XPT, a XML general-purpose preprocessing tool

Bram Latupeirissa

Version 0.1.1 (CVS xpt.xml 1.42 2014/04/13)

2014-04-16

Table of Contents

1. Introduction

Somewhere in 2010 I had a project where DocBook XML documents needed to be enriched with data coming from various, non-XML sources. The idea was to have scripting insert DocBook elements into a DocBook template. There were two options, first write a program to convert non-XML data into DocBook XML elements and second write a more generic program to insert DocBook XML elements into the original source. XPT (xml-xpt) does this. It replaces XML processing instructions with target xpt in an XML input file.

This way a library can be setup with (user-) predefined functions.

This document is also available in Acrobat pdf format at http://xml-xpt.sourceforge.net/docu/xpt.pdf.

2. Functionality

This paragraph presents a global picture about what functionality xpt delivers and how and where to use it. Some use cases.

A DocBook use-case

DocBook is a XML vocabulary (http://en.wikipedia.org/wiki/DocBook) for writing books and articles. Often a make tool-chain is used, where a DocBook XML source is transformed into a certain format, e.g. HTML or PDF. A place where, with every version of a document, manual action is required: releaseinfo and pubdate.

3. XPT command-line options

The following command-line options are recognised by xpt:

`xpt` [-h] [-k] [-c *`cmd`*] [-F] [-i *`infile`*] [-I *`dir`*] [-n] [-o *`outfile`*] [-s *`cmd`*] [-t *`stylesheet`*] [-T *`pi-name`*] [-f *`all-func`*] [-P *`init-cmd`*] [-D *`name=[=[value]]`*] [-x] [-v] [-V]

-c *`cmd`*	enter command mode. Currently the following values are allowed:
	ppi — process processing-instructions. this is the default value.
	version — show version information. same as the `-V`.
-D *`name`*[=*`value`*]	define variable *`name`* optionally with value *`value`*.
-f *`all-func`*	register a single function for all selected XML elements. Usually needs -P option to get function installed.
-F	use fancy XML output format for better readability.
-h	show some help.
-i *`infile`*	read XML from *`infile`*. If not specified, standard input is assumed. If *`infile`* is explicitly specified as `-` (dash), the standard input is read.
-I *`dir`*	append *`dir`* to list search directories of xpt and embedded Python engine.
-k	keep temporary files
-n	do not execute external commands
-o *`outfile`*	send output to *`output`*. **Note** needs extra explanation out how the output is created. read the code.
-P *`init-cmd`*	issue *`init-cmd`* command before transforming selected elements. Most often a module is loaded using this option. This option may be used more than once.

-s *`cmd`*	use XPath *`cmd`* in while creating output.
-t *`stylesheet`*	use XSLT *`stylesheet`* in output generation
-T *`pi-name`*	change name of processing-instruction target to *`pi-name`*. Default is `xpt`.
-x	Show debug output. This option may be used more than once to increase the verbosity.
-v	Be verbose.
-V	Show version information.

4. Startup and environment

4.1. environment

Environment variables that are taken into account:

HOME	used for locating the start-up file, see Section 4.2, “start-up file”.
XPT_DEBUG	enables debugging output (see option -x). The value of the variable sets the debug verbosity level. This option allows debugging of code activated during the reading of the start-up file, see Section 4.2, “start-up file”.
TMPDIR	determines the location for placement of temporary files.

4.2. start-up file

When xpt starts it checks for the existance of a start-up file.

5. Command syntax

Syntax of the XPT processing instructions. Output redirection (string, XML node, XML nodeset). Comment

6. Built-in commands

This paragraph describes the built-in commands that come standard with xpt.

6.1. XML bindings

This paragraph describes the built-in commands that can be used with processing-instructions in the input XML stream. User-defined commands can be created from the embedded Python engine (http://www.python.org/). Most commands can have arguments. There are two types of arguments: positional and named arguments. EXPLAIN. Then there is quoting: unquoted, single-quoted and double-quoted. Optionally glueing those parts together. Note that built-in commands (currently?) have no named arguments, only positional. See the section on variable substitution Section 6.2, “Variable substitution”.

6.1.1. code

`code` NEWLINE *`CODE`*

Insert a block of executable code. Note that a `NEWLINE` must be after the `code` keyword. From that point the text is passed to the Python interpreter. The node wil be removed from the output.

Example 1. example built-in code usage

```
<example-code>
<?xpt code
import libxml2
import xpt

def myFunction(nd,arg1):
    return "myFunction: nd="+nd.name+" arg1='"+arg1+"'"

xpt.registerfunction("myFunction")?></example-code>

<example-call>
<?xpt myFunction "an argument"?>
</example-call>
```

In the example above, the first processing-instruction using the **code** statement registers a new xpt function called **myFunction** which is associated with the Python function with the same name. After evaluation it will be removed from the output. In the second processing-instruction the xpt function is evaluated. It will be replaced with the result returned by the Python function, in example Example 1, “example built-in code usage” that is the string `"myFunction: nd=xpt arg1='an argument'"`.

In Python a module variable `__xpt_code_xmlNode__` is set to the libXML2 node containing the current code statement. The variable is removed when the execution of the module stops.

Note

All Python functions that are called from xpt have at least one argument, i.e. a xmlNode pointer referencing the XML element from where the actual call originates. In this is the **example-call** element.

See also Section 6.1.6, “loadmodule”. xpt.debug

6.1.2. dumpdoc

`__dumpdoc` {STRING}

Debugging command which displays the current document to the standard error output, using `STRING` as a message. Note that debugging output must be enabled by using option -x or environment variable.

6.1.3. dumpvars

`__dumpvars` {STRING}

Debugging command which shows a list of currently defined variables to the standard error output, using `STRING` as a message. Note that debugging output must be enabled by using option -x or environment variable.

6.1.4. echo

`echo` {STRING}

Evaluate `STRING` and replace the processing-instruction by the string representation.

See also Section 6.1.8, “get”.

6.1.5. error

`error` {*`STRING`*}

Evaluate STRING, report it on the standard error channel and stop processing of the input. No output file will be written. The xpt process returns an exit-code of 1 to the parent process.

6.1.6. loadmodule

`loadmodule` *`MODULE`*

Load Python module *`MODULE`*. The node wil be removed from the output.

Example 2. example built-in loadmodule usage

```
<?xpt showNextPI?>
```

See also ???.

6.1.7. python

`python` {*`python-expression`*}

Replace the processing-instruction with the result of the *`python-expression`*.

6.1.8. get

`get` {*`NAME`*}

Replace the processing-instruction with the value of variable *`NAME`*.

Example 3. example built-in get usage

```
<?xpt get __hostname__?>
```

Evaluation of the processing-instruction above gives: pluto

Here See also Section 6.1.4, “echo”.

6.1.9. select

`select` [*`NAME`*] {*`XPATH`*}

Replace the processing-instruction with the result of the *`XPATH`* expression. If *`NAME`* is given the result is saved in a variable and the processing-instruction is removed from the output.

6.1.10. set

`set` *`NAME`* *`VALUE`*

Create or replace variable *`NAME`* with value *`VALUE`*. The node wil be removed from the output.

Example 4. example built-in set usage

```
<?xpt set var1 "**** This is var1 ****"?>
```

Evaluation of the processing-instruction above gives:

In the example above, *`var1`* will have value `**** This is var1 ****`.

6.1.11. setmatch

CASE-like expression

6.1.12. shell

`shell` CMD

Replace the processing-instruction with the value of the shell command *`CMD`*.

6.1.13. warning

`warning` MSG

Evaluate string *`MSG`* and send it to the standard error output. The processing-instruction is removed from the output.

6.2. Variable substitution

Variable substitution: replace `${NAME}` constructions with the value associated the the variable `NAME`. There are shell-like quoting rules: single and double quotes have different effect.

6.3. Python bindings

This paragraph describes the XPT bindings for the Python environment.

6.3.1. xpt.debug

import xpt

```
None xpt.debug();
```

```
message;
```

Issue a message on the debug output channel. See -x.

6.3.2. xpt.dumpvars

import xpt

```
None xpt.dumpvars();
```

```
message;
```

Write list of XPT variables on the debug output channel. See -x and Section 6.1.3, “dumpvars”.

6.3.3. xpt.error

import xpt

```
None xpt.error();
```

```
message;
```

Write a *`message`* to the standard error channel and stop processing of the input. See Section 6.1.5, “error”.

6.3.4. xpt.get

Get value of a variable. An exception is raised if the variable does not exist.

6.3.5. xpt.registerfunction

import xpt

```
None xpt.registerfunction(, );
```

```
name;
func;
```

Registers xpt function *`name`* which is associated with the Python function *`func`*. If this second parameter is not specified, it's name is assumed to be identical to *`name`* prefixed with the name of the module from which the `registerfunction` is called.

6.3.6. xpt.set

6.3.7. xpt.warning

import xpt

```
None xpt.warning();
```

`message;`

Write a *`message`* to the standard error channel. See Section 6.1.13, “warning”.

6.4. XPT library

Xpt comes with extra functionality through a library of functions.

6.4.1. xpt.docbook5 - DocBook 5 functions

6.4.2. xpt.csv - CSV functions

6.4.3. Extending xpt

There are several way of extending xpt. First by pointing xpt where the Python modules are using the `-I` option. A second way is by extending the library, giving you an opportunity to enter the XPT Hall of Fame.

7. XPT development

This chapter describes the various parts of the build-system.

The sources are hosted on sourceforge (http://sourceforge.net/projects/xml-xpt/).

7.1. Sources and build-system

- make use of configure to make the software more portable. currently the Makefile needs gmake.
- document dependencies: libxml2-2.7.7 (xml2-config, perl-libxml-mm.c) + sources, perl-5.8. python26-2.6.5_1 perl-libxml-mm.c) + sources, perl-5.8. python-config: python-dev

7.1.1. Ubuntu build environment

Xpt is developed under Ubuntu. Binary packages are automatically build via https://launchpad.net/xml-xpt and currently made avaiable through a personal packaging archive (PPA).

7.1.2. FreeBSD build environment

Xpt is known to work used FreeBSD 8.1.

7.1.3. MAC OS X build environment

Xpt has been tested under MAC OS X 10.5.6.

7.1.4. MS Windows build environment

Xpt has been tested under MS Windows XP.

7.2. Documentation

Part of the documentation, especially the manpages, are extracted from the source code. The tools for extraction and converting it to DocBook 5 refentry elements, are part of the standard library that comes along with the xtp tool itself. See also xpt.embedded_manpage.embed_manpage.

1. Ideas and things todo

Here's a list of ideas and things that need to be done.

1. better/more robust implementation of xpt command parsing (see `dopi()` in `xpt.c`). goal is to make a more complete syntax in the processing-instruction available, eg. allowing a semi-colon to separate multiple commands, where currently multiple processing-instructions are needed. In this way the interdependence of a group of PIs is fixed and clearly visible.

2. maybe it's possible to use namespaced xpt elements making use of a DTD to check correctness of a document. Eg. use **<xpt:date format="%Y-%m-%d">** instead of **<?xpt date "%Y-%m-%d"?>**

   Investigate the possibility to validate the function calls make from xpt. Through processing instructions there is hardly any checking possible. The checking done by xpt is just syntax checking. The second level of checking could be done by the called function, but that's left to the implementor. It would be nice to have DTD features on this level. Maybe this could be accomplished by using namespaced xpt elements as shown above.

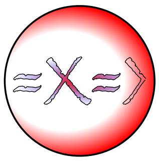
   Maybe it's a nice feature to have validation on processing-instructions. This, may need extending the XML specification.

   Maybe using libxslt extentions is a way of doing that, but i'm not certain. See http://xmlsoft.org/XSLT/extensions.html.

3. xpt function to read in external data with conversion and selection capability. reading not ony from the filesystem but also from commands and ports. watch out to make this a god-function.

4. sorting tables and lists.

5. define a logo for the project. Using Inkscape the following logo is currently proposed. The idea behind it is that data is flowing into the program from the left, gets processed and than is flowing out on the right side. Note that the right arrow is in fact another Xcharacter from which the right side is removed.

2. Pointers

- docbook-util, xmlmind, XMLStarlet Command Line XML Toolkit (http://xmlstar.sourceforge.net/)

- pandoc, ea sdf, xmlif ()
- asciidoc, http://www.methods.co.nz/asciidoc/
- xproc (http://xproc.org/). “XProc: An XML Pipeline Language”
- lxml (http://lxml.de/). “lxml - XML and HTML with Python”. Especially the functionality for extending XSLT and XPath (http://lxml.de/extensions.html).

3. Manpages

This section contains manual pages which are automatically extracted from the sources. More information about extracting manpages can be found in xpt.embedded_manpage.embed_manpage.

Name

doc_source — TEST test doc_source....

Synopsis

```
c3=tbl.defColumn('col2',default=1)
```

Description

Class to create some thing

See Also

xpt(1), python(1), xpt.docbook.table().

Name

xpt.embedded_manpage.embed_manpage — extract embedded man-pages from source code

Synopsis

```
import xpt.embedded_manpage

xml = embed_manpage(nd,directory)
```

Description

Embed_manpage extracts Unix man-pages from sources. Why embedded documentation: See eg. http://www.fossil-scm.org/index.html/doc/tip/www/embeddeddoc.wiki advocacy + ideas http://www.digitalmars.com/d/1.0/ddoc.html usage, otherlinks eg. doxygen http://www.python.org/community/sigs/current/doc-sig/otherlangs/ http://docs.python.org/devguide/documenting.html http://sphinx-doc.org/markup/inline.html Arguments: directory is a colon-separated list of directories which is searched for files holding embedded man-pages VARIABLES Variables are set using the .set command and can be used in the text with the special %{NAME} construction. Standard variables are __file__, __linenr__ and __null__. COMMANDS Command start with a dot on the first position on a line. The following commands are defined: .set NAME VALUE set variable NAME to value VALUE .setre REGEX NAME VALUE search text to match regex REGEX. REGEX must have one or more remembered values which can be used in the NAME and/or VALUE part as $1, $2, etc. Eg.: .setre (RETURN)\s+(VALUE) name_$1 With value: ($2) will set variable name_RETURN to hold value 'With value: (VALUE)'. Note that the text search starts right after the .setre command.

Return value

A Python list containing LibXML2 nodes in DocBook 5 refentry format

Errors

See Also

xpt(1), python(1), xpt.docbook.docbook5_refentry().

Name

xpt.table.table — xpt table class

Synopsis

```
import xpt.table

tbl = xpt.table.table(caption,rowsonly,table_name,tablegroup_name,tablebody_name,row_na
c1 = tbl.defColumn('nr.',type='auto')
c2 = tbl.defColumn('col1',format="(-%s-)")
c3 = tbl.defColumn('col2',default=1)
```

Description

Class to create a table in a XPT

Errors

See Also

xpt(1), python(1), xpt.docbook.table().

Name

version — extract CVS version information from string

Synopsis

```
<?xpt loadmodule xpt.docbook5 ?>
<?xpt version "$Id: document.xml,v 1.2 2011-04-03 04:51:11 coder Exp $"?>
```

Description

The version function extracts CVS information from the supplied string which is assumed to be the value of the CVS Id keyword. Extracted information is saved in the XPT variables __cvs_filename__, __cvs_version__, __cvs_date__, __cvs_time__, __cvs_user__.

Return value

Nothing

Errors

In case of an error an exception is raised.

See Also

xpt(1).

Name

xpt.debug — write out XPT debug message

Synopsis

```
import xpt
xpt.debug("Hello debugger")
```

Description

Write out a message on the debug output channel. If xpt(1) was compiled without debugging code, this command has no effect.

Return value

Nothing

Errors

An exception is raised if incorrect parameters are passed.

See Also

xpt(1), python(1), xpt.warning(), xpt.message(), warning().

Name

xpt.message — write out XPT regular information message

Synopsis

```
import xpt
xpt.message("Hello debugger")
```

Description

Write out a message on the standard output channel.

Return value

Nothing

Errors

An exception is raised if incorrect parameters are passed.

See Also

xpt(1), python(1), xpt.debug(), xpt.warning(), message().

Name

xpt.warning — write out XPT warning message

Synopsis

```
import xpt
xpt.warning("Hello debugger")
```

Description

Write out a message on the warning output channel.

Return value

Nothing

Errors

An exception is raised if incorrect parameters are passed.

See Also

xpt(1), python(1), xpt.debug(), xpt.message().

Name

xpt.error — issue an XPT error

Synopsis

```
import xpt
xpt.error("This is a grave error")
```

Description

Write out a message on the error output channel and terminate the xpt process via the builtin error() funtions.

Return value

Nothing

Errors

An exception is raised if incorrect parameters are passed.

See Also

xpt(1), python(1), xpt.warning(), xpt.message(), error().

Name

xpt.get — get a XPT variable

Synopsis

```
import xpt
xpt.get(NAME)
```

Description

Get the value of an xpt maintained variable called NAME.

Return value

The value of the variable called NAME.

Errors

An exception is raised if incorrect parameters are passed or NAME does not exist.

See Also

xpt(1), python(1), xpt.get().

Name

xpt.set — set a XPT variable

Synopsis

```
import xpt
xpt.set(NAME, VALUE)
```

Description

Set the xpt maintained variable NAME to the value VALUE.

Return value

Nothing

Errors

An exception is raised if incorrect parameters are passed.

See Also

xpt(1), python(1), xpt.get().

Name

xpt.dumpvars — Write out list of xpt variables

Synopsis

```
import xpt
xpt.dumpvars(msg)
```

Description

Write out the list of variables maintained by xpt.

Return value

Nothing

Errors

An exception is raised if incorrect parameters are passed.

See Also

xpt(1), python(1).

Name

xpt.registerfunction — register a Python function to XPT

Synopsis

```
import xpt
xpt.registerfunction(xptname,pythonname)
```

Description

Register a Python function to xpt. Inside xpt the function is named xptname, the name of the Python function is pythonname. The latter argument is optional. If not specified it will be computed using the specified xptname.

Return value

Nothing

Errors

An exception is raised if incorrect parameters are passed.

See Also

xpt(1), python(1).

Name

xpt.acc_string2boolean — argument check and convert string to boolean

Synopsis

```
import xpt
xpt.acc_string2boolean(ARG, default=False)
```

Description

Check and convert ARG to a boolean value. ARG is assumed to be a string passed to a XPT registered Python function. Note that passing a boolean value via ARG just returns it's value. Is ARG is None, the default value is returned. Valid string values for True are "yes", "true" and "1". All other valuesa are assumed False.

Return value

A boolean value.

Errors

An exception is raised if incorrect parameters are passed.

See Also

xpt(1), python(1), xpt.acc_string2integer().

Name

xpt.acc_string2integer — argument check and convert string to integer

Synopsis

```
import xpt
xpt.acc_string2integer(ARG, default=0)
```

Description

Check and convert ARG to an integer value. ARG is assumed to be a string passed to a XPT registered Python function. Note that passing a integer value via ARG just returns it's value. Is ARG is None, the default value is returned.

Return value

An integer value.

Errors

An exception is raised if incorrect parameters are passed.

See Also

xpt(1), python(1), xpt.acc_string2integer().

Name

loadmodule — load a Python module

Synopsis

```
<?xpt loadmodule MODULENAME?>
```

Description

The xpt builtin loadmodule command, loads a Python module MODULENAME. The processing instruction is removed from the input.

Return value

None

See Also

xpt(1), python(1), xpt.docbook.docbook5_refentry().

Name

code — insert Python code

Synopsis

```
<?xpt code
# Python code inserted here
?>
```

Description

The xpt builtin loadmodule command, loads a Python module MODULENAME. The processing instruction is removed from the input.

Return value

None

Errors

In the example below, a Python function is defined inside a XML processing-instruction and then called from an other processing-instruction. The latter will get replaced by the value returned by the Python function. ::code-start:: <?xpt code import libxml2 import xpt * def myFunction(nd,arg1): return "myFunction: nd="+nd.name+" arg1="'"+arg1+"'"" xpt.registerfunction("myFunction") ?> <example-code> <?xpt myFunction "an argument"?> </example-call> ::code-end::

See Also

xpt(1), python(1), xpt.registerfunction().

Name

shell — start external command

Synopsis

```
<?xpt shell COMMAND?>
```

Description

The xpt builtin shell command, executes external command COMMAND and replaces the processing instruction with the output of the command

Return value

None

See Also

xpt(1), python(1), system(3).

Name

warning — issue a warning

Synopsis

```
<?xpt warning MESSAGE?>
```

Description

Evaluate string MESSAGE and send it to the standard error output. The processing-instruction is removed from the output.

Return value

None

See Also

xpt(1), python(1), error().

Name

error — issue a error

Synopsis

```
<?xpt error MESSAGE ?>
```

Description

Evaluate MESSAGE, report it on the standard error channel and stop processing of the input. No output file will be written. The xpt process returns an exit-code of 1 to the parent process.

Return value

None

See Also

xpt(1), python(1), warning().

Name

unlink — parent - remove the parent node

Synopsis

```
<?xpt unlink-parent ?>
```

Description

Remove the parent node and all of it's children.

Return value

None

See Also

xpt(1), python(1).

Name

xpt — a XML preprocessor

Synopsis

```
xpt [-h] [-k] [-c cmd] [-F] [-i infile] [-I dir] [-n] [-o outfile] [-s cmd] [-t styleshe
```

Description

XPT is a XML preprocessor. * Python: if a Python function returns a none standard object, i.e. a string, an integer, a LibXML2 node, etc. xpt tries to convert it to something of a known type. In case of an instance of a certain class, a method called xpt_repr() in that class is assumed to handle this case. This method should do this conversion. Note that the returned result of xpt_repr() is allowed to be an instance of yet an other class. Even lists of instances are handled.

Options

-c cmd enter command mode. Currently the following values are allowed: ppi process processing-instructions. this is the default value. version show version information. same as the -V. -D name[=value] define variable name optionally with value value. -f all-func register a single function for all selected XML elements. Usually needs -P option to get function installed. -F use fancy XML output format for better readability. -h show some help. -i infile read XML from infile. If not specified, standard input is assumed. If infile is explicitly specified as - (dash), the standard input is read. -I dir append dir to list search directories of xpt and embedded Python engine. -k keep temporary files -n do not execute external commands -o outfile send output to output. Note: need extra explanation how the output is created. For now read the code. -P init-cmd issue init-cmd command before transforming selected elements. Most often a module is loaded using this option. This option may be used more than once. -s cmd use XPath cmd in while creating output. -t stylesheet use XSLT stylesheet in output generation. -T pi-name change name of processing-instruction target to pi-name. Default is xpt. -x Show debug output. This option may be used more than once to increase the verbosity. -v Be verbose. -V Show version information.

Return value

None

See Also

python(1)..