT from ANT.
  http://ant.apache.org/

- PrinceXML. It turns out to be very usefull to use WXT to build pages and pagegroups, and convert to PDF with PrinceXML.
  http://www.princexml.com/

- The database module in WXT was originally written by three students at Østfold College: Fredrik Helgesen, Bjørn O Samdal, Kristoffer Mysen. Redesigned and rewritten in sept. 2005.

Bygget med WXT , Jan 11, 2009