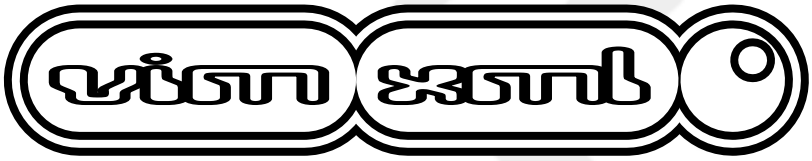
Vim as XML Editor



Tobias Reif
email: tobiasreif pinkjuice com

Vim as XML Editor

Tobias Reif

Revision History

Revision 0.10.7	2005-05-19
last change	
Revision 0.1	2002-12-15
first draft	

Draft

Table of Contents

List of Examples

Welcome :)

If Vim is your main text editor, and if you do a lot of XML editing, then this howto might help you to make that even more fun.

The latest print version is at http://www.pinkjuice.com/howto/vimxml/print/.

The online version is at http://www.pinkjuice.com/howto/vimxml/.

I appreciate any feedback, error reports are especially welcome. My email address is `tobias-reif pinkjuice com`.

You can rate this howto at www.vim.org [url 1].

Chapter 1. Intro

Vim is a great editor with many features, and it's extensible. So by learning a subset of the features, customizing it by setting preferences, and by extending it with plugins and scripts, each user builds his own editor. Existing extensions are available from http://www.vim.org/ (alternative URL: http://vim.sourceforge.net/).

Vim also is very fast and is available for many platforms. I use Vim for all my editing (XHTML, Ruby, XSLT, CSS, DBX, SVG, etc.) which means that I don't have to learn a new editor for each language. On my Vim page [url 2] there's some general info.

Vim is a text editor, and even when extended with some XML specific functionality, it is not a full-blown XML editor: For example, there's no tags- or WYSIWYG view, and no real structure/tree view. I don't really miss entry help (context-sensitive suggestions for completion), but this and other schema-driven (DTD, WXS, RNG, etc) functionality found in editors like nXML [url 3] could be implemented using Vim's Python interface [url 4] plus libxml's Python bindings [url 5], for example.

But since XML editing most often also involves a lot of general text editing tasks, and Vim offers such a vast range of functionality for those, Vim can be set up to be a surprisingly useful XML editor, especially for people who like reading the documentation and schemas of the specific XML language.

This is not really a howto. It's more a collection of tips, examples, and ideas than a complete instructional guide intended to be read from cover to back. I will simply show my current setup; there are many different variations possible. You could for example browse the sections, then pick the things which seem useful to you. There's a Vim XML Wiki page [url 6] on the SVG Wiki, and on vim.org you can search for XML scripts [url 7] and XML tips [url 8]. Please add your tips, scripts, and links.

Vim is Free Software and Charityware. If you like and use it, please consider making a donation [url 9] to ICCF Holland [url 10]. Also see `:help uganda` [url 11].

Most of the things described should work on Linux and other Unix-like operating systems such as BSD, and might also work on Windows. If you're using an emulation layer things might complicate. For example if you're using Cygwin you'll probably have to figure out some mix of the Linux and Windows setup instructions.

Note

Some of the Windows-specific stuff might be outdated since I don't have Windows anymore.

Many of the tools and techniques described in this howto can also be used without Vim, for example from the shell, from scripts, and from other editors.

There are some screenshots at http://www.pinkjuice.com/howto/vimxml/pics/screens/.

Chapter 2. About

... this howto.

Prerequisites

Most importantly, you should be familiar with Vim. If that's not the case, check out the next section and come back in a few days.

Learning Vim

Teaching Vim is outside of the scope of this howto. Here are some resources, ranging from basic to advanced:

- First you should make sure that you have the latest version of Vim. Vim is available for many OSs, check http://www.vim.org/download.php. I recommend to use GVim, the GUI version of Vim, unless you're in a remote shell session.
- Vim comes with a tutorial. After firing up Vim, enter `:help tutor`. One way to start with it is to read it into the buffer

  ```
  :read $VIMRUNTIME/tutor/tutor
  ```

  optionally delete the first line via `g g d d`, then write it to some file

  ```
  :write del/vim_tutorial.txt
  ```

- Vim comes with lots of documentation which you can consult while working with Vim. Simply enter `:help`.
- More documentation including the FAQ is listed on http://www.vim.org/docs.php.
- There's a book about Vim titled “Vi IMproved - Vim” (New Riders Publishing, Author: Steve Oualline, ISBN: 0735710015). You can buy it through the book page on iccf-holland.org [url 12]. A list of errata is listed on the publisher's page (search http://www.newriders.com/ for “vim”) and the creator of Vim Bram Moolenaar also maintains a list of errata [url 13].

Conventions

All instructions are just examples, TMTOWTDI. For example: To move the cursor upwards I use `[up]`, others use `k`.

Notation

Key sequences to be entered in Vim are written like this: `2 G [ctrl-v] [down] x`. All commands are to be entered in normal mode. Insert mode commands start with an `i`, command line ones start with a `:`. Longer commands that are to be entered in Vim's command line are

written like this:

```
:'<,'>s/<span\_s\+class="bold">\(\_.\{-}\)<\/span>/<em>\1<\/em>/g
```

In key sequences like this one `[mapleader] x i < c h a p t e r > > [up]` each group of non-space characters stands for one key press. If a space is to be entered it is listed as `[space]`.

Vim commands and option names most often have a full and an abbreviated version. When using Vim I use the short versions, eg `:h` for `:help`, `:w` for `:write`, `:q` for `:quit`, etc. In the howto I sometimes use the full versions because they explain themselves better.

Mappings in Vi can be specified to start with <Leader>, which is a variable that can be set in the `vimrc`; see `:help mapleader` [url 14]. If you did set your mapleader to some character other than the default (\), use this character instead wherever `[mapleader]` is written.

I don't use the notation used in Vim scripts because I think it's not very descriptive. If you want to use some of the commands in your Vim scripts, you must translate the function keys, eg `[enter]` to <CR> and `[backspace]` to <BS>.

Code that's to be entered in the shell of the OS most often is listed like this:

```
$ cd foo
```

The prompt character $ may indicate that the command is Linux-specific, and a > before a command may mean that the command is Windows-specific (to be entered in the “MS-DOS Prompt”). If there is no prompt character the command should work on at least both OSs.

Terminology

“Linux” most often stands for Unix-like systems such as GNU/Linux, BSD, etc.

Installation on Linux

Vim

The Vim download page [url 15] contains instructions and offers various formats.

If your distro came with sample files containing Vim settings you might want to disable them, eg by appending `.bak` to their name or by moving them to some backup directory. `:version` lists these files.

If you want additional features you might have to compile Vim yourself, check the `README` and `INSTALL` files. Here's a shortened version of how I installed it (after having resolved some dependencies such as `termcap`, `ncurses-devel`, `XFree86-devel`, `pkgconfig`).

```
$ wget -q \
> ftp://ftp.de.vim.org/unix/vim-6.3.tar.bz2
$ tar -xjf vim-6.3.tar.bz2
$ cd vim63/
$ ./configure \
> --prefix=$HOME/bulk/run/vim/6_3 \
> --with-features=big \
> --enable-gui=gtk2 \
> --with-compiledby='tobiasreif pinkjuice com' \
> --enable-rubyinterp \
```

```
> 2>&1 | tee > config.log
$ make 2>&1 | tee > make.log
$ make test 2>&1 | tee > test.log
$ make install 2>&1 | tee > install.log
$ ed
a
#!/usr/bin/env sh
${HOME}/bulk/run/vim/6_3/bin/vim "$@"
.
w /home/tobi/data/commands/vim
56
q
$ ed
a
#!/usr/bin/env sh
${HOME}/bulk/run/vim/6_3/bin/gvim "$@"
.
w /home/tobi/data/commands/gvim
57
q
$ chmod 700 ~/data/commands/vim
$ chmod 700 ~/data/commands/gvim
$ vim --version | head -n 3
VIM - Vi IMproved 6.3 (2004 June 7, compiled Jun 10 2004 15:19:15)
Compiled by tobiasreif pinkjuice com
Big version with GTK2 GUI.  Features included (+) or not (-):
$ ed ~/.bashrc
256
/EDITOR/
export EDITOR='/usr/bin/vim'
s/'.\+$/'\/home\/tobi\/data\/commands\/vim'/p
export EDITOR='/home/tobi/data/commands/vim'
wq
272
$ rpm -v --test --erase gvim
$ rpm -v --test --erase kvim
$ su root
Password:
# rpm -v --erase gvim
# rpm -v --erase kvim
# rpm -v --test --erase vim
# rpm -v --erase vim
# SuSEconfig
# exit
$ ed ~/.bashrc
272
a
MANPATH=$HOME/bulk/run/vim/6_3/man:$MANPATH
export MANPATH

.
wq
332
$ source ~/.bashrc
$ man vim
Reformatting vim(1), please wait...
```

Then I made **vim** available to root:

```
$ su -
Password:
# vim
-bash: vim: command not found
# ed .bashrc
```

```
16
a

alias vim='/home/tobi/bulk/run/vim/6_3/bin/vim'

.
wq
66
# vim
-bash: vim: command not found
# source .bashrc
# vim
# exit
logout
$
```

TMTOWTDI.

Tools

There are many different ways to install software on Linux. For some tools I describe how I installed them, but those are just examples; you'll most likely have different preferences and a different environment. I guess one typical strategy would be:

1. Ask your package manager, try it on the command line: Is the tool already installed? Is it up to date?
2. Otherwise check if there's a sufficiently recent package (eg `.deb` or `.rpm`) supplied by your distro project or vendor...
3. ... or by the project itself ...
4. ... or by a third party, if you trust it. In this and the former case it makes sense to check if there's a package specifically for your distro and -version.
5. Otherwise you could get the source from the project web site then compile and install via some variation of the common `./configure;make;make install` command sequence . If you don't want to risk conflicts caused by systemwide installation you can install the tool as user somewhere under your home directory, by omitting `make install` or by overriding the default installation directory.
6. Or the project might provide a binary for your platform.

Sources

Here are the DBX sources [url 16] and the pics [url 17].

Colophon

Note

This howto deals with XML editing thus does not deal with setting up a DocBook publishing environment. DBX documents and schemas are used as examples representing XML languages in general. Explaining how to get (X)HTML or PDF from DBX is outside the scope of this howto. Getting started is simple, but typically there's a lot more to it. For example if you use the docbook.sf.net XSLTs [url 18] check out Bob Stayton's DocBook XSL: The Complete Guide [url 19].

DBX to XHTML

The online version is generated by a set of XSLTs I have written (they are not a customization layer for an existing package). The files are listed for the curious; they are not published as supported releases. You're free to use the XSLTs, but you're on your own :). I wrote them to get the kind of XHTML I like. Ideally it should be

purely structural
: no presentational markup

used appropriately
: regarding semantics and structure (no tables except for tabular data, etc)

valid XHTML 1.0 Strict
: not so difficult, since I only use a subset of DocBook

accessible
: implementing many but not all WAI guidelines (WCAG 1.0)

Please drop me a line if you experience any problems.

Here are the XSLT files used to generate the online version of this howto [url 20]. They work well for me, but they are very incomplete, not general or flexible enough, and quirky to set up and run, so I don't recommend to use them.

The XSLTs are under the GPL. A rewrite of the DocBook to XHTML XSLT set will be at http://www.pinkjuice.com/joocs/.

Note

If you're looking for XSLTs that you can use to transform any DocBook document to various formats, you might want to check out the docbook.sf.net XSLTs. They are very complete and have various customization hooks. The project provides packaged releases [url 21], and most importantly, it provides support [url 22].

Joocs [url 23] might become an alternative to existing packages in the future, but it will be a while before Joocs supports all of DBX. It only supports XHTML output and will only be interesting for a subset of scenarios.

DBX to XSLFO to PDF

I use the docbook.sf.net XSLTs to get XSLFO from DBX. Here is the customization layer [url 24] and here are the XSLFO and PDF files [url 25]. FOP was used to get PDF from the XSLFO.

Chapter 3. Setup

Again, this is just one way of setting things up. TMTOWTDI ;) Your local environment is likely to be different, as well as your preferences, and the tools evolve. I'll give a short overview of what I use; please refer to the documentation of the respective tool for instructions on how to best set things up.

If you experience problems with any of the tools featured below please file bug reports directly to the respective project.

General

Linux

Here's my `.bashrc`:

Example 3.1. `~/.bashrc`

```
set -o vi
export EDITOR='/usr/bin/vim'

PATH="${HOME}/data/commands:${PATH}"
export PATH

PS1='\u \w \$ '

XML_CATALOG_FILES="${HOME}/data/conf/xml/catalog /etc/xml/catalog"
export XML_CATALOG_FILES
```

Often it helps to do `source .bashrc` after having modified `.bashrc` and after having created new command scripts.

Windows

When creating batch files on Windows (file name suffix `.bat`) don't forget to make sure that the file format is set correctly. You can check it with `:set fileformat?` and set it with `:set fileformat=dos`.

`vimrc`

The `vimrc` file is the place where you set your preferences. Here's mine:

Example 3.2. `vimrc`

```
" Windows:  $VIM/_vimrc (original)
" *n*x   :  ~/.vimrc
" www.pinkjuice.com/vim/vimrc.txt
```

```
" for more info check
" www.pinkjuice.com/vim/
" regarding XML related stuff check
" www.pinkjuice.com/howto/vimxml/
" email: tobiasreif pinkjuice com

" The following works for me with Vim 6.2 on Windows
" (most stuff also works on Linux),
" but I don't recommend to blindly copy and use it.
" (check the respective documentation)

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
" general

set nocompatible

highlight Normal guifg=Black guibg=#ffefd5

set formatoptions=t
set textwidth=70
set encoding=utf-8
set termencoding=latin1
set fileformat=unix
"set guifont=courier_new:h10
set guifont=Courier\ New:h10,Courier,Lucida\ Console,Letter\ Gothic,
 \Arial\ Alternative,Bitstream\ Vera\ Sans\ Mono,OCR\ A\ Extended
set nowrap
set shiftwidth=2
set visualbell
set noerrorbells
set number
set autoindent
set ruler
set expandtab
set whichwrap=<,>,h,l
set guioptions=bgmrL
set backspace=2
set history=50
set backup
set wildmenu
set nrformats=
set foldlevelstart=99
if has("unix")
  set shcf=-ic
endif

let mapleader = ","
let $ADDED = '~/.vim/added/'
if has("win32")
  let $ADDED = $VIM.'/added/'
endif

map <Leader>cd :exe 'cd ' . expand ("%:p:h")<CR>
nmap <F1> :w<CR>
imap <F1> <ESC>:w<CR>a
map <F8> gg"+yG

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
" Installed

" www.vim.org/scripts/script.php?script_id=301
" $ADDED/xml.vim

" www.vim.org/scripts/script.php?script_id=39
```

```
" copied macros/matchit.vim to plugin/

""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
" XML

map <Leader>x :set filetype=xml<CR>
  \:source $VIMRUNTIME/syntax/xml.vim<CR>
  \:set foldmethod=syntax<CR>
  \:source $VIMRUNTIME/syntax/syntax.vim<CR>
  \:colors peachpuff<CR>
  \:source $ADDED/xml.vim<CR>
  \:iunmap <buffer> <Leader>.<CR>
  \:iunmap <buffer> <Leader>><CR>
  \:inoremap \> ><CR>
  \:echo "XML mode is on"<CR>
  " no imaps for <Leader>
  "\:inoremap \. ><CR>

" catalog should be set up
nmap <Leader>l <Leader>cd:%w !xmllint --valid --noout -<CR>
nmap <Leader>r <Leader>cd:%w !rxp -V -N -s -x<CR>
nmap <Leader>d4 :%w !xmllint --dtdvalid
 \ "http://www.oasis-open.org/docbook/xml/4.2/docbookx.dtd"
 \ --noout -<CR>

vmap <Leader>px !xmllint --format -<CR>
nmap <Leader>px !!xmllint --format -<CR>
nmap <Leader>pxa :%!xmllint --format -<CR>

nmap <Leader>i :%!xsltlint<CR>

" todo:
" check
" http://mugca.its.monash.edu.au/~djkea2/vim/compiler/xmllint.vim

""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
" Ruby

" check
" www.rubygarden.org/ruby?VimExtensions
```

The latest version is online at http://www.pinkjuice.com/vim/vimrc.txt.

For example there is

```
let mapleader = ","
```

and

```
nmap <Leader>l <Leader>cd:%w !xmllint --valid --noout -<CR>
```

which means that whenever I want xmllint to validate the buffer I do `, l`.

---

matchit.vim

Jump around the buffer with %, eg between opening and closing angle brackets of XML tags and between opening and closing tags when XML syntax recognition is turned on.

home

http://www.vim.org/scripts/script.php?script_id=39

For installation instructions enter `:help add-local-help` [url 26]. If I remember correctly I simply copied `macros/matchit.vim` to `plugin/` (`.vim/plugin/` on Linux).

---

xmledit

Devin Weaver's xmledit provides some editing features such as closing and deleting tags, and wrapping strings in tag pairs.

Home
http://www.vim.org/scripts/script.php?script_id=301

Manual
http://tritarget.com/vim/xml-plugin/doc/xml-plugin.txt (may be outdated, check the above home page)

I don't want it to be turned on automatically, so I created a directory `added/`. In my `vimrc` I have

```
let $ADDED = '~/.vim/added/'
if has("win32")
  let $ADDED = $VIM.'/added/'
endif
```

and

```
map <Leader>x :set filetype=xml<CR>
  \:source $VIMRUNTIME/syntax/xml.vim<CR>
  \:set foldmethod=syntax<CR>
  \:source $VIMRUNTIME/syntax/syntax.vim<CR>
  \:colors peachpuff<CR>
  \:source $ADDED/xml.vim<CR>
```

so I can turn on XML editing mode whenever I want. I don't want it to be turned on automatically because
- when I open a large file it would slow down editing
- sometimes I have files with mixed syntax (eg XML snippets in a text file)
- I prefer to be independent from filename suffixes

When the buffer grows large and changes become slow, try `:syn off`.

Catalogs

When validating an XML document which references an online DTD in it's doctype declaration,

```
<!DOCTYPE html
  PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

this DTD needs to get loaded. Normally it gets downloaded from the web, which can take a while with large schemas such as DocBook. Changing the URL to a local path each time is tedious, especially if there are many documents.

Convenient and fast offline validation can be achieved though setting up a catalog system.

When using **xmllint**, this means nothing more than creating a simple catalog file, and making its path available to **xmllint**.

A catalog lists the FPIs of document types such as SVG, XHTML, and DBX, and says where to find the corresponding DTDs locally. In the OASIS XML Catalog language, a simple entry looks like this:

```
<public
  publicId="-//W3C//DTD XHTML 1.0 Strict//EN"
  uri="xhtml/1_0/strict/dtd/xhtml1-strict.dtd"/>
```

Any XML tool which supports OASIS XML Catalogs [url 27] (OASIS Catalogs specification [url 28]) can now see if there is a local copy of the corresponding DTD available. It simply grabs the FPI from the document, and looks it up in the catalog.

OASIS catalogs can also be used to map system identifiers to local paths:

```
<system
  systemId="http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"
  uri="xhtml/1_0/strict/dtd/xhtml1-strict.dtd"/>
```

This is useful in cases where there is no FPI available.

The typical location and name for an OASIS XML catalog is `/etc/xml/catalog`. On Linux I don't edit `/etc/xml/catalog` because various various installers write to that file. Instead I place my catalog under my home directory so that it doesn't get modified by any automatic process. Tools like **rxp**, **xmllint**, and **xsltproc** support multiple catalogs. In my `.bashrc` I have

```
XML_CATALOG_FILES="${HOME}/data/conf/xml/catalog /etc/xml/catalog"
export XML_CATALOG_FILES
```

and my catalog looks like this:

Example 3.3. `~/data/conf/xml/catalog`

```
<?xml version="1.0"?>
<!DOCTYPE catalog
PUBLIC "-//OASIS//DTD Entity Resolution XML Catalog V1.0//EN"
"http://www.oasis-open.org/committees/entity/release/1.0/catalog.dtd">
<catalog
xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog">

<group
prefer="public"
xml:base="file:///home/tobi/bulk/xml/schemas/">

<public
publicId="-//OASIS//DTD Entity Resolution XML Catalog V1.0//EN"
uri="oasis_catalogs/1_0/dtd/catalog.dtd"/>
<system
systemId=
"http://www.oasis-open.org/committees/entity/release/1.0/catalog.dtd"
uri="oasis_catalogs/1_0/dtd/catalog.dtd"/>

<public
publicId="-//W3C//DTD XHTML 1.0 Strict//EN"
uri="xhtml/1_0/strict/dtd/xhtml1-strict.dtd"/>
<system
systemId="http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"
uri="xhtml/1_0/strict/dtd/xhtml1-strict.dtd"/>

<public
```

```
publicId="-//OASIS//DTD DocBook XML V4.2//EN"
uri="docbook/4_2/dtd/docbookx.dtd"/>
<system
systemId="http://www.oasis-open.org/docbook/xml/4.2/docbookx.dtd"
uri="docbook/4_2/dtd/docbookx.dtd"/>

<public
publicId="-//W3C//DTD SVG 1.0//EN"
uri="svg/1_0/dtd/svg10.dtd"/>
<system
systemId="http://www.w3.org/TR/SVG10/DTD/svg10.dtd"
uri="svg/1_0/dtd/svg10.dtd"/>
<system
systemId="http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd"
uri="svg/1_0/dtd/svg10.dtd"/>

<public
publicId="-//W3C//DTD SVG 1.1//EN"
uri="svg/1_1/dtd/svg11-flat.dtd"/>
<system
systemId="http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd"
uri="svg/1_1/dtd/svg11-flat.dtd"/>
<system
systemId="http://www.w3.org/Graphics/SVG/1.1/DTD/svg11-flat.dtd"
uri="svg/1_1/dtd/svg11-flat.dtd"/>

</group>

<public
publicId="-//Pinkjuice//DTD link-tree 0.1//EN"
uri="../../../public_html/pj/svg/links/link-tree_0_1.dtd"/>
<system
systemId="http://www.pinkjuice.com/svg/links/link-tree_0_1.dtd"
uri="../../../public_html/pj/svg/links/link-tree_0_1.dtd"/>

<system
systemId="http://www.example.org/tobias/reif/contacts.dtd"
uri="../../xml/schemas/contacts/dtd/contacts.dtd"/>

</catalog>
```

My catalog is not only used by **xmllint**, but also by the resolvers of some other tools such as validators and XSLT processors.

---

xmllint

xmllint is part of libxml, the XML C library for GNOME. It's lightning fast and can be used for validation and pretty printing.

home
: http://xmlsoft.org/xmldtd.html

man
: http://www.xmlsoft.org/xmllint.html

On Linux

Most of the following should work analogously on other Unix-like OSs such as BSD.

If you are on Linux **xmllint** might already be installed on your system. To make sure you have the latest version check

```
$ xmllint --version
```

and http://xmlsoft.org/downloads.html. In order to not risk any conflicts I installed the latest versions under my home directory using the following shell script:

Example 3.4. `install_libxml`

```
#!/bin/bash -x
# or
#!/usr/bin/env bash

# todo:
#
# refactor, wrap a lot of redundant code in functions
#
# pass --disable-shared to configure?
# or --enable-static, or both? do
# tobi ~/del $ grep AttributeError \
# libxml_cvs_installation.log | wc -l

# this is just an example you could use as basis for your script
# (do not run it without having revised and adjusted it)

# usage: (be online)
# (remove existing identical/redundant commands and logs,
# and empty compilation, (download,) and run directories)
# if not CVS then set versions below
# [as user do:]
# $ cd del
# $ ~/data/run/install_libxml --cvs \
# 2>&1 | tee libxml_cvs_installation.log
# (or &> libxml_cvs_installation.log ?)
# (then run without [...]cvs:
# $ ~/data/run/install_libxml 2>&1 | tee libxml_installation.log
# )
# optionally monitor with lines like
# $ tail -f \
# ~/del/compile_libxml/libxml2-2.6.9/my_xml_make_errors.log

# if the installation succeeded,
# set latest stable versions as defaults
# cd
# cp -i data/commands/xmllint_2.6.9 data/commands/xmllint
# cp -i data/commands/xsltproc_1.1.6 data/commands/xsltproc
# [not necessarily required:]
# source .bashrc
# xmllint --version
# xsltproc --version

# this script is install_libxml
# based on
# http://xmlstar.sourceforge.net/doc/run-xmlstarlet-build
# (any errors are mine :)
# Various lines were contributed by William M. Brack; thanks!

# zlib-devel should be available among other things,
# see libxml2-[version]/INSTALL
```

```
my_home=/home/tobi
if [ $HOME != $my_home ]; then
  exit
fi
if [ `whoami` != 'tobi' ]; then
  exit
fi

cd ${HOME}/del

case $# in
  0) ;;
  *)
  for a do
    case $a in
      -c|--cvs)
      cvs=true;;
      *)
      echo "unexpected arg $a"
      exit;;
    esac
  done
esac

# set:
# check for latest versions at ftp://xmlsoft.org/
if [ $cvs ]; then
  ver_libxml='cvs'
  ver_libxslt='cvs'
else
  ver_libxml=2.6.9
  ver_libxslt=1.1.6
fi

uname -a
tools="
  gcc
  make
  automake
  install
"
for tool in $tools ; do
  $tool --version
done
/lib/ld-linux.so.2 /lib/libc.so.6 | head -1

# does it make any sense to scan the files?
av_command="antivir -rs -z"

compile=${HOME}/del/compile_libxml

run_libxml_top=${HOME}/bulk/run/libxml
run_libxml=${run_libxml_top}/${ver_libxml}
run_libxslt_top=${HOME}/bulk/run/libxslt
run_libxslt=${run_libxslt_top}/${ver_libxslt}

if [ ! -d $compile ]; then
  mkdir $compile
fi

if [ ! -d $compile/download ]; then
  mkdir ${compile}/download
fi
```

```
if [ -d $run_libxml ]; then
  echo ${run_libxml}' exists, exiting'
  exit
else
  if [ ! -d $run_libxml_top ]; then
    mkdir $run_libxml_top
  fi
  if [ ! -d $run_libxml ]; then
    mkdir $run_libxml
  fi
fi

if [ -d $run_libxslt ]; then
  echo ${run_libxslt}' exists, exiting'
  exit
else
  if [ ! -d $run_libxslt_top ]; then
    mkdir $run_libxslt_top
  fi
  if [ ! -d $run_libxslt ]; then
    mkdir $run_libxslt
  fi
fi

cd $compile

if [ $cvs ]; then
  url_libxml='ftp://xmlsoft.org/libxml2-cvs-snapshot.tar.gz'
  url_libxslt='ftp://xmlsoft.org/libxslt-cvs-snapshot.tar.gz'
else
  url_libxml="ftp://xmlsoft.org/libxml2-${ver_libxml}.tar.gz"
  url_libxslt="ftp://xmlsoft.org/libxslt-${ver_libxslt}.tar.gz"
fi
file_libxml=`basename ${url_libxml}`
file_libxslt=`basename ${url_libxslt}`

if [ ! -f download/$file_libxml ]; then
  cd download
  wget $url_libxml
  # should: exit if download didn't succeed
  if [ ! -e  $file_libxml ]; then
    exit
  fi
  $av_command $file_libxml
  if [ $? != 0 ]; then
    exit
  fi
  cd ../
fi

if [ ! -f download/$file_libxslt ]; then
  cd download
  wget $url_libxslt
  # should: exit if download didn't succeed
  if [ ! -e  $file_libxslt ]; then
    exit
  fi
  $av_command $file_libxslt
  if [ $? != 0 ]; then
    exit
  fi
  cd ../
fi
```

```
libxml2_cvs_dir_name=libxml2-cvs
libxslt_cvs_dir_name=libxslt-cvs

if [ $cvs ]; then

  mkdir $libxml2_cvs_dir_name && cd $libxml2_cvs_dir_name
  tar -xzf ../download/${file_libxml}
  cd ../
  # toplevel_xml_dir="${compile}/${libxml2_cvs_dir_name}/*/"
  toplevel_xml_dir_parent="${compile}/${libxml2_cvs_dir_name}"
  toplevel_xml_dir="${toplevel_xml_dir_parent}/$(ls $toplevel_xml_dir_parent)"

  mkdir $libxslt_cvs_dir_name && cd $libxslt_cvs_dir_name
  tar -xzf ../download/${file_libxslt}
  cd ../
  # toplevel_xslt_dir="${compile}/${libxslt_cvs_dir_name}/*/"
  toplevel_xslt_dir_parent="${compile}/${libxslt_cvs_dir_name}"
  toplevel_xslt_dir="${toplevel_xslt_dir_parent}/$(ls $toplevel_xslt_dir_parent)"

else

  toplevel_xml_dir="${compile}/libxml2-${ver_libxml}"
  toplevel_xslt_dir="${compile}/libxslt-${ver_libxslt}"
  tar -xzf download/${file_libxml}
  tar -xzf download/${file_libxslt}

fi

if [ $cvs ]; then
  # configure='autogen.sh'
  configure='configure'
  # --enable-static --disable-shared ?
else
  configure='configure'
  # --enable-static --disable-shared ?
fi

cd $toplevel_xml_dir
if [ $cvs ] && [ -f Makefile ]; then
  make distclean
fi
./${configure} --prefix=${run_libxml} \
  2>&1 | tee my_xml_config.log
# make CFLAGS="-g -O2 -static" \
make \
  2>my_xml_make_errors.log | tee my_xml_make.log
make tests 2>&1 | tee my_xml_tests.log
make install

cd $toplevel_xslt_dir
if [ $cvs ] && [ -f Makefile ]; then
  make distclean
fi
./${configure} --prefix=${run_libxslt} \
  --with-libxml-src=${toplevel_xml_dir} \
  2>&1 | tee my_xslt_config.log
# make CFLAGS="-g -O2 -static" \
make \
  2>my_xslt_make_errors.log | tee my_xslt_make.log
make tests 2>&1 | tee my_xslt_tests.log
make install

command_xmllint=${HOME}/data/commands/xmllint_${ver_libxml}
```

```
command_xsltproc=${HOME}/data/commands/xsltproc_${ver_libxslt}

if [ ! -f $command_xmllint ]; then
  cat > $command_xmllint << EOF
#!/usr/bin/env sh
# may get overwritten
${run_libxml}/bin/xmllint "\$@"
EOF
  chmod 700 $command_xmllint
  $command_xmllint --version
fi

if [ ! -f $command_xsltproc ]; then
  cat > $command_xsltproc << EOF
#!/usr/bin/env sh
# may get overwritten
${run_libxslt}/bin/xsltproc "\$@"
EOF
  chmod 700 $command_xsltproc
  $command_xsltproc --version
fi

# email my_*.log files?
# below tgz plus
# del/libxml_cvs_installation.log
# and this script

cd $compile
tar -czf tobilogs${ver_libxml}.tgz \
   -C $toplevel_xml_dir \
     my_xml_config.log my_xml_make.log \
     my_xml_make_errors.log my_xml_tests.log \
   -C $toplevel_xslt_dir \
     my_xslt_config.log my_xslt_make.log \
     my_xslt_make_errors.log my_xslt_tests.log
```

It's always a good idea to store all schemas that you use locally.

If there is no catalog set up and you need to get started quickly you can validate Vim buffers with commands like this:

```
:%w !xmllint --noout --dtdvalid /path/to/xhtml1-strict.dtd -
```

As long as most documents specify their language via doctype declarations it makes sense to set up an OASIS catalog for validation. Most often I put my tool call scripts in `~/data/commands/` but you can use any directory.

```
$ echo $PATH
```

might already include `~/bin/` so if you don't want to add a new directory to the system path you can put your calls into `/your/home/bin/`.

You can set `XML_CATALOG_FILES` in the rc file of your shell or in a shell script calling xmllint. Here's an example:

Example 3.5. `~/bin/xmlval`

```
#!/usr/bin/env bash
XML_CATALOG_FILES=/path/to/catalog
```

```
export XML_CATALOG_FILES
xmllint --valid --noout "$@"
```

```
$ chmod 700 ~/bin/xmlval
```

If you are in a bash shell,

```
$ xmlval --version
```

should return the version of xmllint.

If you get “command not found” when trying to call **xmlval** you need to ask the shell to search the path again. If

```
$ echo $SHELL
```

returns `csh` or `tcsh` you can try

```
$ set path=($path)
```

and test it with

```
$ xmlval --version
```

Now you should be able to use

```
:%w !xmlval -
```

to validate documents which have doctype declarations.

Another way is to use **xmllint** directly, and set the environment variable `XML_CATALOG_FILES` in the rc file of the shell.

If you use csh or tcsh do

```
$ vim .cshrc
```

then append

```
setenv XML_CATALOG_FILES "/path/to/catalog"
```

and do

```
$ source .cshrc
```

Now

```
:%w !xmllint --valid --noout -
```

should work fast and offline if there's a doctype declaration and if the catalog contains a link to a corresponding schema.

If you use bash you can append these lines to your `.bashrc`:

```
XML_CATALOG_FILES="/path/to/catalog"
export XML_CATALOG_FILES
```

then do

```
$ source .bashrc
```

and test it with

```
$ echo $XML_CATALOG_FILES
```

To validate documents which have a doctype declaration do

```
:%w !xmllint --valid --noout -
```

in Vim.

On Windows

Follow the installation instructions on http://www.zlatkovic.com/libxml.en.html, or try the following:

After having downloaded and unzipped `libxml2-`*`version`*`.win32.zip` from ftp://ftp.zlatkovic.com/pub/libxml/ (otherwise check http://xmlsoft.org/sources/win32/), create the following batch file calling xmllint and put it in a directory which you added to the system path; mine is called `calls\` and I use it for most tools.

Example 3.6. `xmllint.bat`

```
@echo off
set XML_CATALOG_FILES=/path/to/catalog
"\path\to\libxml2-version.win32\bin\xmllint" %1 %2 %3 %4 %5
```

On Win9*-systems you might encounter an error message related to insufficient memory. In this case try increasing the available memory: rename `xmllint.bat` to `xmllint_raw.bat` and create the following batch file:

Example 3.7. `xmllint.bat` (Windows 95/98/ME)

```
@echo off
command.com /e:3000 /c xmllint_raw %1 %2 %3 %4 %5
```

On NT-based Windows versions such as Windows 2000 the prefix to explicitly pass the command to the shell would be

```
cmd.exe /c
```

but it shouldn't be necessary.

Alternatively you could add `libxml2-`*`version`*`.win32\bin\` to the system path but you'd have to restart Windows, and if you do this for all the directories of the tools the path can get too long. I don't recommend putting the files into Windows directories such as `C:\WINDOWS\` since this can cause conflicts.

To see if it works, try some commands which call **xmllint** from Vim's command line and ask it to validate some XML. Below are some tiny skeletons you can use. SVG:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE svg
  PUBLIC "-//W3C//DTD SVG 1.0//EN"
  "http://www.w3.org/TR/SVG/DTD/svg10.dtd">
<svg>
</svg>
```

On `pinkjuice.com` there is a more complete SVG skeleton [url 29]. XHTML:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html
  PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html
  xmlns="http://www.w3.org/1999/xhtml"
  xml:lang="en" lang="en">
  <head>
    <title></title>
  </head>
  <body>
    <h1></h1>
    <div>
    </div>
  </body>
</html>
```

The following command for example passes the path of the file to xmllint:

```
:!xmllint --valid --noout %
```

The `--valid` option tells **xmllint** to validate the doc, and the other arguments should be self-explanatory.

Find more command examples in chapter Tasks.

RXP

You can never have enough validators `:)` The error messages vary in usefulness, and some errors aren't caught when using only one validator. RXP is developed by Richard Tobin who is one of the editors of Namespaces in XML 1.1 [url 30].

home
: http://www.cogsci.ed.ac.uk/~richard/rxp.html

download
: ftp://ftp.cogsci.ed.ac.uk/pub/richard/

man
: ftp://ftp.cogsci.ed.ac.uk/pub/richard/rxp.txt

On Linux

Following is how I installed RXP on Linux. The output of **make** and **wget** is omitted.

```
$ cd del
$ mkdir compile_rxp && cd compile_rxp
$ wget ftp://ftp.cogsci.ed.ac.uk/pub/richard/rxp-1.4.0pre10.tar.gz
$ tar xzf rxp-1.4.0pre10.tar.gz
$ cd rxp-1.4.0pre10
$ make
$ mkdir ~/bulk/run/rxp
$ mkdir ~/bulk/run/rxp/1_4_0_pre_10
$ ls | egrep "^[^\.]+$" | \
```

```
> xargs cp --target-directory=$HOME/bulk/run/rxp/1_4_0_pre_10
$ ed
a
#!/usr/bin/env sh
${HOME}/bulk/run/rxp/1_4_0_pre_10/rxp "$@"
.
w /home/tobi/data/commands/rxp
61
q
$ chmod 700 ~/data/commands/rxp
$ source ~/.bashrc
$ rxp -v
RXP 1.4.0pre10 Copyright Richard Tobin,
LTG, HCRC, University of Edinburgh
<foo   />
Input encoding UTF-8, output encoding UTF-8
<foo/>
$
```

If you follow the steps, don't type the > after the line ending with `\`, and type `[ctrl-d` to end input to the `rxp -v` command.

On Windows

On NT-based Windows versions the following batch file should work:

Example 3.8. `rxp.bat`

```
@echo off
set XML_CATALOG_FILES=file:///drive:/path/to/catalog
\path\to\rxpversion.exe %1 %2 %3 %4 %5 %6 %7 %8 %9
```

Put it in a directory which is on the system's path.

On Windows ME/9* you might see an error message related to limited memory. The following batch files should solve this:

Example 3.9. `rxp_raw.bat` (Win9*)

```
@echo off
set XML_CATALOG_FILES=file:///drive:/path/to/catalog
\path\to\rxpversion.exe %1 %2 %3 %4 %5 %6 %7 %8 %9
```

Example 3.10. `rxp.bat` (Win9*)

```
@echo off
command.com /e:3000 /c rxp_raw %1 %2 %3 %4 %5 %6 %7 %8 %9
```

In order to shorten command lines you can add

Example 3.11. `rxpval.bat`

```
@echo off
rxp -V -N -s -x %1 %2 %3 %4 %5 %6 %7 %8 %9
```

draft

Chapter 4. More Setup

The tools listed in this chapter are less basic/crucial than those in the previous chapter, and are optional for many users.

Ruby

Ruby is a very nice object-oriented programming language from Japan. Some scripts in this howto are written in Ruby so I recommend to install it. Alternatively you could translate the scripts to your favourite language.

home
: http://www.ruby-lang.org/en/

books
: http://www.rubygarden.org/ruby?RubyBookList

Make sure that you have the latest version or 1.8

```
$ ruby -v
```

otherwise install it.

On Linux

The latest stable version of Ruby (1.8.1) is available from various places, some are are listed below. The first one is a redirector, the last one is the original location which should not be used if possible.

- http://www.ruby-lang.org/cgi-bin/download-1.8.1.mrb
- http://www.approximity.com/ruby/mirror/ruby-1.8.1.tar.gz
- http://rubyforge.org/download.php/262/ruby-1.8.1.tar.gz
- http://www.ruby-doc.org/downloads/ruby-1.8.1.tar.gz
- ftp://ftp.ruby-lang.org/pub/ruby/ruby-1.8.1.tar.gz

The MD5 check sum is `5d52c7d0e6a6eb6e3bc68d77e794898e`.

After having downloaded and unpacked the archive read the `README`, under “How to compile and install”. If you're on Mac OS X, check Rich Kilmer's “Building Ruby 1.8.1 on Panther” [url 31].

Here's how I installed Ruby: First I installed readline-devel (I don't know if this was necessary since readline was installed already). Then I did the following (output of some commands is omitted):

```
$ mkdir del/compile/ruby
$ cd del/compile/ruby
$ wget http://www.approximity.com/ruby/mirror/ruby-1.8.1.tar.gz
$ md5sum --check
5d52c7d0e6a6eb6e3bc68d77e794898e *ruby-1.8.1.tar.gz
ruby-1.8.1.tar.gz: OK
$ tar -xzf ruby-1.8.1.tar.gz
$ cd ruby-1.8.1/
$ mkdir /home/tobi/bulk/run/ruby
$ mkdir /home/tobi/bulk/run/ruby/1_8_1
```

```
$ autoconf
$ ./configure --prefix=/home/tobi/bulk/run/ruby/1_8_1
$ make
$ make test
test succeeded
$ make install
$ ed
a
#!/usr/bin/env sh
${HOME}/bulk/run/ruby/1_8_1/bin/ruby "$@"
.
w /home/tobi/data/commands/ruby_1.8.1
60
q
$ chmod 700 ~/data/commands/ruby_1.8.1
$ ruby_1.8.1 -v
ruby 1.8.1 (2003-12-25) [i686-linux]
$ ruby_1.8.1 test/runner.rb
$ ed
a
#!/usr/bin/env sh
${HOME}/bulk/run/ruby/1_8_1/bin/irb "$@"
.
w /home/tobi/data/commands/irb_1.8.1
59
q
$ chmod 700 ~/data/commands/irb_1.8.1
$ irb_1.8.1
irb(main):001:0> puts 6
6
=> nil
irb(main):002:0> puts 6
6
=> nil
```

With the Ruby that came with my distro, readline doesn't work; `[up]` results in `^[[A`. With the Ruby I installed IRB works (although I have to hit `[escape]` before entering `[up]`).

On Windows

Run the latest `ruby`*`version`*`.exe` from http://rubyinstaller.sf.net/, restart, then test with

```
>ruby -v
```

Cross-OS Tool Calls

Unfortunately Ruby doesn't yet fully support Windows. The following file can help making system calls more portable, all you need to do is to `require` it in your scripts.

Example 4.1. `cross_os_calls.rb`

```
#!/usr/bin/env ruby

# based on code by Park Heesob (http://www.ruby-talk.org/10006)
# (also see http://www.ruby-talk.org/9739 and
# http://www.ruby-talk.org/81128)
# please feed back improvements: tobiasreif pinkjuice com
# Before using this, please confirm that you have the latest version
# of Ruby, and that the problem still exists.
```

```
#############################################################################
# problem
# (if this type of test works for you, you don't need to
# require this file)

# puts `tidy -v`
# ruby 1.8.0 (2003-08-04) [i386-mswin32]
# =>
# cross_os_calls.rb:15:in ``': No such file or directory -
# tidy -v (Errno::ENOENT)

#############################################################################
# workaround

def windows?
  if Config::CONFIG["arch"] =~ /win/
    true
  else
    false
  end
end

require 'rbconfig'

alias oldSystem system
def system(command)
  if windows?
    require 'Win32API'
    Win32API.new("crtdll", "system", ['P'], 'L').Call(command)
  else
    oldSystem command
  end
end

alias oldBacktick `
def `(command)
  if windows?
    require 'Win32API'
    popen = Win32API.new("crtdll", "_popen", ['P','P'], 'L')
    pclose = Win32API.new("crtdll", "_pclose", ['L'], 'L')
    fread = Win32API.new("crtdll", "fread", ['P','L','L','L'], 'L')
    feof = Win32API.new("crtdll", "feof", ['L'], 'L')
    saved_stdout = $stdout.clone
    psBuffer = " " * 128
    rBuffer = ""
    f = popen.Call(command,"r")
    while feof.Call( f )==0
      l = fread.Call( psBuffer,1,128,f )
      rBuffer += psBuffer[0...l]
    end
    pclose.Call f
    $stdout.reopen(saved_stdout)
    rBuffer
  else
    oldBacktick command
  end
end

TempDir =
if ENV['TMP']
   ENV['TMP']
elsif windows?
  `echo %temp%`.strip
```

```
else
  '/tmp'
end

DirSep =
if File::ALT_SEPARATOR
  File::ALT_SEPARATOR
# work around cygwin returning '/'
elsif windows?
  '\\'
else
  File::SEPARATOR
end

# puts TempDir
# puts DirSep

#########################################################################
# tests
# ... pass in 1.6.5 and 1.8.0 (rubyinstaller.sf.net) on Windows ME

# puts `tidy -v`

# if Config::CONFIG["arch"] =~ /win/
#   temp_path_command = 'echo %temp%'
#   p `#{temp_path_command}`.strip
# end
```

Jing

The normative schemas of many XML standards will be written in RNG, and Jing is an RNG validator written by one of the main creators of RNG, James Clark.

home
: http://www.thaiopensource.com/relaxng/jing.html

man
: See `readme.html` in the toplevel dir of the package and `doc/jing.html`.

Jing doesn't (yet) support stdin so I use a simple Ruby script to fake it for now.

Make sure you have the required version of the JRE. I recommend the latest version, currently that's 1.4.

```
java -version
```

After having downloaded `jing-`*`version`*`.zip` from http://www.thaiopensource.com/download/ and having unzipped it, save the following Ruby script. On Windows add suffix `rb` to the file name so that you get `\`*`any`*`\`*`directory`*`\jing.rb`, on Linux put it into a directory which is on the system path and do

```
$ chmod 700 jing
```

Example 4.2. `jing`

```
#!/usr/bin/env ruby

# jing - faking stdin for Jing

Jing_Jar =
'/path/to/jing-version/bin/jing.jar'
$:<<'/path/to/ruby/shared/'
require 'cross_os_calls.rb'

files=ARGV.grep(/^[^-]/)
argument_string=ARGV.join(' ')
tempfile=TempDir+DirSep+'jing'+$$.to_s+Time.now.to_f.to_s
JING='java -jar '+Jing_Jar+' '

case files.length
when 1 then
  # If there's only one file arg given it's is the RNG;
  # stdin will be the XML doc to validate.
  if stdin = $stdin.read
    tf = File.new(tempfile,'w')
    tf.flock(File::LOCK_EX)
    tf.write stdin; tf.close
    command = JING+argument_string+' '+tempfile
    system command
    File.delete tempfile
  else
    puts "If you supply only one file arg (the RNG)\n"+
      'you must supply stdin (the XML).'
  end
else
  # If there are zero or more than one file args,
  # pass all args to Jing.
  command = JING+argument_string
  system command
end
```

Adjust the two paths in the script. On Windows put the following batch script into a directory which is on the system path:

Example 4.3. `jing.bat`

```
@echo off
ruby /path/to/jing.rb %1 %2 %3 %4 %5 %6 %7 %8 %9
```

To test it go to `jing-version/doc/xhtml/` and do

```
jing xhtml-strict.rng index.html
```

in the command line. Change `index.html` to be invalid, run **jing** again: You should see errors. Change it back to it's original state, validate again: There should be no output.

To validate XHTML documents you can do

```
:%w !jing /path/to/xhtml-strict.rng
```

in Vim. If the doc has a doctype declaration referencing an online DTD Jing will fetch it from the web which takes a while. You could comment it out if it's not needed for entity declarations or

attribute value defaults, or you can exclude it by sending just the root element with contents, eg

```
:6,$w !jing /path/to/xhtml-strict.rng
```

As schema you can use `jing-version/doc/xhtml/xhtml-strict.rng`.

Note

Although the name suggests otherwise it is based on XHTML 1.1 not on 1.0 Strict, thus excludes attribute `lang` etc; for details see XHTML 1.1 Appendix A [url 32].

As an ad hoc solution for validating SVG you can download the SVG 1.1 RNG [url 33] via

```
wget -q -nd -A rng -l 1 -r http://www.w3.org/Graphics/SVG/1.1/rng/
```

wget is available for many OSs including Windows, check the **wget** home page [url 34] (alternative **wget** home page [url 35]). Recently they added a zip file, check the directory or try http://www.w3.org/Graphics/SVG/1.1/rng/rng.zip. If the files still contain the following line

```
<!DOCTYPE grammar SYSTEM "../relaxng.dtd">
```

delete it from all files via the following command (substitute the apostrophes for quotes on Windows):

```
ruby -ni.bak -e 'print if not /^<!DOCTYPE/' *.rng
```

or change the path to a real system identifier (URI or canonical URL).

XMLStarlet

From the web site:

> XMLStarlet is a set of command line utilities (tools) which can be used to transform, query, validate, and edit XML documents and files using simple set of shell commands in similar way it is done for plain text files using UNIX **grep**, **sed**, **awk**, **diff**, **patch**, **join**, etc commands.

It's fast and promising.

home
: http://xmlstar.sourceforge.net/

doc
: http://xmlstar.sourceforge.net/docs.php

On Linux

Here's the script that I use to install XMLStarlet:

Example 4.4. `install_xmlstar`

```
#!/bin/bash -x
```

```
# This is just an example you could use as basis for your script.
# (do not run it without having revised and adjusted it)

# The --with-[...]-src paths must point to the libxml and libxslt
# sources.
# The sources are available after install_libxml finished, for
# example.
# Set the version numbers below.
# Be online, then do
# tobi ~/del $ ~/data/run/install_xmlstar

# this doesn't really make sense ...
av_command="antivir -rs -z"

my_home=/home/tobi

if [ ! $HOME == $my_home ]; then
  exit
fi
if [ `whoami` != 'tobi' ]; then
  exit
fi

# set these:
ver_xmlstar=0.8.1
ver_libxml=2.6.5
ver_libxslt=1.1.2

run_top=${HOME}/bulk/run/xmlstar
run=${run_top}/${ver_xmlstar}
compile=${HOME}/del/compile_libxml
command=${HOME}/data/commands/xmlstar

if [ -d $run ]; then
  echo ${run}' exists, exiting'
  exit
else
  if [ ! -d $run_top ]; then
    mkdir $run_top
  fi
  if [ ! -d $run ]; then
    mkdir $run
  fi
fi
cd $compile

#############################################################################

# based on
# http://xmlstar.sourceforge.net/doc/run-xmlstarlet-build

url_xmlstar="http://xmlstar.sourceforge.net/downloads/\
xmlstarlet-${ver_xmlstar}.tar.gz"

file_xmlstar=`basename ${url_xmlstar}`

if [ ! -f download/$file_xmlstar ]; then
  cd download
  wget $url_xmlstar
  $av_command $file_xmlstar
  # if [ $? != 0 ]; then
  if [ $? -ne 0 ]; then
    exit
```

```
  fi
  cd ../
fi

tar -xzf download/${file_xmlstar}

cd xmlstarlet-${ver_xmlstar}
./configure --prefix=${run} \
  --with-libxml-src=${compile}/libxml2-${ver_libxml} \
  --with-libxslt-src=${compile}/libxslt-${ver_libxslt}
make
make tests
make install

########################################################################

# if [ ! -f $command ]; then
  cat > $command << EOF
#!/usr/bin/env sh
# may get overwritten
${run}/bin/xml "\$@"
EOF
  chmod 700 $command
# fi

xmlstar --version
```

On Windows

Installation is very simple. After having downloaded and unzipped XMLStarlet (`xmlstarlet-`*`version`*`-win32.zip`) I added the directory containing `xml.exe` to the system path. This makes the system path longer and requires a restart, but batch files support only up to nine arguments which often is not enough when using XMLStarlet. I think that `xml` is a confusingly generic name for a command so I renamed it to `xmlstar` by renaming `xml.exe` to `xmlstar.exe`.

Try it out

Caution

Whenever you filter your data through a tool it can get corrupted. If something went wrong you can use u to undo the filtering.

XMLStarlet can be used to remove all objects matching an XPath, eg all style attributes from an XHTML document. Paste the following into Vim:

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>foo</title>
  </head>
  <body>
    <div style="text-align:center">
      <p id="foo" style="color:green" class="blammo">
        foo
      </p>
    </div>
```

```
  </body>
</html>
```

Then do

```
:%!xmlstar ed --delete //@style
```

You should get something like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>foo</title>
  </head>
  <body>
    <div>
      <p id="foo" class="blammo">
        foo
      </p>
    </div>
  </body>
</html>
```

Tidy

Sometimes I receive HTML files generated by Microsoft Word; Often they are very bloated. Tidy can make them around five times smaller, and can help with turning them into valid XHTML. The results can't be guaranteed to be really good code regarding semantics and structure, but the files become much easier to work with.

Current Home
: http://tidy.sourceforge.net/

Original Home
: http://www.w3.org/People/Raggett/tidy/

Project
: http://sourceforge.net/projects/tidy/

On Linux

Here's how I installed Tidy:

```
$ tidy -help
bash: tidy: command not found
$ cd bulk/run/
$ mkdir tidy && cd tidy
$ wget http://tidy.sourceforge.net/cf/tidy_linux_x86.tgz
$ md5sum --check
476326c3d44292108111841a42bd27f6 *tidy_linux_x86.tgz
tidy_linux_x86.tgz: OK
$ tar -xzf tidy_linux_x86.tgz
$ ed
a
#!/usr/bin/env sh
${HOME}/bulk/run/tidy/bin/tidy "$@"
```

```
.
w /home/tobi/data/commands/tidy
54
q
$ chmod 700 ~/data/commands/tidy
$ tidy -v
HTML Tidy for Linux/x86 released on 1st November 2003
$
```

On Windows

A `tidy.bat` could look like this: (two lines)

Example 4.5. `tidy.bat`

```
@echo off
\path\to\tidy.exe -config /path/to/tidyrc.txt
-f /log/errors/here/tidyerrs.txt %1 %2 %3 %4 %5 %6 %7 %8 %9
```

(put it in a directory which is on the system path)

Settings

Sample tidyrc.txt:

```
word-2000: yes
clean: yes
doctype: strict
bare: yes
drop-font-tags: yes
drop-proprietary-attributes: yes
enclose-block-text: yes
escape-cdata: yes
logical-emphasis: yes
output-xhtml: yes
```

Chapter 5. Tasks

Most of the examples and instructions will only work as shown when carried out in an environment set up as described in the previous chapters. But many of them might work with your settings when you adjust the various details. They are meant as illustrations of the respective general idea and not as perfect recipes.

Creating Tags

Turn on XML mode via `[mapleader] x`, then enter insert mode via `i`.

To create a pair of tags enter the opening tag, you should get the closing one for free. Now you can enter content, for example text.

```
<foo>bar</foo>
```

If you want the content to be indented start again but this time hit > again after having closed the opening tag :

```
<foo>
  bar
</foo>
```

This way you can quickly create nicely indented XML documents:

```
<drinks>
  <juice>
    <mango>delicious</mango>
  </juice>
</drinks>
```

Creating Documents

```
<chapter><title>Flowers</title>
  <section><title>Lilies</title>
    <para>...</para>
  </section>
  <section><title>Orchids</title>
    <para>...</para>
  </section>
</chapter>
```

Here is one way to create the above document : `[mapleader] x i < c h a p t e r > > [up] < t i t l e > F l o w e r s [down] < s e c t i o n > > [up] < t i [ctrl-x] [ctrl-p] > L i l i e s [down] < p a r a > . . . [esc] [down] o < s e c [ctrl-x] [ctrl-p] > > [up] < t i [ctrl-x] [ctrl-p] > O r c h i d s [down] < p a r [ctrl-x] [ctrl-l] [esc]`

There are many different ways to create the same document. For example it would have been faster to create just one section skeleton and duplicate it.

Now you can fold away the section you don't work on, with `2 G z c`:

```
<chapter><title>Flowers</title>
+---  3 lines: <section><title>Lilies</title>----------
  <section><title>Orchids</title>
    <para>...</para>
  </section>
</chapter>
```

To learn more about folding see `:help folding` [url 36] and `:help fold-commands` [url 37].

Marking up Text

With xmledit

Turn on XML editing mode via `[mapleader] x`.

To mark up the literal command name string in

```
<para>
Use cd to go to a different directory.
</para>
```

select it (eg via `2 G w v e`), then type `[mapleader] x` in quick succession (eg `\ x` or `, x`). At the prompt, enter the name of the element, eg **`command`**, then press enter twice. You should get something like the following:

```
<para>
Use <command>cd</command> to go to a different directory.
</para>
```

With Recording

Although the above method is convenient it is not the best choice when there are more than one text portions to be marked up because it can't be repeated or automated easily. But Vim lets you record actions, the documentation describes this under “complex repeats”, see `:help recording` [url 38].

Let's say you want to mark up the acronyms in

```
<para>These XSLTs can be used for transforming
DBX to XHTML.</para>
```

Search for groups of uppercase characters via `/ \ u \ { 2 , } [enter]` Enter n until you reach the first acronym you want to mark up. Then record one of the following macros:

In XML mode (with xmledit)

```
q a i < a c r o n y m > [esc] [right] d % / \ U [enter] P / \ u \ {
2 , } [enter] q
```

Without xmledit

```
q a i < a c r o n y m > [esc] / \ U [enter] i < / a c r o n y m >
[esc] / \ u \ { 2 , } [enter] q
```

Now the cursor should be at the beginning of the next group of uppercase characters. If you

want to mark it up (if it's an acronym that's not yet marked up) do `@-a`, otherwise do `n` to jump to the next group of uppercase letters. To repeat the last executed recorded action do `@-@`. You should get this:

```
<para>These <acronym>XSLT</acronym>s can be used for transforming
<acronym>DBX</acronym> to <acronym>XHTML</acronym>.</para>
```

Marking up Tables

Recording

Let's say you have some space-separated strings which you want to mark up as XHTML table:

```
1  2 3  4   5
2 4    6 8  10
3 6 9  12  15
4 8  12 16 20
5 10    15 20 25
```

One way is to record some sub-tasks:

get set
```
[mapleader] x g g > G : % s / $ / [space] . / [enter] g g O
[ctrl-d] < t a b l e > > [esc] d d . G p 2 G
```

cells
```
q c i < t d > [esc] [right] d % t [space] p [right] d / \ S [enter]
q 4 @ c
```

rows
```
q r x I < t r > [esc] [right] d % $ p + q
```

cells and rows
```
q a 5 @ c @ r q
```

and now comes the fun part
```
3 @ a
```

You should get this:

```
<table>
  <tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td></tr>
  <tr><td>2</td><td>4</td><td>6</td><td>8</td><td>10</td></tr>
  <tr><td>3</td><td>6</td><td>9</td><td>12</td><td>15</td></tr>
  <tr><td>4</td><td>8</td><td>12</td><td>16</td><td>20</td></tr>
  <tr><td>5</td><td>10</td><td>15</td><td>20</td><td>25</td></tr>
</table>
```

Substitution

Using substitution commands should be faster than using recorded macros in XML mode. Hundreds of lines should be marked up instantly, thousands of lines still take less than a second on my box. With thousands of lines, undo can take seconds though; test and adjust your command

line with a small number of representative lines before you run it on larger data sets.

The following examples show how semicolon-separated data which has been exported from a spreadsheet application can be marked up as XHTML table.

If the data is simple you can use simple commands. Here's some CSV data which has
- no empty fields
- no quoted fields (no record separators inside fields, no quotes inside quotes)
- and no fields that span mutiple lines

```
1;2;3
2;4;6
3;6;9
4;8;12
5;10;15
```

Mark up the table cells via

```
:%s/\([^;]*\);\?/<td>\1<\/td>/g
```

or

```
:%!ruby -ne "print gsub(/([^;\n]+);?/,'<td>\1</td>')"
```

so that you get

```
<td>1</td><td>2</td><td>3</td>
<td>2</td><td>4</td><td>6</td>
<td>3</td><td>6</td><td>9</td>
<td>4</td><td>8</td><td>12</td>
<td>5</td><td>10</td><td>15</td>
```

Then mark up the table rows via

```
:%s/.\+/<tr>&<\/tr>/
```

Note

If you want to mark up just a range of lines (eg lines 29 through 33) replace `:%` with `:29,33`.

Alternatively you could chain the two steps:

```
:g/./ s/\([^;]*\);\?/<td>\1<\/td>/g | s/.\+/<tr>&<\/tr>/
```

```
<tr><td>1</td><td>2</td><td>3</td></tr>
<tr><td>2</td><td>4</td><td>6</td></tr>
<tr><td>3</td><td>6</td><td>9</td></tr>
<tr><td>4</td><td>8</td><td>12</td></tr>
<tr><td>5</td><td>10</td><td>15</td></tr>
```

Transforming CVS data to XHTML would typically not be done by manipulating a text editor buffer with commands but by running a little script on a file, but let's try another example just for the fun of it.

Here's some more complex CSV data, as exported from a spreadsheet app:

```
1;2;3;4
```

```
"this field contains a line
break and a record ; separator";4;6;"line
break"
;8;12;
"quoted ""word""";16;;
;;"record ; separator";
```

Inserting a record separator at the end of each record which is at the end of the line simplifies things a bit:

```
:v/"[^";]\+$/ s/$/;/
```

To mark up all records in the buffer as table cells do

```
:%s/\("\(""\_.*""\|\_[^"]*\)"\|[^;]*\);/<td>\1<\/td>/g
```

Mark up table rows:

```
:%s/\(^\(<td>\_.\{-}<\/td>\)\{-}\)\(\n\|$\)/<tr>\1<\/tr>\3/g
```

All this is still based on some assumptions; before you process real data run tests with some lines representing your data and adjust the commands. Cleaning up is not simple either, here's a cheap way to remove the double quotes inside quotes

```
:%s/""/"/g
```

... and the quotes around the fields:

```
:g/./ s/"\(<\/td\)/\1/g | s/\(td>\)"/\1/g
```

A pretty cryptic way to do both at once: (one line)

```
:%!ruby -e "$nlt='[^<]'; print $stdin.read.gsub(Regexp.compile(
'(<td>)\"('+$nlt+'+)\"(<\/td>)')){($1+$2+$3).gsub(/\"\"/,'\"')}"
```

This is what you should get:

```
<tr><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>this field contains a line
break and a record ; separator</td><td>4</td><td>6</td><td>line
break</td></tr>
<tr><td></td><td>8</td><td>12</td><td></td></tr>
<tr><td>quoted "word"</td><td>16</td><td></td><td></td></tr>
<tr><td></td><td></td><td>record ; separator</td><td></td></tr>
```

Now you could insert dashes into the empty cells

```
:%s/\(<td>\)\(<\/td>\)/\1-\2/g
```

mark up the line breaks

```
:v/<\/tr>$/ s/$/<br \/>/
```

and then wrap the whole thing in an XHTML table.

Important

Before you publish web content please check it against the WCAG [url 39]. For example there's a guideline explaining how to make tables accessible [url 40].

Generating XML

From Vim's command line, you can call any tool which is available on your system's path. So when there's a repetitive task like writing lots of similar tags, you could ask your favorite programming language to do it for you.

Grab an SVG skeleton

```
:r /path/to/basictemplate_1_1.svg [url 41]
```

insert a circle, and set the view box, so that you get something like this:

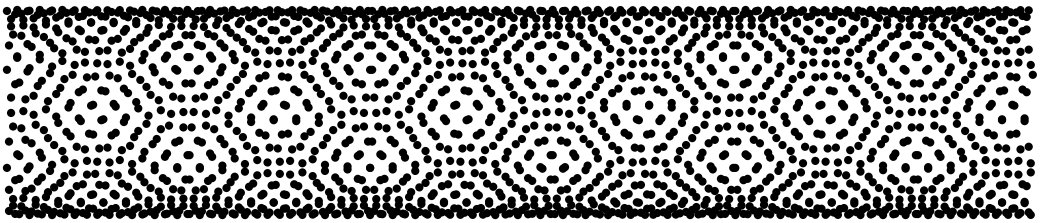
```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<svg version="1.1"
  xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  viewBox="-10 -60 520 120"
  width="400" height="100">
  <title>Snake Skin</title>
  <desc>A generated pattern.</desc>
  <defs>
    <circle id="c" r="2"/>
  </defs>


</svg>
```

then place your cursor where the content should go, eg via `G 2 k`. Enter the following in Vim's command line, all in one line. You can paste it (after having pressed `:`), eg via the middle mouse button (don't forget to delete the duplicate `:`).

```
:r !ruby -e "1.upto(2000){|i|puts '<use xlink:href=\'\#c\' x=\''+
(i/4).to_s+'\' y=\''+((Math.cos(i)*50)).to_s+'\'/>'}"
```

Now save the SVG and open it in some SVG viewer [url 42]. You should see something like this:



Here is the SVG itself: cos_pattern.svg [url 43].

Search and Replace

Via search and replace, you can make hundreds of similar changes at once.

To go from

```
<span class="bold">foo</span>
```

to

```
<em>foo</em>
```

select the block(s) via

```
V}}
```

then enter the following in Vim's command line:

```
:'<,'>s/<span\_s\+class="bold">\(\_.\{-}\)<\/span>/<em>\1<\/em>/g
```

To rename all `sect1` tags to `section` tags, you can ask Vim to execute something like

```
:%s/<\(\/\?\)sect1\(\_s*\)/<\1section\2/gc
```

To index some titles in a DBX document I use the following command (one line):

```
:%s/\(<title>\)\(.\+\)\(<\/title>\)\n\?\(\s*\)/\1\2<indexterm>\r\4
<primary>\2<\/primary>\r\4<\/indexterm>\3\r/gc
```

This only automates part of the task and is not really general but unlike XSLT it leaves CDATA sections intact (simply skip them) which make editing of code listings easier. If you get it to work for titles spanning multiple lines send me an email `:)`.

XPath-based Editing

Sometimes a task is expressed best using XPath. Simpler stuff can be done conveniently from Vim's command line, without having to create an XSLT file.

Rows

Here's an example showing a simple two-step approach to the common task of marking table rows with alternating attributes. This way the command lines become shorter and easier to write. Between each screen the command line is listed which describes the change between the previous and the following screen.

```
<?xml version="1.0"?>
<table>
  <tr>
    <td>foo</td>
  </tr>
  <tr>
    <td>foo</td>
  </tr>
  <tr>
    <td>foo</td>
  </tr>
  <tr>
    <td>foo</td>
  </tr>
</table>
```

```
:%!xmlstar ed -i /table/tr -t attr -n class -v odd
```

Here's what the option names in the above command line stand for:

```
ed
   edit

-i
   insert

-t
   type

-n
   name

-v
   value
```

XMLStarlet also supports long options (eg `--insert` as alternative for `-i`), but the short versions save typing and space, which is especially useful when calling XMLStarlet from the shell and not from a script.

```
<?xml version="1.0"?>
<table>
  <tr class="odd">
    <td>foo</td>
  </tr>
  <tr class="odd">
    <td>foo</td>
  </tr>
  <tr class="odd">
    <td>foo</td>
  </tr>
  <tr class="odd">
    <td>foo</td>
  </tr>
</table>
```

```
:%!xmlstar ed -u "/table/tr[(position() mod 2)=0]/@class" -v even
```

```
<?xml version="1.0"?>
<table>
  <tr class="odd">
    <td>foo</td>
  </tr>
  <tr class="even">
    <td>foo</td>
  </tr>
  <tr class="odd">
    <td>foo</td>
  </tr>
  <tr class="even">
    <td>foo</td>
  </tr>
</table>
```

`-u` stands for or “update”.

Chapter 6. More Tasks

Questions

Sometimes you need to find out facts about the buffer with which you're working. Here are some examples: (each command line is to be entered as one line)

How many `indexterm` elements are there?

```
:%w !xmlstar sel -t -v "count(//indexterm)"
```

What's the title of the first `section` which has more than one nested `sections`?

```
:%w !xmlstar sel -t -v "//section[count(.//section)&gt;1]/title"
```

What's the title of the first `section` which is nested deeper than one level?

```
:%w !xmlstar sel -t -v "//section[count(ancestor::section)&gt;1]/title"
```

How many `programlistings` are there which don't have a `role` attribute?

```
:%w !xmlstar sel -t -v "count(//programlisting[not(@role)])"
```

How many XHTML `div` elements are in the document?

```
:%w !xmlstar sel -N "xh=http://www.w3.org/1999/xhtml"
 -t -v "count(//xh:div)"
```

How many words are there in the document?

```
:%w !xmlstar sel -t -v "/" |
 ruby -e 'puts($stdin.read.scan(/\S+/).length)'
```

You might need to exchange the apostrophes around the Ruby code for quotes on Windows. To count the words inside the (first) chapter with title “Foo” replace `"/"` with `"//chapter[title='Foo']"`. If **wc** is available on your system you can try `| wc -w` instead of `| ruby -e` *`[...]`*.

Validation

If you just want to check for well-formedness, enter

```
:!xmllint --noout %
```

Most often you will want to validate the document. The following command writes the buffer to xmllint:

```
:%w !xmllint --valid --noout -
```

This means you can validate the document with all changes, without having to save it first. If you set up a shell script or batch file just for validation, eg

Example 6.1. `xmllintval.bat`

```
@echo off
xmllint --valid --noout %1 %2 %3 %4 %5
```

then you can simply do

```
:%w !xmllintval -
```

If you want to validate the file, pass the file's path to the validator:

```
:!xmllint --valid --noout %
```

or

```
:!xmllintval %
```

xmllint also allows you to validate documents which don't have a document type declaration . Let's say you have a DBX chapter file

```
<?xml version="1.0" encoding="UTF-8"?>
<chapter>
  <title>Setup</title>
...
```

which is included in the main book file via an entity reference

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE book
  PUBLIC "-//OASIS//DTD DocBook XML V4.2//EN"
  "http://www.oasis-open.org/docbook/xml/4.2/docbookx.dtd"
[
 <!ENTITY setup SYSTEM "setup.dbx">
 ...
]>
...
&setup;
...
```

This means the chapter file can't have a doctype declaration. XInclude is a solution for that problem but it's not yet as widely supported as the entity reference mechanism which is part of the XML standard itself. Being able to validate documents which have no document type declaration is generally useful, and factors like the growing popularity of RNG, namespaces and version/profile attributes, and the hopefully decreasing dependence on DTDs to modify the document's information set will probably mean that less documents will have document type declarations. So let's say you edit the chapter in Vim and want to validate the buffer.

```
:%w !xmllint --valid --noout -
```

will fail because there is no FPI included and no DTD referenced. So you can try the following (one line):

```
:%w !xmllint --dtdvalid
```

```
 "http://www.oasis-open.org/docbook/xml/4.2/docbookx.dtd" --noout -
```

If you set up the catalog as described in the previous chapter then this works offline. (It won't work if the chapter contains references to entities that are declared in the main (book) file. In this case you probably would feed the main file to the validator.) The line doesn't have to be changed when used on a different machine which means that you can add it to your crossplatform `vimrc`:

```
nmap <Leader>d4 :%w !xmllint --dtdvalid
 \ "http://www.oasis-open.org/docbook/xml/4.2/docbookx.dtd"
 \ --noout -<CR>
```

Alternatively you can pass the FPI to **xmllint** (one line)

```
:%w !xmllint --dtdvalidfpi
 "-//OASIS//DTD DocBook XML V4.2//EN" --noout -
```

if it's listed in a catalog. `[mapleader]-d-4` works great for DBX chapters which are external entities referenced from the main book file as long as the following requirements are met: In addition to being an external parsed entity (can't contain a doctype declaration) the file must also be an XML document. In other words: Since the file can't contain entity declarations it also can't contain entity references. Making sure that modules of a DBX document are well-formed XML documents that can stand alone is generally a good idea since it simplifies collaboration and exchange, reuse, and last but not least editing and validation.

To validate the buffer with RXP, type

```
:%w !rxp -V -N -s -x
```

If you have set up **rxpval** you can enter

```
:%w !rxpval
```

This will not work if there are relative references to external entities and the current directory is not that of the file being validated (this also goes for stuff like `:%w !xmllintval -`). When RXP or xmllint eat stdin they use the current directory as base directory when resolving relative references. So if you want to validate a buffer which has relative references to external entities, you can do

`:cd` *`/directory/of/the/file/`*

to change the working directory, and

```
:pwd
```

to check it. A more convenient way to change the working directory is to put

```
map <Leader>cd :exe 'cd ' . expand ("%:p:h")<CR>
```

into your `vimrc` then do `[mapleader] c d` before writing the buffer to the validator via `:%w !`*`command`*`...`.

If you want to validate the file, then it's not necessary to change the working directory. You can simply pass the file path, eg

```
:!rxp -V -N -s -x %
```

or

```
:!rxpval %
```

RXP works with XHTML documents, but with DBX documents I experienced problems which hopefully will be resolved with future releases. There are various command line options; check the RXP man page [url 44].

The commands are long enough to be tedious to type each time. After you've entered a specific one once, you can look for it in the command history: Type `:` to go to Vim's command line, optionally type the start of the command, then repeat `[up]` until you see the command you were looking for. Now you can edit it, or enter it unmodified. Another way of saving typing is to map commands to a very short key sequence in the `vimrc`.

Pretty-printing

Pretty-printing is not a trivial task. Relevant details of your document might be changed, so beware. I mainly use pretty-printing when viewing documents with extremely long lines or no indention, or when editing XML generated by tools, but I nearly never use a pretty-printing tool to format the code of documents I author.

Warning

Again: Your data can get corrupted whenever you filter the buffer. This goes for search'n'replace, external tools, etc. Use u to undo.

Select a well-formed fragment (one root element), then filter it through **xmllint**'s pretty-printer, by entering

```
!xmllint --format -
```

xmllint will insert an XML prolog; if you didn't filter the whole buffer, this probably isn't desired. You can map this filter command to some shorter key sequence, and also include some commands to delete the prolog.

Caution

Namespace prefixes not declared inside the fragment are stripped.

To pretty-print the whole buffer, do

```
:%!xmllint --format -
```

Example

Let's say you receive a file which looks like this:

```
<?xml version="1.0"?>
<chapter><title>Ökopläne</title><simplelist>
<member role="überfällig">übermäßige
Ölförderung stoppen</member></simplelist></chapter>
```

The code is laid out in a way which makes it hard to work with. This

```
:%!xmllint --format --encode UTF-8 -
```

should bring

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<chapter>
  <title>Ökopläne</title>
  <simplelist>
    <member role="&#xFC;berf&#xE4;llig">übermäßige
Ölförderung stoppen</member>
  </simplelist>
</chapter>
```

which looks better, but **xmllint** currently doesn't resolve the NCRs of "special characters" (probably those outside the US-ASCII range) inside attribute values which can make editing harder.

You could also try **tidy** for pretty-printing.

Cleaning up

To delete all comments do

```
:%!xmlstar ed -d //comment()
```

HTML

If you have to deal with tag soup you might want to try turning it into XHTML so that you can enjoy all the advantages of XML. To filter the whole buffer through Tidy, do

```
:%!tidy
```

If there are irrecoverable errors, eg unknown elements, everything will be deleted. You can undo the filtering via u.

To filter a block, do

```
!}tidy
```

but don't forget that depending on the `doctype` flag Tidy might insert a doctype declaration if there is none.

If there's unwanted stuff left, strip tags like this:

```
:%s,</\?u>,,g
```

Chapter 7. ed

And now for something completely different, just for the fun of it :)

Let's explore the ways of Vim's grand-grandfather, the standard Unix editor **ed**.

Why **ed**?

Working with **ed** is like walking in the dark. Although you do have a flashlight, going by train or car would be much more convenient.

But for certain tasks this ancient editor still is useful, for example when you want to show the editing steps to someone (every command is visible on the screen), or apply them automatically.

Basics

We'll use GNU ed, which might be on your system already if you're on Linux.

```
$ ed --version
GNU ed version 0.2
```

Note

You can try out the original ed at the UNIX Seventh Edition page [url 45] of The Online Software Museum [url 46], eg by doing

```
$ telnet dino.sysun.com 4004
```

Let's try to create a tiny XML file:

```
$ ed -p '§ '
§ <fruits>
?
§ h
Unknown command
§ q
$
```

The ? shows that there was an error, and command h caused ed to print the error message. q stands for quit.

Just like Vim, **ed** has modes. When started it's in command mode. Input commands such as a (append) or `i` (insert) make **ed** switch to input mode. So let's try it again:

```
$ ed -p '§ '
§ a
<fruits>
  <fruit name="Mango"/>
```

```
  <fruit name="Rsapberry"/>
</fruits>
.
§ w fruits.xml
71
§ q
$
```

The dot causes ed to switch back to command mode (it only has two). `-p '§ '` sets the prompt which indicates command mode. `w` and `q` are familiar from Vim.

Let's Go

I misspelled the name of the second fruit. I could have deleted the rest of the line (with `[backspace]`) then complete it correctly (the arrow keys don't work, at least here), but I already had entered that line (pressed `[enter]`). There are various ways to correct it. We could change the whole line with c

```
$ ed -p '§ ' fruits.xml
71
§ 3c
  <fruit name="Raspberry"/>
.
§ wq
71
$ tee < fruits.xml
<fruits>
  <fruit name="Mango"/>
  <fruit name="Raspberry"/>
</fruits>
$
```

or we could substitute the erratic string:

```
$ ed -p '§ ' fruits.xml
71
§ 3s/Rsap/Rasp/p
  <fruit name="Raspberry"/>
```

After the substitution command the changed line becomes the current line. The p after the substitution prints the new version of the line, and `%p` would have printed whole document.

Now let's add some attributes to each `fruit` element. We can walk through them with the `G/regex/` command:

```
§ G/<fruit /
  <fruit name="Mango"/>
s/fruit /&latin-name="Mangifera Indica" /p
  <fruit latin-name="Mangifera Indica" name="Mango"/>
  <fruit name="Raspberry"/>
s/fruit /&latin-name="Rubus Idaeus" /p
  <fruit latin-name="Rubus Idaeus" name="Raspberry"/>
```

Let's double the indent:

```
§ g/^ /s/ \+/&&/
§ %p
<fruits>
    <fruit latin-name="Mangifera Indica" name="Mango"/>
```

```
    <fruit latin-name="Rubus Idaeus" name="Raspberry"/>
</fruits>
```

Pretty printing

... can be done like shown in the following example, and makes it easier to enter many elements:

```
$ ed -p '§ '
§ a
<a><b ><c /> </b></a>
.
§ w pp.xml
22
§ e !xmllint --format %
xmllint --format pp.xml
53
§ %n
1       <?xml version="1.0"?>
2       <a>
3         <b>
4           <c/>
5         </b>
6       </a>
§ 4a
<d><e><f/></e></d>
.
§ w
72
§ e !xmllint --format %
xmllint --format pp.xml
104
§ %p
<?xml version="1.0"?>
<a>
  <b>
    <c/>
    <d>
      <e>
        <f/>
      </e>
    </d>
  </b>
</a>
§ wq
104
$
```

More

Ed offers a lot more, check **man ed**.

Here's an example:

```
$ ed -p '§ '
§ w news.rss
0
§ a
```

```
wget -q -O - \
'http://p.moreover.com/cgi-local/page?c=Open%20source%20news&o=rss' \
| xmlstar sel -t -c '/rss/channel/item[
position() &lt; 5 and position() &gt; 1]'
.
§ w !tee >get_news
167
§ !chmod 700 get_news
!
§ e !./get_news
648
§ 1i
<?xml version="1.0"?>
<rss version="0.91">
  <channel>
.
§ $a
  </channel>
</rss>
.
§ wq
723
$
```

Chapter 8. Thanks

Thanks to all those who develop and support Free software, write Free documentation, are active and helpful on the lists, develop languages, or provide infrastructure.

Thanks to those who love coding and those who enjoy sharing their knowledge.

Thanks to the community.

Have fun :)

Acknowledgements

Rich Caloggero [url 47] and David Pawson [url 48] provided some feedback regarding the accessibility of the HTML version. David Turner gave feedback on the license. Emma Jane Hogbin posted a review at discuss@en.tldp.org [url 49] (here's my reply to Emma's review [url 50], and here's the thread [url 51]). Thanks!

If you think you should be listed here, if you would like to volunteer as technical reviewer, or if you want to offer code or content, please drop me a line at <tobiasreif pinkjuice com>. Any feedback is welcome, especially error reports.

Chapter 9. License

This document (Vim as XML Editor) is copyright Tobias Reif.

As long I'm maintaining the document I won't publish copies elsewhere on the web and I don't encourage you to do so either. It's hard to keep multiple copies up to date and people will be confused by finding potentially different versions at multiple locations when searching the web. At some point I might offer the document to http://tldp.org/ and move it there if they accept it.

Distributing printed copies on the other hand is alright, the conditions are detailed below.

The document (sources, presentations) is released under the below license. All code in the document is released under the GNU GPL.

Freedoc License

About

This copyleft license is a draft. It tries to describe what freedom can mean for documentation, its users, and its authors.

This license should not violate the spirit of the DFSG or the GNU copyleft definition [url 52].

Sources
: The files that were created when the original document was created. They should should adhere to an open standard and must be easily editable with a text editor: words are words, vectors are vectors. This requirement doesn't apply to photographs, sound files, videos, etc. The sources are all files which are needed to create a complete presentation. This can for example be the DocBook sources with all media objects such as images (SVG, PNG, etc), sound files, movies, etc. Or the source files for a documentation presentation may consist of source code (with or without embedded documentation). If the document's sources and it's presentation consist of the same set of files then the format of the presentation document must adhere to the same rules as detailed for source files above. The sources of a modified version are in the same format as the original document's sources.

Presentation
: What's offered to the audience (eg readers, listeners, “users”), for example a book, an article in a print magazine, a collection of HTML pages, a PDF, slides, a live talk, a video, or an audio file.

Modified version
: Documents which are a copy of this document where some parts are modified, documents which contain parts of this document, and documents which are based on this document (eg translations).

Distribution
: ... includes publishing alone and as part of a collection.

Main

This document is free documentation, it can't be locked in. You're free to use, copy, share, distribute, display, or sell it (in source or presentation form). You're also free to modify it, for example in order to keep it up to date after the original author(s) stop(s) maintaining it or to improve it by evolving or expanding it, or in order to translate it.

It's important that users are allowed to keep the the technical content of the documentation in sync with the evolving technology it describes. It also is important that the audience is free to generally evolve the document in case it is not being maintained anymore, for example by adding examples illustrating usage of new technologies etc. But although modification is allowed you're obviously required to exercise common sense and to obey common rules and (n)etiquette. This license is based on the belief that it shouldn't be necessary to explicitly regulate all possible abuse of the freedom this license grants. It obviously is not right to modify text which expresses the views of someone other than yourself, or to modify quoted text. Defamation is morally wrong and socially unacceptable (libel for example is illegal under certain jurisdictions).

When you distribute modified versions of this document you must state that you modified the document and you should also specify when and how you changed which portions of the document, and say where the original can be found. This text must appear prominently in the sources and in the presentation. If the presentation is of a type which makes the latter problematic (eg audio) it's OK to place the text near the presentation, eg on the packaging.

When you distribute a copy or a modified version of this document as presentation then you must provide the sources of the presentation at no cost, for example as free download, or on a CD for the price of the blank CD.

When you distribute an unmodified copy of the original document please use the latest version and keep your publication up to date by checking for the latest version before each publication cycle (eg site update or print run). If the presentation is available on a tangible medium (eg as book) please consider offering a copy to the original author.

If you distribute a modified version or a copy of this document you must preserve the original author's copyright notice and you must license it under the terms of this license, for example by including this license unchanged, or by linking to it.

Any and all imaginable and unimaginable disclaimers apply to the document, and also to this license.

This license can't meet all requirements of all authors and users. But if it limits the document's or your freedom in unacceptable ways please drop me a line at `tobiasreif pinkjuice com`.

Glossary

ASCII (American Standard Code for Information Interchange)

A tiny character set.

ASF (Apache Software Foundation)

ASF [url 53]

BSD (Berkeley Software Distribution)

FreeBSD, OpenBSD, and NetBSD are Unix-like OSs.

CSS (Cascading Style Sheets)

CSS [url 54]

CDATA (character data)

Inside CDATA sections [url 55] the XML parser treats markup constructs as character data, not as markup.

DBX (DocBook XML)

schemas [url 56]

MS-DOS (MicroSoft Disk Operating System)

“MS-DOS” in the Jargon File [url 57]

DTD (Document Type Definition)

The DTD schema language is specified in the XML specification. [url 58]

DFSG (Debian Free Software Guidelines)

DFSG [url 59]

FAQ (Frequently Asked Questions)

A list of question - answer pairs.

FOP (Formatting Objects Processor)

FOP [url 60]

FPI (Formal Public Identifier)

FPIs are described in the XML specification. [url 61]

FDL (GNU Free Documentation License)

FDL [url 62]

GNU (GNU's Not Unix!)

GNU [url 63]

GPL (General Public License)

GPL [url 64]

GNOME (GNU Network Object Model Environment)

GNOME [url 65]

GUI (Graphical User Interface)

Vim can be used in text-based environments and in GUI environments.

HTML (HyperText Markup Language)	HTML [url 66]
IE (Internet Explorer)	IE [url 67]
JRE (Java Runtime Environment)	Java [url 68]
JVM (Java Virtual Machine)	JVM spec [url 69]
NG (Next Generation)	“NG” sometimes is used as part of a name for a technology.
NCR (Numerical Character Reference)	Also “Numeric Character Reference”. Definition of “character reference” in the XML specification. [url 70]
NN (Netscape Navigator)	NN [url 71]
OS (Operating System)	Windows, Linux, and Mac OS are operating systems.
OASIS (Organization for the Advancement of Structured Information Standards)	OASIS [url 72]
PDF (Portable Document Format)	Popular print file format.
PNG (Portable Network Graphics)	PNG [url 73]
rc file	See “rc file” in the Jargon File [url 74] and http://www.multicians.org/shell.html.
RELAX (Regular Language Description for XML)	An XML schema language.
RNG (RELAX NG)	Based on RELAX Core and TREX. (pronounced “relaxing”) RELAX NG [url 75], RELAX NG at OASIS [url 76]
SVG (Scalable Vector Graphics)	SVG [url 77]
TMTOWTDI (There's More Than One Way To Do It)	The Jargon Lexicon [url 78].
URI (Uniform Resource Identifiers)	“Naming and Addressing: URIs, URLs, ...” [url 79]
URL (Uniform Resource Locator)	

	“Naming and Addressing: URIs, URLs, ...” [url 80]
WAI (Web Accessibility Initiative)	WAI [url 81]
WCAG (Web Content Accessibility Guidelines)	Web Content Accessibility Guidelines [url 82], WAI checklist [url 83]
WXS (W3C XML Schema)	WXS [url 84]
XHTML (Extensible HyperText Markup Language)	XHTML 1.0 [url 85]
XML (Extensible Markup Language)	XML specification [url 86]
XPath (XML Path Language)	specification of version 1.0 [url 87], specification of version 2.0 [url 88]
XSD (XML Schema Definition)	a WXS instance
XSL (Extensible Stylesheet Language)	XSL home [url 89], XSL specification [url 90]
XSLT (XSL Transformations)	XSLT 1.0 [url 91]
XSLFO (XSL Formatting Objects)	XSLFO 1.0 [url 92]

Index

U

V

W

X

Appendix A. URLs

[url 1] rate this howto at www.vim.org
http://www.vim.org/tips/tip.php?tip_id=583

[url 2] my Vim page
http://www.pinkjuice.com/vim/

[url 3] nXML
http://www.xmlhack.com/read.php?item=2061

[url 4] Vim's Python interface
http://vim.sourceforge.net/htmldoc/if_pyth.html

[url 5] libxml's Python bindings
http://www.xmlsoft.org/python.html

[url 6] Vim XML Wiki page
http://www.protocol7.com/svg-wiki/?VimXml

[url 7] XML scripts
http://www.vim.org/scripts/script_search_results.php?keywords=xml

[url 8] XML tips
http://www.vim.org/tips/tip_search_results.php?keywords=xml

[url 9] donation
http://iccf-holland.org/donate.html

[url 10] ICCF Holland
http://iccf-holland.org/

[url 11] `:help uganda`
http://vim.sourceforge.net/htmldoc/uganda.html#uganda

[url 12] the book page on iccf-holland.org
http://iccf-holland.org/click5.html

[url 13] list of errata
http://www.moolenaar.net/vim_errata.html

[url 14] `:help mapleader`
http://vim.sourceforge.net/htmldoc/map.html#mapleader

[url 15] The Vim download page
http://www.vim.org/download.php

[url 16] DBX sources
http://www.pinkjuice.com/howto/vimxml/docbook/

[url 17] pics
http://www.pinkjuice.com/howto/vimxml/pics/

[url 18] docbook.sf.net XSLTs
http://docbook.sourceforge.net/projects/xsl/

[url 19] DocBook XSL: The Complete Guide
http://www.sagehill.net/docbookxsl/

[url 20] the XSLT files used to generate the online version of this howto
http://www.pinkjuice.com/howto/vimxml/xslt/tinydbk2xhtml/

[url 21] packaged releases
http://sourceforge.net/project/showfiles.php?group_id=21935

[url 22] support
http://docbook.sourceforge.net/support.html

[url 23] Joocs
http://www.pinkjuice.com/joocs/

[url 24] customization layer
http://www.pinkjuice.com/howto/vimxml/xslt/docbook_sf_net_fo/

[url 25] XSLFO and PDF files
http://www.pinkjuice.com/howto/vimxml/print/

[url 26] `:help add-local-help`
http://vim.sourceforge.net/htmldoc/usr_05.html#add-local-help

[url 27] OASIS XML Catalogs
http://www.oasis-open.org/committees/entity/

[url 28] OASIS Catalogs specification
http://www.oasis-open.org/committees/entity/spec-2001-08-06.html

[url 29] SVG skeleton
http://www.pinkjuice.com/txt/basictemplate.svg

[url 30] Namespaces in XML 1.1
http://www.w3.org/TR/xml-names11/

[url 31] Rich Kilmer's “Building Ruby 1.8.1 on Panther”
http://richkilmer.blogs.com/ether/2003/10/building_ruby_1.html

[url 32] XHTML 1.1 Appendix A
http://www.w3.org/TR/xhtml11/changes.html#a_changes

[url 33] SVG 1.1 RNG
http://www.w3.org/Graphics/SVG/1.1/rng/

[url 34] **wget** home page
http://www.gnu.org/software/wget/wget.html

[url 35] alternative **wget** home page
http://wget.sunsite.dk/

[url 36] `:help folding`
http://vim.sourceforge.net/htmldoc/fold.html#folding

[url 37] `:help fold-commands`
http://vim.sourceforge.net/htmldoc/fold.html#fold-commands

[url 38] `:help recording`

http://vim.sourceforge.net/htmldoc/repeat.html#recording

[url 39] WCAG
http://www.w3.org/TR/WAI-WEBCONTENT/

[url 40] how to make tables accessible
http://www.w3.org/TR/WAI-WEBCONTENT/#gl-table-markup

[url 41] basictemplate_1_1.svg
http://www.pinkjuice.com/txt/basictemplate_1_1.svg

[url 42] some SVG viewer
http://www.protocol7.com/svg-wiki/?ViewerImplementations

[url 43] cos_pattern.svg
http://www.pinkjuice.com/howto/vimxml/pics/cos_pattern.svg

[url 44] the RXP man page
ftp://ftp.cogsci.ed.ac.uk/pub/richard/rxp.txt

[url 45] the UNIX Seventh Edition page
http://museum.sysun.com/museum/u7conn.html

[url 46] The Online Software Museum
http://museum.sysun.com/museum/index.html

[url 47] Rich Caloggero
http://www.accessexpressed.net/

[url 48] David Pawson
http://www.dpawson.co.uk/

[url 49] Emma Jane Hogbin posted a review at discuss@en.tldp.org
http://lists.tldp.org/index.cgi?1:mss:6362

[url 50] my reply to Emma's review
http://lists.tldp.org/index.cgi?1:mss:6604

[url 51] thread
http://lists.tldp.org/index.cgi?1:sss:6604:200401:gbkcamoaipcapobjdklo

[url 52] GNU copyleft definition
http://www.gnu.org/copyleft/

[url 53] ASF
http://www.apache.org/

[url 54] CSS
http://www.w3.org/Style/CSS/

[url 55] CDATA sections
http://www.w3.org/TR/REC-xml#sec-cdata-sect

[url 56] schemas
http://www.oasis-open.org/docbook/xml/

[url 57] “MS-DOS” in the Jargon File
http://catb.org/esr/jargon/html/M/MS-DOS.html

[url 58] The DTD schema language is specified in the XML specification.
http://www.w3.org/TR/REC-xml

[url 59] DFSG
http://www.debian.org/social_contract.html#guidelines

[url 60] FOP
http://xml.apache.org/fop/

[url 61] FPIs are described in the XML specification.
http://www.w3.org/TR/REC-xml.html#NT-PubidLiteral

[url 62] FDL
http://www.gnu.org/licenses/fdl.html

[url 63] GNU
http://www.gnu.org/

[url 64] GPL
http://www.gnu.org/copyleft/gpl.html

[url 65] GNOME
http://www.gnome.org/

[url 66] HTML
http://www.w3.org/MarkUp/

[url 67] IE
http://www.microsoft.com/windows/ie/

[url 68] Java
http://java.sun.com/

[url 69] JVM spec
http://java.sun.com/docs/books/vmspec/

[url 70] Definition of “character reference” in the XML specification.
http://www.w3.org/TR/REC-xml.html#dt-charref

[url 71] NN
http://channels.netscape.com/ns/browsers/

[url 72] OASIS
http://www.oasis-open.org/

[url 73] PNG
http://www.w3.org/Graphics/PNG/

[url 74] “rc file” in the Jargon File
http://catb.org/esr/jargon/html/R/rc-file.html

[url 75] RELAX NG
http://www.relaxng.org/

[url 76] RELAX NG at OASIS
http://www.oasis-open.org/committees/relax-ng/

[url 77] SVG

http://www.w3.org/Graphics/SVG/

[url 78] The Jargon Lexicon
http://www.tuxedo.org/~esr/jargon/html/entry/TMTOWTDI.html

[url 79] “Naming and Addressing: URIs, URLs, ...”
http://www.w3.org/Addressing/

[url 80] “Naming and Addressing: URIs, URLs, ...”
http://www.w3.org/Addressing/

[url 81] WAI
http://www.w3.org/WAI/

[url 82] Web Content Accessibility Guidelines
http://www.w3.org/TR/WAI-WEBCONTENT/

[url 83] WAI checklist
http://www.w3.org/TR/WAI-WEBCONTENT/full-checklist.html

[url 84] WXS
http://www.w3.org/XML/Schema

[url 85] XHTML 1.0
http://www.w3.org/TR/xhtml1/

[url 86] XML specification
http://www.w3.org/XML/

[url 87] specification of version 1.0
http://www.w3.org/TR/xpath

[url 88] specification of version 2.0
http://www.w3.org/TR/xpath20/

[url 89] XSL home
http://www.w3.org/Style/XSL/

[url 90] XSL specification
http://www.w3.org/TR/xsl/

[url 91] XSLT 1.0
http://www.w3.org/TR/xslt

[url 92] XSLFO 1.0
http://www.w3.org/TR/xsl/