XmlStarlet Command Line XML Toolkit User's Guide

Mikhail Grushinskiy

XmlStarlet Command Line XML Toolkit User's Guide

by Mikhail Grushinskiy

Table of Contents

Chapter 1. Introduction

1.1. About XmlStarlet

XMLStarlet (http://xmlstar.sourceforge.net/) 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.

This set of command line utilities can be used by those who deal with many XML documents on UNIX shell command prompt as well as for automated XML processing with shell scripts.

XMLStarlet command line utility is written in C and uses libxml2 and libxslt from http://xmlsoft.org/. Implementation of extensive choice of options for XMLStarlet utility was only possible because of rich feature set of libxml2 and libxslt (many thanks to the developers of those libraries for great work).

'diff' and 'patch' options are not currently implemented. Other features need some work too. Please, send an email to the project administrator (see http://sourceforge.net/projects/xmlstar/) if you wish to help.

XMLStarlet is linked statically to both libxml2 and libxslt, so generally all you need to process XML documents is one executable file. To run XmlStarlet utility you can simple type 'xml' on command line and see list of options available.

XMLStarlet is open source freeware under MIT license which allows free use and distribution for both commercial and non-commercial projects.

We welcome any user's feedback on this project which would greatly help us to improve its quality. Comments, suggestions, feature requests, bug reports can be done via SourceForge project web site (see XMLStarlet Sourceforge forums (http://sourceforge.net/forum/?group_id=66612), or XMLStarlet mailing list (http://lists.sourceforge.net/lists/listinfo/xmlstar-devel/))

1.2. Main Features

The toolkit's feature set includes options to:

- Check or validate XML files (simple well-formedness check, DTD, XSD, RelaxNG)
- Calculate values of XPath expressions on XML files (such as running sums, etc)
- Search XML files for matches to given XPath expressions
- Apply XSLT stylesheets to XML documents (including EXSLT support, and passing parameters to stylesheets)

- Query XML documents (ex. query for value of some elements of attributes, sorting, etc)
- Modify or edit XML documents (ex. delete some elements)
- Format or "beautify" XML documents (as changing indentation, etc)
- Fetch XML documents using http:// or ftp:// URLs
- Browse tree structure of XML documents (in similar way to 'ls' command for directories)
- Include one XML document into another using XInclude
- XML c14n canonicalization
- Escape/unescape special XML characters in input text
- Print directory as XML document
- Convert XML into PYX format (based on ESIS - ISO 8879), and vice versa

1.3. Supported Platforms

Here is a list of platforms on which XmlStarlet is known to work.

- Linux
- Solaris
- Windows
- MacOS X
- FreeBSD
- HP-UX

You might be able to compile and make it on others too.

Chapter 2. Installation

2.1. Installation on Linux

Execute the following command as root

```
rpm -i xmlstarlet-x.x.x-1.i386.rpm
```

where x.x.x indicates package version.

You can use http://rpmfind.net (http://fr2.rpmfind.net/linux/rpm2html/search.php?query=xmlstarlet&system=&arch=) to search for RPM appropriate for your distribution.

2.2. Installation on Solaris

Execute the following commands as root

```
gunzip xmlstarlet-x.x.x-sol8-sparc-local.gz
pkgadd -d xmlstarlet-x.x.x-sol8-sparc-local all
```

2.3. Installation on MacOS X

XmlStarlet is available on MacOS in Fink. See fink.sourceforge.net (http://fink.sourceforge.net/pdb/package.php/xmlstarlet)

2.4. Installation on Windows

Unzip the file xmlstarlet-x.x.x-win32.zip to some directory. To take advantage of UNIX shell scripting you might want to run XmlStarlet from Cygwin. Consider installing Cygwin (http://www.cygwin.com/) on your Windows machine.

Chapter 3. Getting Started

3.1. Basic Command-Line Options

Basic command line syntax:

```
bash-2.03$ xml
XMLStarlet Toolkit: Command line utilities for XML
Usage: xml [<options>] <command> [<cmd-options>]
where <command> is one of:
   ed    (or edit)      - Edit/Update XML document(s)
   sel   (or select)    - Select data or query XML document(s) (XPATH, etc)
   tr    (or transform) - Transform XML document(s) using XSLT
   val   (or validate)  - Validate XML document(s) (well-formed/DTD/XSD/RelaxNG)
   fo    (or format)    - Format XML document(s)
   el    (or elements)  - Display element structure of XML document
   c14n  (or canonic)   - XML canonicalization
   ls    (or list)      - List directory as XML
   esc   (or escape)    - Escape special XML characters
   unesc (or unescape)  - Unescape special XML characters
   pyx   (or xmln)      - Convert XML into PYX format (based on ESIS - ISO 8879)
   p2x   (or depyx)     - Convert PYX into XML
<options> are:
   --version            - show version
   --help               - show help
Wherever file name mentioned in command help it is assumed
that URL can be used instead as well.

Type: xml <command> --help <ENTER> for command help

XMLStarlet is a command line toolkit to query/edit/check/transform
XML documents (for more information see http://xmlstar.sourceforge.net/)
```

3.2. Studying Structure of XML Document

Before you do anything with your XML document you probably would like to know its structure at first. ’el’ option could be used for this purpose.

Let’s say you have the following XML document (table.xml)

```
<xml>
  <table>
    <rec id="1">
      <numField>123</numField>
```

```
      <stringField>String Value</stringField>
    </rec>
    <rec id="2">
      <numField>346</numField>
      <stringField>Text Value</stringField>
    </rec>
    <rec id="3">
      <numField>-23</numField>
      <stringField>stringValue</stringField>
    </rec>
  </table>
</xml>
```

```
xml el table.xml
```

would produce the following output.

```
xml
xml/table
xml/table/rec
xml/table/rec/numField
xml/table/rec/stringField
xml/table/rec
xml/table/rec/numField
xml/table/rec/stringField
xml/table/rec
xml/table/rec/numField
xml/table/rec/stringField
```

Every line in this output is an XPath expression which indicates a 'path' to elements in XML document. You would use these XPath expressions to navigate through your XML documents in other XmlStarlet options.

XML documents can be pretty large but with a very simple structure. (This is espesially true for data driven XML documents ex: XML formatted result of select from SQL table). If you just interested in structure but not order of the elements you can use -u switch combined with 'el' option.

EXAMPLE:

```
xml el -u table.xml
```

Output:

```
xml
xml/table
xml/table/rec
xml/table/rec/numField
xml/table/rec/stringField
```

If you are interested not just in elements of your XML document, but you want to see attributes as well you can use -a switch with 'el' option. And every line of the output will still be a valid XPath expression.

EXAMPLE:

```
xml el -a table.xml
```

Output:

```
xml
xml/table
xml/table/rec
xml/table/rec/@id
xml/table/rec/numField
xml/table/rec/stringField
xml/table/rec
xml/table/rec/@id
xml/table/rec/numField
xml/table/rec/stringField
xml/table/rec
xml/table/rec/@id
xml/table/rec/numField
xml/table/rec/stringField
```

If you are looking for attribute values as well use -v switch of 'el' option. And again - every line of output is a valid XPath expression.

EXAMPLE:

```
xml el -v table.xml
```

Output:

```
xml
xml/table
xml/table/rec[@id='1']
xml/table/rec/numField
xml/table/rec/stringField
xml/table/rec[@id='2']
xml/table/rec/numField
xml/table/rec/stringField
xml/table/rec[@id='3']
xml/table/rec/numField
xml/table/rec/stringField
```

Chapter 4. XmlStarlet Reference

4.1. Querying XML documents

XmlStarlet 'select' or 'sel' option can be used to query or search XML documents. Here is synopsis for 'xml sel' command:

```
XMLStarlet Toolkit: Select from XML document(s)
Usage: xml sel <global-options> {<template>} [ <xml-file> ... ]
where
  <global-options> - global options for selecting
  <xml-file> - input XML document file name/uri (stdin is used if missing)
  <template> - template for querying XML document with following syntax:

<global-options> are:
  -C or --comp       - display generated XSLT
  -R or --root       - print root element <xsl-select>
  -T or --text       - output is text (default is XML)
  -I or --indent     - indent output
  -D or --xml-decl   - do not omit xml declaration line
  -B or --noblanks   - remove insignificant spaces from XML tree
  -N <name>=<value>  - predefine namespaces (name without 'xmlns:')
                       ex: xsql=urn:oracle-xsql
                       Multiple -N options are allowed.
  --net              - allow fetch DTDs or entities over network
  --help             - display help

Syntax for templates: -t|--template <options>
where <options>
  -c or --copy-of <xpath>  - print copy of XPATH expression
  -v or --value-of <xpath> - print value of XPATH expression
  -o or --output <string>  - output string literal
  -n or --nl               - print new line
  -f or --inp-name         - print input file name (or URL)
  -m or --match <xpath>    - match XPATH expression
  -i or --if <test-xpath>  - check condition <xsl:if test="test-xpath">
  -e or --elem <name>      - print out element <xsl:element name="name">
  -a or --attr <name>      - add attribute <xsl:attribute name="name">
  -b or --break            - break nesting
  -s or --sort op xpath    - sort in order (used after -m) where
  op is X:Y:Z,
      X is A - for order="ascending"
      X is D - for order="descending"
      Y is N - for data-type="numeric"
      Y is T - for data-type="text"
      Z is U - for case-order="upper-first"
      Z is L - for case-order="lower-first"
```

```
There can be multiple --match, --copy-of, --value-of, etc options
in a single template. The effect of applying command line templates
can be illustrated with the following XSLT analogue

xml sel -t -c "xpath0" -m "xpath1" -m "xpath2" -v "xpath3" \
        -t -m "xpath4" -c "xpath5"

is equivalent to applying the following XSLT

<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <xsl:call-template name="t1"/>
  <xsl:call-template name="t2"/>
</xsl:template>
<xsl:template name="t1">
  <xsl:copy-of select="xpath0"/>
  <xsl:for-each select="xpath1">
    <xsl:for-each select="xpath2">
      <xsl:value-of select="xpath3"/>
    </xsl:for-each>
  </xsl:for-each>
</xsl:template>
<xsl:template name="t2">
  <xsl:for-each select="xpath4">
    <xsl:copy-of select="xpath5"/>
  </xsl:for-each>
</xsl:template>
</xsl:stylesheet>

XMLStarlet is a command line toolkit to query/edit/check/transform
XML documents (for more information see http://xmlstar.sourceforge.net/)

Current implementation uses libxslt from GNOME codebase as XSLT processor
(see http://xmlsoft.org/ for more details)
```

’select’ option allows you basically avoid writting XSLT stylesheet to perform some queries on XML documents. I.e. various combinations of command line parameters will let you to generate XSLT stylesheet and apply in to XML documents with a single command line. Very often you do not really care what XSLT was created for you ’select’ command, but in those cases when you do; you can always use -C or --comp switch which will let you see exactly which XSLT is applied to your input.

’select’ option supports many EXSLT functions in XPath expressions.

Here are few examples which will help to understand how ’xml select’ works:

EXAMPLE:

Count elements matching XPath expression:

```
xml sel -t -v "count(/xml/table/rec/numField)" table.xml
```

Input (table.xml):

```
<xml>
  <table>
    <rec id="1">
      <numField>123</numField>
      <stringField>String Value</stringField>
    </rec>
    <rec id="2">
      <numField>346</numField>
      <stringField>Text Value</stringField>
    </rec>
    <rec id="3">
      <numField>-23</numField>
      <stringField>stringValue</stringField>
    </rec>
  </table>
</xml>
```

Output:

```
3
```

Let's take a close look what it did internally. For that we will use '-C' option

```
$ xml sel -C -t -v "count(/xml/table/rec/numField)"
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:exslt="http://exslt.org/common"
 xmlns:math="http://exslt.org/math"
 xmlns:date="http://exslt.org/dates-and-times"
 xmlns:func="http://exslt.org/functions"
 xmlns:set="http://exslt.org/sets"
 xmlns:str="http://exslt.org/strings"
 xmlns:dyn="http://exslt.org/dynamic"
 xmlns:saxon="http://icl.com/saxon"
 xmlns:xalanredirect="org.apache.xalan.xslt.extensions.Redirect"
 xmlns:xt="http://www.jclark.com/xt"
 xmlns:libxslt="http://xmlsoft.org/XSLT/namespace"
 xmlns:test="http://xmlsoft.org/XSLT/"
 extension-element-prefixes="exslt math date func set str dyn saxon xalanredirect xt libxslt test"
 exclude-result-prefixes="math str">
<xsl:output omit-xml-declaration="yes" indent="no"/>
<xsl:param name="inputFile">-</xsl:param>
<xsl:template match="/">
```

```
  <xsl:call-template name="t1"/>
</xsl:template>
<xsl:template name="t1">
  <xsl:value-of select="count(/xml/table/rec/numField)"/>
</xsl:template>
</xsl:stylesheet>
```

Ignoring some XSLT stuff to make it brief:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output omit-xml-declaration="yes" indent="no"/>
<xsl:param name="inputFile">-</xsl:param>
<xsl:template match="/">
  <xsl:call-template name="t1"/>
</xsl:template>
<xsl:template name="t1">
  <xsl:value-of select="count(/xml/table/rec/numField)"/>
</xsl:template>
</xsl:stylesheet>
```

Every -t option is mapped into XSLT template. Options after '-t' are mapped into XSLT elements:

- -v to <xsl:value-of>
- -c to <xsl:copy-of>
- -e to <xsl:element>
- -a to <xsl:attribute>
- -s to <xsl:sort>
- -m to <xsl:for-each>
- -i to <xsl:if>
- and so on

By default subsequent options (for instance: -m) will result in nested corresponding XSLT elements (<xsl:for-each> for '-m'). To break this nesting you would have to put '-b' or '--break' after first '-m'.

Below are few more examples:

EXAMPLE

Count all nodes in XML documents. Print input name and node count after it.

```
xml sel -t -f -o " " -v "count(//node())" xml/table.xml xml/tab-obj.xml
```

Output:

```
xml/table.xml 32
xml/tab-obj.xml 41
```

EXAMPLE

Find XML files matching XPath expression (containing 'object' element)

```
xml sel -t -m //object -f xml/table.xml xml/tab-obj.xml
```

Result output:

```
xml/tab-obj.xml
```

EXAMPLE

Calculate EXSLT (XSLT extentions) XPath value

```
echo "<x/>" | xml sel -t -v "math:abs(-1000)"
```

Result output:

```
1000
```

EXAMPLE

Adding elements and attributes using command line 'xml sel'

```
echo "<x/>" | xml sel -t -m / -e xml -e child -a data -o value
```

Result Output:

```
<xml><child data="value"/></xml>
```

EXAMPLE

Query XML document and produce sorted text table

```
xml sel -T -t -m /xml/table/rec -s D:N:- "@id" -v "concat(@id,'|',numField,'|',stringField)" -n xml/table.xml
```

Result Output:

```
3|-23|stringValue
2|346|Text Value
1|123|String Value
```

Equivalent stylesheet

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output omit-xml-declaration="yes" indent="no" method="text"/>
<xsl:param name="inputFile">-</xsl:param>
<xsl:template match="/">
  <xsl:call-template name="t1"/>
</xsl:template>
<xsl:template name="t1">
  <xsl:for-each select="/xml/table/rec">
    <xsl:sort order="descending" data-type="number" case-order="upper-first" select="@id"/>
    <xsl:value-of select="concat(@id,'|',numField,'|',stringField)"/>
    <xsl:value-of select="'&#10;'"/>
  </xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```

EXAMPLE

Predefine namespaces for XPath expressions

```
xml sel -N xsql=urn:oracle-xsql -t -v /xsql:query xsql/jobserve.xsql
```

Input (xsql/jobserve.xsql)

```
$ cat xsql/jobserve.xsql
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="jobserve.xsl"?>
<xsql:query connection="jobs" xmlns:xsql="urn:oracle-xsql" max-rows="5">
  SELECT substr(title,1,26) short_title, title, location, skills
  FROM job
  WHERE UPPER(title) LIKE '%ORACLE%'
  ORDER BY first_posted DESC
</xsql:query>
```

Result output

```
  SELECT substr(title,1,26) short_title, title, location, skills
  FROM job
  WHERE UPPER(title) LIKE '%ORACLE%'
  ORDER BY first_posted DESC
```

EXAMPLE

Print structure of XML element using xml sel (advanced XPath expressions and xml sel command usage)

```
xml sel -T -t -m '//*' \
-m 'ancestor-or-self::*' -v 'name()' -i 'not(position()=last())' -o . -b -b -n \
xml/structure.xml
```

Input (xml/structure.xml)

```
<a1>
  <a11>
    <a111>
      <a1111/>
    </a111>
    <a112>
      <a1121/>
    </a112>
  </a11>
  <a12/>
  <a13>
    <a131/>
  </a13>
</a1>
```

Result Output:

```
a1
a1.a11
a1.a11.a111
a1.a11.a111.a1111
a1.a11.a112
a1.a11.a112.a1121
a1.a12
a1.a13
a1.a13.a131
```

This example is a good demonstration of nesting control. Here is corresponding XSLT:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output omit-xml-declaration="yes" indent="no" method="text"/>
<xsl:param name="inputFile">-</xsl:param>
<xsl:template match="/">
  <xsl:call-template name="t1"/>
</xsl:template>
<xsl:template name="t1">
  <xsl:for-each select="//*">
    <xsl:for-each select="ancestor-or-self::*">
      <xsl:value-of select="name()"/>
      <xsl:if test="not(position()=last())">
```

```
        <xsl:value-of select="'.'"/>
      </xsl:if>
    </xsl:for-each>
    <xsl:value-of select="'&#10;'"/>
  </xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```

EXAMPLE

Print all links of xhtml document

```
xml sel --net --html -T -t -m "//*[local-name()='a']" \
   -o 'NAME: ' -v "translate(. , '&#10;', ' ')" -n \
   -o 'LINK: ' -v @href -n -n \
   http://xmlstar.sourceforge.net/
```

Sample output

```
NAME: XmlStarlet SourceForge Site
LINK: http://sourceforge.net/projects/xmlstar/

NAME: XmlStarlet CVS Source
LINK: http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/xmlstar/

NAME: XmlStarlet on Freshmeat.Net
LINK: http://freshmeat.net/projects/xmlstarlet/

NAME: XMLStarlet Sourceforge forums
LINK: http://sourceforge.net/forum/?group_id=66612

NAME: XMLStarlet mailing list
LINK: http://lists.sourceforge.net/lists/listinfo/xmlstar-devel
```

4.2. Transforming XML documents

Here is synopsis for 'xml tr' command:

```
XMLStarlet Toolkit: Transform XML document(s) using XSLT
Usage: xml tr [<options>] <xsl-file> {-p|-s <name>=<value>} [ <xml-file-or-uri> ... ]
where
   <xsl-file>      - main XSLT stylesheet for transformation
   <xml-file>      - input XML document file name (stdin is used if missing)
```

```
   <name>=<value>  - name and value of the parameter passed to XSLT processor
   -p              - parameter is XPATH expression ("'string'" to quote string)
   -s              - parameter is a string literal
<options> are:
   --omit-decl     - omit xml declaration <?xml version="1.0"?>
   --show-ext      - show list of extensions
   --val           - allow validate against DTDs or schemas
   --net           - allow fetch DTDs or entities over network
   --xinclude      - do XInclude processing on document input
   --maxdepth val  - increase the maximum depth
   --html          - input document(s) is(are) in HTML format
   --catalogs      - use SGML catalogs from $SGML_CATALOG_FILES
                     otherwise XML catalogs starting from
                     file:///etc/xml/catalog are activated by default

XMLStarlet is a command line toolkit to query/edit/check/transform
XML documents (for more information see http://xmlstar.sourceforge.net/)

Current implementation uses libxslt from GNOME codebase as XSLT processor
(see http://xmlsoft.org/ for more details)
```

EXAMPLE:

```
# Transform passing parameters to XSLT stylesheet
xml tr xsl/param1.xsl -p Count='count(/xml/table/rec)' -s Text="Count=" xml/table.xml
```

Input xsl/params1.xsl

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="text"/>
<xsl:param name="Text"/>
<xsl:param name="Count"/>
<xsl:template match="/">
  <xsl:call-template name="t1"/>
</xsl:template>
<xsl:template name="t1">
  <xsl:for-each select="/xml">
    <xsl:value-of select="$Text"/>
    <xsl:value-of select="$Count"/>
    <xsl:value-of select="'&#10;'"/>
  </xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```

Output

```
Count=3
```

4.3. Editing XML documents

Here is the synopsis for 'xml ed' command:

```
XMLStarlet Toolkit: Edit XML document(s)
Usage: xml ed <global-options> {<action>} [ <xml-file-or-uri> ... ]
where
  <global-options>  - global options for editing
  <xml-file-or-uri> - input XML document file name/uri (stdin is used if missing)

<global-options> are:
  -P (or --pf)        - preserve original formatting
  -S (or --ps)        - preserve non-significant spaces
  -O (or --omit-decl) - omit XML declaration (<?xml ...?>)
  -N <name>=<value>   - predefine namespaces (name without 'xmlns:')
                        ex: xsql=urn:oracle-xsql
                        Multiple -N options are allowed.
                        -N options must be last global options.
  --help or -h        - display help

where <action>
   -d or --delete <xpath>
   -i or --insert <xpath> -t (--type) elem|text|attr -n <name> -v (--value) <value>
   -a or --append <xpath> -t (--type) elem|text|attr -n <name> -v (--value) <value>
   -s or --subnode <xpath> -t (--type) elem|text|attr -n <name> -v (--value) <value>
   -m or --move <xpath1> <xpath2>
   -r or --rename <xpath1> -v <new-name>
   -u or --update <xpath> -v (--value) <value>
                          -x (--expr) <xpath> (-x is not implemented yet)

XMLStarlet is a command line toolkit to query/edit/check/transform
XML documents (for more information see http://xmlstar.sourceforge.net/)
```

EXAMPLE:

```
# Delete elements matching XPath expression
xml ed -d "/xml/table/rec[@id='2']" xml/table.xml
```

Input

```
<xml>
  <table>
    <rec id="1">
      <numField>123</numField>
      <stringField>String Value</stringField>
    </rec>
    <rec id="2">
      <numField>346</numField>
      <stringField>Text Value</stringField>
    </rec>
    <rec id="3">
```

```
      <numField>-23</numField>
      <stringField>stringValue</stringField>
    </rec>
  </table>
</xml>
```

Output

```
<xml>
  <table>
    <rec id="1">
      <numField>123</numField>
      <stringField>String Value</stringField>
    </rec>
    <rec id="3">
      <numField>-23</numField>
      <stringField>stringValue</stringField>
    </rec>
  </table>
</xml>
```

EXAMPLE

```
# Move element node
echo '<x id="1"><a/><b/></x>' | xml ed -m "//b" "//a"
```

Output

```
<x id="1">
  <a>
    <b/>
  </a>
</x>
```

EXAMPLE

```
# Rename attributes
xml ed -r "//*/@id" -v ID xml/tab-obj.xml
```

Output:

```
<xml>
  <table>
    <rec ID="1">
      <numField>123</numField>
      <stringField>String Value</stringField>
      <object name="Obj1">
        <property name="size">10</property>
        <property name="type">Data</property>
      </object>
```

```
    </rec>
    <rec ID="2">
      <numField>346</numField>
      <stringField>Text Value</stringField>
    </rec>
    <rec ID="3">
      <numField>-23</numField>
      <stringField>stringValue</stringField>
    </rec>
  </table>
</xml>
```

EXAMPLE

```
# Rename elements
xml ed -r "/xml/table/rec" -v record xml/tab-obj.xml
```

Output:

```
<xml>
  <table>
    <record id="1">
      <numField>123</numField>
      <stringField>String Value</stringField>
      <object name="Obj1">
        <property name="size">10</property>
        <property name="type">Data</property>
      </object>
    </record>
    <record id="2">
      <numField>346</numField>
      <stringField>Text Value</stringField>
    </record>
    <record id="3">
      <numField>-23</numField>
      <stringField>stringValue</stringField>
    </record>
  </table>
</xml>
```

EXAMPLE

```
# Update value of an attribute
xml ed -u "/xml/table/rec[@id=3]/@id" -v 5 xml/tab-obj.xml
```

Output:

```
<xml>
  <table>
    <rec id="1">
```

```
      <numField>123</numField>
      <stringField>String Value</stringField>
      <object name="Obj1">
        <property name="size">10</property>
        <property name="type">Data</property>
      </object>
    </rec>
    <rec id="2">
      <numField>346</numField>
      <stringField>Text Value</stringField>
    </rec>
    <rec id="5">
      <numField>-23</numField>
      <stringField>stringValue</stringField>
    </rec>
  </table>
</xml>
```

EXAMPLE

```
# Update value of an element
xml ed -u "/xml/table/rec[@id=1]/numField" -v 0 xml/tab-obj.xml
```

Output:

```
<xml>
  <table>
    <rec id="1">
      <numField>0</numField>
      <stringField>String Value</stringField>
      <object name="Obj1">
        <property name="size">10</property>
        <property name="type">Data</property>
      </object>
    </rec>
    <rec id="2">
      <numField>346</numField>
      <stringField>Text Value</stringField>
    </rec>
    <rec id="3">
      <numField>-23</numField>
      <stringField>stringValue</stringField>
    </rec>
  </table>
</xml>
```

4.4. Validating XML documents

Here is synopsis for 'xml val' command:

```
XMLStarlet Toolkit: Validate XML document(s)
Usage: xml val <options> [ <xml-file-or-uri> ... ]
where <options>
   -w or --well-formed        - validate well-formedness only (default)
   -d or --dtd <dtd-file>     - validate against DTD
   -s or --xsd <xsd-file>     - validate against XSD schema
   -r or --relaxng <rng-file> - validate against Relax-NG schema
   -e or --err                - print verbose error messages on stderr
   -b or --list-bad           - list only files which do not validate
   -g or --list-good          - list only files which validate
   -q or --quiet              - do not list files (return result code only)

NOTE: XML Schemas are not fully supported yet due to its incomplete
      support in libxml (see http://xmlsoft.org)

XMLStarlet is a command line toolkit to query/edit/check/transform
XML documents (for more information see http://xmlstar.sourceforge.net/)
```

EXAMPLE

```
# Validate XML document against DTD
xml val --dtd dtd/table.dtd xml/tab-obj.xml >/dev/null 2>&1; echo $?
```

Output:

```
1
```

EXAMPLE

```
# Validate against XSD schema
xml val -b -s xsd/table.xsd xml/table.xml xml/tab-obj.xml 2>/dev/null; echo $?
```

Output:

```
xml/tab-obj.xml
1
```

4.5. Formatting XML documents

Here is synopsis for 'xml fo' command:

```
XMLStarlet Toolkit: Format XML document
```

```
Usage: xml fo [<options>] <xml-file>
where <options> are
   -n or --noindent            - do not indent
   -t or --indent-tab          - indent output with tabulation
   -s or --indent-spaces <num> - indent output with <num> spaces
   -o or --omit-decl           - omit xml declaration <?xml version="1.0"?>
   -R or --recover             - try to recover what is parsable
   -D or --dropdtd             - remove the DOCTYPE of the input docs
   -C or --nocdata             - replace cdata section with text nodes
   -N or --nsclean             - remove redundant namespace declarations
   -e or --encode <encoding>   - output in the given encoding (utf-8, unicode...)
   -H or --html                - input is HTML
   -h or --help                - print help

XMLStarlet is a command line toolkit to query/edit/check/transform
XML documents (for more information see http://xmlstar.sourceforge.net/)
```

EXAMPLE

```
# Format XML document disabling indent
cat xml/tab-obj.xml | xml fo --noindent
```

Output:

```
<xml>
<table>
<rec id="1">
<numField>123</numField>
<stringField>String Value</stringField>
<object name="Obj1">
<property name="size">10</property>
<property name="type">Data</property>
</object>
</rec>
<rec id="2">
<numField>346</numField>
<stringField>Text Value</stringField>
</rec>
<rec id="3">
<numField>-23</numField>
<stringField>stringValue</stringField>
</rec>
</table>
</xml>
```

EXAMPLE

```
# Recover malformed XML document
xml fo -R xml/malformed.xml 2>/dev/null
```

Input:

```
<test_output>
   <test_name>foo</testname>
   <subtest>...</subtest>
</test_output>
```

Output:

```
<test_output>
  <test_name>foo</test_name>
  <subtest>...</subtest>
</test_output>
```

4.6. Canonicalization of XML documents

Here is synopsis for 'xml c14n' command:

```
XMLStarlet Toolkit: XML canonicalization
Usage: xml c14n <mode> <xml-file> [<xpath-file>] [<inclusive-ns-list>]
where
  <xml-file>   - input XML document file name (stdin is used if '-')
  <xpath-file> - XML file containing XPath expression for
                 c14n XML canonicalization
    Example:
    <?xml version="1.0"?>
    <XPath xmlns:n0="http://a.example.com" xmlns:n1="http://b.example">
    (//. | //@* | //namespace::*)[ancestor-or-self::n1:elem1]
    </XPath>

  <inclusive-ns-list> - the list of inclusive namespace prefixes
                        (only for exclusive canonicalization)
    Example: 'n1 n2'

  <mode> is one of following:
  --with-comments         XML file canonicalization w comments (default)
  --without-comments      XML file canonicalization w/o comments
  --exc-with-comments     Exclusive XML file canonicalization w comments
  --exc-without-comments  Exclusive XML file canonicalization w/o comments

XMLStarlet is a command line toolkit to query/edit/check/transform
XML documents (for more information see http://xmlstar.sourceforge.net/)
```

EXAMPLE

```
# XML canonicalization
xml c14n --with-comments ../examples/xml/structure.xml ; echo $?
```

Input ../examples/xml/structure.xml

```
<a1>
  <a11>
    <a111>
      <a1111/>
    </a111>
    <a112>
      <a1121/>
    </a112>
  </a11>
  <a12/>
  <a13>
    <a131/>
  </a13>
</a1>
```

Output

```
<a1>
  <a11>
    <a111>
      <a1111></a1111>
    </a111>
    <a112>
      <a1121></a1121>
    </a112>
  </a11>
  <a12></a12>
  <a13>
    <a131></a131>
  </a13>
</a1>
0
```

EXAMPLE

```
# XML exclusive canonicalization
xml c14n --exc-with-comments ../examples/xml/c14n.xml ../examples/xml/c14n.xpath
```

Input

```
../examples/xml/c14n.xml

<n0:pdu xmlns:n0='http://a.example.com'>
<n1:elem1 xmlns:n1='http://b.example'>
content
</n1:elem1>
</n0:pdu>

../examples/xml/c14n.xpath
```

```
<XPath xmlns:n0="http://a.example.com" xmlns:n1="http://b.example">
(//. | //@* | //namespace::*)[ancestor-or-self::n1:elem1]
</XPath>
```

Output

```
<n1:elem1 xmlns:n1="http://b.example">
content
</n1:elem1>
```

4.7. XML and PYX format

Here is synopsis for 'xml pyx' command:

```
XMLStarlet Toolkit: Convert XML into PYX format (based on ESIS - ISO 8879)
Usage: xml pyx {<xml-file>}
where
   <xml-file> - input XML document file name (stdin is used if missing)

The PYX format is a line-oriented representation of
XML documents that is derived from the SGML ESIS format.
(see ESIS - ISO 8879 Element Structure Information Set spec,
ISO/IEC JTC1/SC18/WG8 N931 (ESIS))

A non-validating, ESIS generating tool originally developed for
pyxie project (see http://pyxie.sourceforge.net/)
ESIS Generation by Sean Mc Grath http://www.digitome.com/sean.html

XMLStarlet is a command line toolkit to query/edit/check/transform
XML documents (for more information see http://xmlstar.sourceforge.net/)
```

EXAMPLE

```
xml pyx input.xml
```

Input (input.xml)

```
<books>
<book type='hardback'>
<title>Atlas Shrugged</title>
<author>Ayn Rand</author>
<isbn id='1'>0525934189</isbn>
</book>
</books>
```

Output

```
(books
-\n
(book
Atype hardback
-\n
(title
-Atlas Shrugged
)title
-\n
(author
-Ayn Rand
)author
-\n
(isbn
Aid 1
-0525934189
)isbn
-\n
)book
-\n
)books
```

PYX is a line oriented format for XML files which can be helpful (and very efficient) when used in combination with regular line oriented UNIX command such as sed, grep, awk.

'depyx' option is used for conversion back from PYX into XML.

EXAMPLE (Delete all attributes). This should work really fast for very large XML documents.

```
xml pyx input.xml | grep -v  "^A" | xml depyx
```

Output

```
<books>
<book>
<title>Atlas Shrugged</title>
<author>Ayn Rand</author>
<isbn>0525934189</isbn>
</book>
</books>
```

Here is an article which describes how PYX format can be used to grep XML.
http://www-106.ibm.com/developerworks/xml/library/x-matters17.html (???)

4.8. Escape/Unescape special XML characters

Here is synopsis for 'xml esc' command:

```
xml esc --help
XMLStarlet Toolkit: Escape special XML characters
Usage: xml esc [<options>] [<string>]
where <options> are
   --help      - print usage
   (TODO: more to be added in future)
if <string> is missing stdin is used instead.

XMLStarlet is a command line toolkit to query/edit/check/transform
XML documents (for more information see http://xmlstar.sourceforge.net/)
```

EXAMPLE

```
# Escape special XML characters
cat xml/structure.xml | xml esc
```

Input

```
<a1>
  <a11>
    <a111>
      <a1111/>
    </a111>
    <a112>
      <a1121/>
    </a112>
  </a11>
  <a12/>
  <a13>
    <a131/>
  </a13>
</a1>
```

Output

```
&lt;a1&gt;
  &lt;a11&gt;
    &lt;a111&gt;
      &lt;a1111/&gt;
    &lt;/a111&gt;
    &lt;a112&gt;
      &lt;a1121/&gt;
    &lt;/a112&gt;
  &lt;/a11&gt;
  &lt;a12/&gt;
  &lt;a13&gt;
```

```
    &lt;a131/&gt;
  &lt;/a13&gt;
&lt;/a1&gt;
```

4.9. List directory as XML

Here is synopsis for 'xml ls' command:

```
XMLStarlet Toolkit: List directory as XML
Usage: xml ls
Lists current directory in XML format.

XMLStarlet is a command line toolkit to query/edit/check/transform
XML documents (for more information see http://xmlstar.sourceforge.net/)
```

EXAMPLE

```
xml ls
```

Output

```
<xml>
<d a="rwxr-xr-x" acc="2004.02.13 00:06:03" mod="2004.02.13 00:06:00" sz="4096"  n="."/>
<d a="rwxr-xr-x" acc="2004.02.12 23:54:35" mod="2004.02.13 00:00:09" sz="4096"  n=".."/>
<f a="rw-r--r--" acc="2004.02.12 23:54:58" mod="2004.02.12 23:54:58" sz="0"     n="resume.xml"/>
<f a="rw-r--r--" acc="2004.02.12 23:54:58" mod="2004.02.12 23:54:58" sz="0"     n="resume-2004.xml"/>
<d a="rwxr-xr-x" acc="2004.02.13 00:04:52" mod="2004.02.13 00:04:52" sz="4096"  n="old-resumes"/>
</xml>
```

Chapter 5. Common problems

5.1. Namespaces and default namespace

One of the commonly asked questions about XmlStarlet ’select’ or ’edit’ options is: "Why nothing matched for my XPath expression which seems right to me?". Common cause of these problems is not properly defining a namespace for XPath. This chapter will show several examples to illustrate these issues you might encounter.

For example the following XHTML document has a default namespace declaration

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Query Page</title>
<meta http-equiv="Content-Style-Type" content="text/css" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
<meta name="robots" content="noindex,nofollow" />
</head>
<body>
...
</body>
</html>
```

And the following (initially looking correct) query to print all links

```
xml sel -t -m "//a" -c . -n
```

would return nothing. The issue with this query is that it is not addressing element <a> in the right namespace. XPath requires all namespaces used in XPath expression be defined. So for declared namespace <html xmlns="http://www.w3.org/1999/xhtml"> in input XML, you have to do same for XPath (or XSLT). There is another important detail: namespace equivalency is determined not by namespace prefix, but by URI. See query below, which would return expected result

```
xml sel -N x="http://www.w3.org/1999/xhtml" -t -m "//x:a" -c . -n
```

Example of deleting namespace declarations.

Delete namespace declarations and all elements from non default namespace from the following XML document:

Input (file ns2.xml)

```
<doc xmlns="http://www.a.com/xyz" xmlns:ns="http://www.c.com/xyz">
  <A>test</A>
  <B>
    <ns:C>xyz</ns:C>
  </B>
</doc>
```

Command:

```
xml ed -N N="http://www.c.com/xyz" -d '//N:*' ns2.xml | sed -e 's/ xmlns.*=".*"//g'
```

Output

```
<doc>
  <A>test</A>
  <B/>
</doc>
```

5.2. Special characters

Sometimes issues appear with handling of special characters, where 'special' means in XML sence as well as in 'shell' terms. Examples below should clear at least some of the confusions.

You should not forget about the fact that your command lines are executed by shell and shell does substitutions of its special characters too. So for example, one may ask:

"Why does the following query return nothing?"

```
echo '<X name="foo">EEE</X>' | xml sel -t -m /X[@name='foo'] -v .
```

The answer lies in the way shell substitues 'foo', which simply becomes foo before the command is run. So the correct way to write that would be

```
echo '<X name="foo">EEE</X>' | xml sel -t -m "/X[@name='foo']" -v .
```

Another example involves XML special characters. Question: How to search for ' in text nodes?

The following should help

```
xml sel -t -m "//line[contains(text(),&quot;'&quot;)]" -c .
```

5.3. Sorting

Let's take a look at XSLT produced by the following 'xml sel' command:

```
# Query XML document and produce sorted text table
xml sel -T -t -m /xml/table/rec -s D:N:- "@id" -v "concat(@id,'|',numField,'|',stringField)" -n xml/table.xml
```

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output omit-xml-declaration="yes" indent="no" method="text"/>
<xsl:param name="inputFile">-</xsl:param>
<xsl:template match="/">
  <xsl:call-template name="t1"/>
</xsl:template>
<xsl:template name="t1">
  <xsl:for-each select="/xml/table/rec">
    <xsl:sort order="descending" data-type="number" case-order="upper-first" select="@id"/>
    <xsl:value-of select="concat(@id,'|',numField,'|',stringField)"/>
    <xsl:value-of select="'&#10;'"/>
  </xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```

-s option of 'xml sel' command controls 'order', 'data-type', and 'case-order' attributes of <xsl:sort/> element .

5.4. Validation

Many questions are asked about XSD (XML schema) validation. Well, XmlStarlet relies on libxml2 which has incomplete support for XML schemas. Untill it is done in libxml2 it will not be in XmlStarlet.