

TclXML: The Next Generation

Steve Ball

Steve.Ball@zveno.com

Zveno

<http://www.zveno.com>

Abstract

TclXML is a family of packages that together provide comprehensive support for creating and processing XML documents using the Tcl scripting language. The package family is comprised of TclXML (for SAX-style streamed parsing), TclDOM (for in-memory tree manipulation) and TclXSLT (for transformations).

New developments in each of the packages are discussed, as well as new XML-based applications that make use of the TclXML framework. The most important aspect of these developments is its impact on application development and integration.

1. Introduction

The TclXML project [1] encompasses three closely related, but separate packages - TclXML, TclDOM [2] and TclXSLT. Together these packages provide a comprehensive toolkit for accessing and manipulating data in XML documents [3]. From their early development these packages have concentrated on the design of APIs to allow application developers to write code that is independent of the package's implementation. TclXML and TclDOM both have a Tcl implementation, meaning that no Tcl extensions are required to gain their functionality, albeit with slow run-time performance. Recent development has added fast implementations to these packages.

The major addition to the TclXML project has been Tcl wrappers for the Gnome libxml2 and libxslt libraries [4]. libxml2 is a C library that features XML parsing, DTD validation (a priori and posteriori), an in-memory tree representation and support for XPath [5]. The library's run-time performance is very fast, with efficient memory usage. libxml2 also has wrappers for Python, Perl, Ruby as well as other languages and is also available as a PHP module.

libxslt is a C library for performing XSL transformations (XSLT) [6]. Regarded as perhaps the fastest currently available XSLT processor, libxslt also features the ability to pre-compile and cache stylesheets, as well as an interface to XSLT's extension mechanism that allows third-party extensions to be implemented in C. There are wrappers for the

libxslt library for Python, Perl and Ruby.

Since both libraries are in widespread use they are stable, well-tested and well supported.

Tcl applications that process XML documents often need to navigate the document tree, or otherwise identify some part of a document to process. The W3C XPath language has been designed for this purpose. XPath is part of the XSLT and XML-Query languages, and DOM Level 3 [7] also has support for XPath. Addressing a document component using XPath is quite succinct, as the following examples show:

```
/book
/book/chapter
//title
//section[sectioninfo]/para
```

Example 1. XPath Expressions

The first path selects the document element `book`. The second path selects all `chapter` elements within a `book`. The next path selects all `title` elements in the document. The last path selects all `para` elements in a `section` element, as long as that `section` element also contains a `sectioninfo` element.

XPath expressions can also be used to compute values. For example,

```
count(value)
returns the number of value elements that are children of the
current node.
```

TclXML now has an XPath parser. TclDOM/tcl has partial XPath support and TclDOM/libxml2 has full XPath support.

2. TclXML

The TclXML package provides an API for SAX-like [8] XML document parsing. That is, it is a streamed, event-based XML parser. It also provides implementations of a number of

XML parsers. XML parser implementations have different characteristics and trade-offs: some may concentrate on raw performance, some may provide validation while others may be highly flexible and configurable. For this reason, an application may need to use different parsers during the course of program execution. In order to support this requirement, version 2.0 of the TclXML package has introduced a layered architecture.

Like the SAX specification for Java and Python, the new TclXML V2.0 package provides a generic interface to appli-

cations and a separate interface for a parser implementation. The generic layer passes method calls and configuration options from the application to a parser implementation, and passes data from the parser implementation to the application. TclXML's API for applications hasn't changed very much, so in this paper we will concentrate on the new back-end APIs.

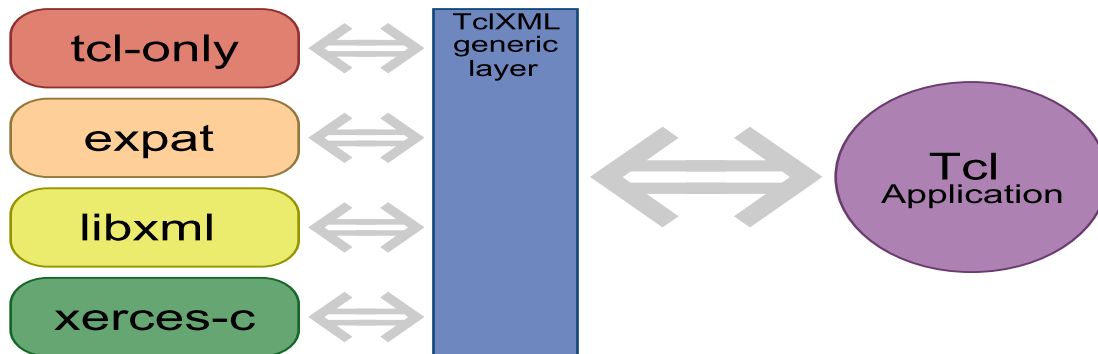


Figure 1. TclXML Architecture

A parser implementation defines a parser class. Parser classes are registered with the TclXML generic layer using the `TclXML_RegisterXMLParser` function (at the C level) or the `::xml::parserclass` command (at the Tcl level). The implementation passes a structure that includes pointers to various functions, such as those to create a parser, destroy a parser, configure a parser as well as parsing data. Whenever the application requests that a parser be created, using the `xml::parser` command, the creation method of the parser class is invoked to create a parser instance. The generic layer also creates a new Tcl command to control the newly created parser instance.

Normally an application will register callback Tcl scripts with a parser instance in order for it to receive data during the parsing operation. These callbacks are registered using the parser instance command. TclXML version 2.0 also introduces equivalent callback registration facilities at the C API level, allowing data to be delivered to the application without any overhead imposed by the Tcl interpreter.

TclXML version 2.0 includes a Tcl implementation of a parser class, as well as a wrapper for the expat library. It is planned to include a wrapper for the libxml2 library in version 3.0 of TclXML.

3. TclDOM

The W3C Document Object Model (DOM) is a language-neutral description of an API for representing an XML (or HTML

or SVG) document as a tree of objects. DOM provides a standard set of objects along with properties and methods for manipulating those objects. Evolution of the DOM standard is described by levels; DOM Level 1 defines basic tree objects, properties and methods for XML, DOM Level 2 adds support for XML Namespaces, as well as other features such as Events, and DOM Level 3 (currently in Working Draft stage) adds support for XPath, as well as other features such as loading and saving.

The TclDOM project is a Tcl language binding for the W3C Document Object Model (DOM). It defines a Tcl API that matches, as closely as is reasonable, the W3C DOM specification. The project also provides a Tcl script package that implements the API. As can only be expected, the script implementation is slow and memory-hungry.

Version 2.0 of TclDOM introduces a number of new features, the most important being a new implementation - a Tcl wrapper for the Gnome libxml2 library. The Gnome libxml2 library is written in C and provides comprehensive set of features for manipulating XML documents in-memory, including SAX parsing, validation and a tree representation. The library is fully conformant with the W3C XML Recommendation as well as the W3C Namespaces in XML Recommendation. libxml2 does *not* provide a DOM API; TclDOM makes use of its native tree functions. A major benefit of using libxml2 is speed; it is regarded as being one of the fastest XML processors available. To illustrate this the following table shows a performance comparison between the Tcl and TclDOM/libxml2 TclDOM implementations. Table 1 shows the time

taken, in microseconds, to run a simple script that creates a number of DOM element nodes. Measurements were performed on a Macintosh 800MHz G4 PowerBook, 512MB RAM, Mac OS 10.1.5.

Although the Tcl wrapper for libxml2 is of great benefit to Tcl applications using the DOM, the original motivation for its development was to support handling libxml2 Document objects in conjunction with TclXSLT (see below). Unfortunately, the functionality of libxml2 does not map directly to TclXML and TclDOM; the one library will require support from both Tcl packages for a full implementation of the feature-set.

| # Elements | Tcl Implementation | TclDOM/libxml2 |
|------------|--------------------|----------------|
| 100 | 1061457 | 48113 |
| 625 | 6334016 | 282178 |
| 2500 | 24883886 | 2680458 |

Table 1. TclDOM Performance Comparison

3.1 TclDOM/libxml2 Design

There were a number of goals for the design of the Tcl wrapper for libxml2:

1. Compatibility with TclXSLT.
2. Compatibility with the Tcl implementation of TclDOM.
3. High-performance.

The first goal has been achieved by making the TclXSLT package use the TclDOM C API.

The second goal has been achieved by ensuring that the TclDOM API was faithfully implemented.

The third goal is achieved in a number of ways. Firstly, where possible internal libxml2 APIs are used to implement functions rather than Tcl APIs. For example, parsing an XML document is handled directly by libxml2. Mainly, the TclDOM/libxml2 package uses the internal representation of Tcl objects to cache Document and Node references.

Unfortunately, it does not appear possible with Tcl to transparently represent an XML document as a DOM tree. This is because DOM trees are mutable objects, and Tcl objects have copy-on-write semantics. That is, the following use case *cannot* be supported:

```
package require dom

set xmldoc {<MyDoc>
<Value>FooBar</Value>
</MyDoc>}

set docElement [dom::document cget $xmldoc \
-documentElement]
dom::document createElement $docElement Value

puts $xmldoc
```

Example 2. Transparent Access to XML Documents

As with other Tcl extensions that must handle mutable objects, the solution is to use object references, or "tokens". TclDOM/libxml2 registers new Tcl object types that correspond to libxml2 xmlDocPtr and xmlNodePtr types. Tcl objects of these types store a reference to the libxml2 object in their internal representation. These objects also have a string representation allocated - the "token". The token string is stored in a Tcl hash table. Tcl object internal representations are easily lost, so in these situations the token is looked up in the hash table and the internal representation restored.

The TclDOM/libxml2 package maintains two global hash tables for all DOM documents. One of the hash tables is indexed by the document's token and the other by the Document object's memory address. Each document also maintains two Tcl hash tables; one for tree nodes and the other for event nodes.

libxml2 Documents and tree nodes are mapped to a single Tcl object. The `_private` field of the Document or Node structure is used to point back to the corresponding Tcl object. In order for the system to be as efficient as possible, the mapping of Document and Node objects to Tcl objects is performed lazily, that is only when required. Initial testing and usage of this scheme appears to indicate that it works well, but there are some issues with it. When a document is destroyed all of the node references in Tcl object internal representations become invalid. If these Tcl objects are subsequently used they will result in program failure due to dangling C pointers, whereas they should result in a catchable Tcl error. To resolve this problem, when a document is to be destroyed the package first iterates through all of the references stored in the document's node and event hash tables, performs a look up of the corresponding Tcl object and resets its internal representation. The (as yet unresolved) problem is that there may be more than one Tcl object with a pointer to the node.

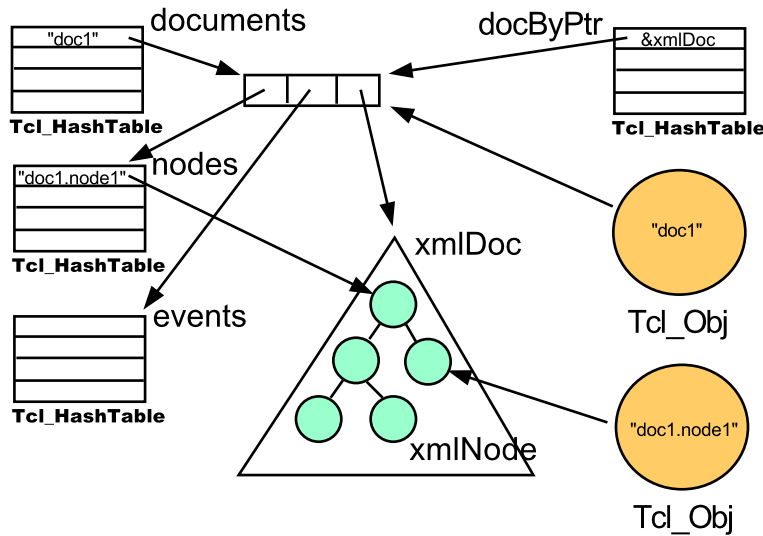


Figure 2. TcIDOM Architecture

There is no limit on the size of a DOM tree, both in terms of data and nodes. This is unlike other tree structures often modelled in Tcl applications, such as the Tk widget hierarchy. While it is unlikely that a Tk GUI would have over a million widgets, it is entirely possible for a DOM tree to have that many or more nodes. For example, this (relatively small) paper has over six hundred element and text nodes, whereas a Tk GUI with over six hundred widgets would be considered to be a moderately complex application. For this reason it is very important to minimise the memory overhead of DOM nodes. TcIDOM/libxml2 adds a hash table entry for each node that is wrapped. In addition, the hash table may become very large, slowing hash entry lookups. For this reason the design of the TcIDOM API has avoided defining node commands, since having many node commands defined may slow down lookup of unrelated Tcl commands. However, creating node commands in a separate Tcl namespace may make this design choice feasible. This approach may be explored in TcIDOM version 3.0.

3.2 XPath

Another major improvement in TcIDOM is support for XPath expressions. Applications using TcIDOM commonly need to navigate the DOM tree, or to select nodes in the tree for processing. This can be quite tedious when using only standard DOM methods and attributes. XPath is a language for addressing parts of an XML document. It is extremely convenient to use for the purpose of document navigation and node selection. TcIDOM provides a facility for selecting new DOM nodes given a context and an XPath expression, which can be either an absolute or relative location path. XPath support is present in both the Tcl and libxml2 implementations, but is incomplete in the former.

```
set doc [dom::parse $xml]
foreach node \
  [dom::selectNode $doc /records/customer] {
  set nodeName [lindex \
    [dom::node selectNode $node name] \
    0]
  set name [dom::node stringValue $nodeName]
  set db [DB_Create -customer $name]
}
```

Example 3. Using XPath

The return value of the `selectNode` method is a static list of node tokens. When the `selectNode` method is used with the `dom::DOMImplementation` command, the root node of the document is the initial context for the location path. This is useful when the location path is an absolute path. When the method is used with the `dom::node` command the given node is the initial context for the location path. This is useful when the location path is a relative path, but absolute paths will also be resolved correctly.

4. TcIXSLT

Another package available from the Gnome libxml project is the libxslt XSLT processor library. libxslt is a fast XSLT engine written in C, fully conformant to the W3C XSLT version 1.0 specification. The libxslt code uses libxml2 to store and manipulate XML documents in memory, as DOM trees. libxslt has been written as a library, and is easily embeddable in an application.

TcIXSLT, a newcomer to the TcIXML family, is a wrapper for the Gnome libxslt library. Unlike the TcIXML and

TclDOM packages, TclXSLT provides only this single XSLT processor implementation and has no provision for a layered architecture to allow alternate implementations. It provides an interface to compile XSL stylesheets and then transform XML documents using those stylesheets. The source XML document and stylesheet document must be supplied to TclXSLT as TclDOM/libxml2 Document objects, thus TclXSLT is dependent upon TclDOM in the same way that libxslt is dependent upon libxml2.

4.1 Transforming XML Documents

The major purpose of the TclXSLT package is to make use of the libxslt library to transform XML documents with an XSL stylesheet. To do this, an XSL stylesheet must first be compiled using the `::xslt::compile` Tcl command. This command requires a TclDOM/libxml2 Document object as its argument (since XSL stylesheets are, in fact, XML documents). Internally, the `compile` command copies the Document object and then invokes the libxslt stylesheet compiler upon the copied Document. Copying the document is necessary because libxslt makes use of the `_private` member of the `xmlDoc` and `xmlNode` structures.

```
set styledoc [dom::libxml2::parse $styleXML]
set stylesheet [xslt::compile $styledoc]
```

Example 4. Compiling An XSL Stylesheet

The return result of the `xslt::compile` command is a token for the compiled stylesheet. A side effect of the command is to create a new Tcl command, called the `stylesheet` command, with the same name as the returned token. This new command may be used to access and manipulate the compiled stylesheet. The `stylesheet` command accepts the `transform`, `cget` and `configure` methods.

`transform` is the most important method of the `stylesheet` command. This method transforms the TclDOM/libxml2 Document object supplied as an argument and returns the result document as a new TclDOM/libxml2 document. The result document may be used via the TclDOM/libxml2 package, just like any other DOM tree created by the package. For example, the result document may become the source document of another transformation, or may even be compiled and used as a stylesheet. Thus TclXSLT allows efficient pipelining of XSL transformations and caching of compiled stylesheets, as well as caching of source and result documents.

```
set sourcedoc [dom::libxml2::parse $sourceXML]
set styledoc [dom::libxml2::parse $styleXML]
set stylesheet [xslt::compile $styledoc]
set resultdoc [$stylesheet transform \
    $sourcedoc]
set resultXML [dom::libxml2::serialize \
    $resultdoc]
```

Example 5. Performing A Transformation

4.2 XSLT Extensions

The XSLT standard provides a means for XSLT processor implementations to extend the number of functions and elements available for use by the XSL stylesheet. Extension functions and elements allow the XSLT processor implementation to provide features not present in the XSLT specification. XML Namespaces are used to identify which functions and elements belong to an extension. The libxslt library provides an API for defining extensions, and includes an implementation of the EXSLT extension set.

```
<xsl:stylesheet version='1.0'
  xmlns:xsl=
'http://www.w3.org/1999/XSL/Transform'
  xmlns:ext='http://exslt.org/common'
  extension-element-prefixes='ext'>

  <xsl:template match='Example'>
    <ext:document href='example.xml'>
      This is an extension.
    </ext:document>
  </xsl:template>

</xsl:stylesheet>
```

Example 6. An XSLT Extension

TclXSLT features a binding to the libxslt extension mechanism that allows extensions to be implemented as Tcl scripts. The extension mechanism provided by TclXSLT associates the XML Namespace used by the extension to a Tcl namespace that implements the extension. The `::xslt::extension` command is used to manage these associations.

```
xslt:extension add \
  http://www.zveno.com/resources ::resources
```

Example 7. The `xslt::extension` Command

The `add` method of the `::xslt::extension` command creates an association between an XML Namespace and a Tcl namespace. There are also methods to `remove` and `list` these associations.

The TclXSLT package's extension mechanism registers all Tcl procedures in the given Tcl namespace as either XSLT extension functions or XSLT extension elements. If the procedure accepts a variable number of arguments, then it is registered as an extension function. Otherwise it is registered as an extension element. TclXSLT uses Tcl introspection (ie, the `info args` command) to determine what formal parameters a procedure accepts. Thus it is not possible to directly register

Tcl built-in commands. Registration occurs when an XSLT stylesheet is initialised using the `::xslt::compile` command.

```
namespace eval ::resources {
    proc exists {resource args} {
        return [info exists $resource]
    }
}
```

Example 8. Registering An Extension Function

The example above would cause the `exists` extension function to be registered in the `http://www.zveno.com/resources` extension namespace.

```
<xsl:stylesheet version='1.0'
  xmlns:xsl=
'http://www.w3.org/1999/XSL/Transform'
  xmlns:resource='http://www.zveno.com/resources'
  extension-element-prefixes='resource'>

  <xsl:template match='Foo'>
    <xsl:value-of
      select=
        'resource:exists("/home/steve/doc")' />
  </xsl:template>

</xsl:stylesheet>
```

Example 9. Using an Extension

The stylesheet in the example above declares the extension using the `extension-element-prefixes` attribute. Now the XML namespace `http://www.zveno.com/resources` is associated with the Tcl namespace `::resources` and the procedure `::resources::exists` is registered as an extension function. The current implementation of TcIXSLT, version 2.2, converts the arguments to an extension functions to a string value, appends these values to the procedure name and then evaluates the resulting command line, as shown in the following example.

```
::resources::exists /home/steve/doc
```

Example 10. Evaluated Command

The return value of the procedure is returned as an XPath string object. If the procedure results in an error, then an error object is returned.

The current implementation of TcIXSLT does not support extension elements. Future implementations of TcIXSLT will preserve the data type of arguments. Extension elements will be supported by passing a single argument to the Tcl procedure which will be the DOM node of the extension element. The data type of the return value will also be preserved, with DOM nodes being passed as node objects and lists of DOM nodes passed as a nodeset.

5. Implications

Whereas Tcl is a very-high-level, general-purpose scripting language, XSLT is a very-high-level, special-purpose transformation language. XSLT has been deliberately designed to not be suitable for handling all types of programming tasks. However, in an environment where data is increasingly made available as, or within, XML documents and where the inputs to other processes use XML it becomes very attractive to use XSLT in place of a scripting language. The fact that XSLT is a W3C standard, and supported by major software vendors on all major computing platforms, reinforces the choice of XSLT as the language for implementation of business, application and presentation logic.

So there is a desire to use XSLT as much as possible to implement an application. A major limitation to realising this engineering solution is that XSLT has few facilities for interfacing with the "real world". The solution to this problem is to define XSLT extensions that provide the "glue" to the external environment. TcIXSLT's extension mechanism provides an ideal way to overcome this problem, since Tcl has excellent interfaces to the various resources of a system and implementing XSLT extensions is much easier than using C.

Another impediment to making use of XSLT is that the standard does not specify the environment in which transformations take place (nor should it). That is, an application needs a framework to run the XSLT processor: marshalling the source documents, supplying the parameters and disposing of the result document. TcIXSLT also provides a solution, because an XSLT processor can be embedded in a Tcl application: a GUI tool, such as `xmltool`, or a Web server, such as `tcIhttpd` or `mod_dtcl`.

It is interesting to note how TcIXSLT makes XSLT an embeddable library for XML processing for the Tcl language. TcIXSLT allows Tcl to invoke an XSL stylesheet, as well as allowing the XSL stylesheet to call back into the application's Tcl code. This is completely analogous to the way in which Tcl is an embeddable library for C. Previously, one might have considered software engineering based on scripting languages to be two-tiered, with application logic implemented using the scripting language (Tcl, Python, Perl) and low-level components implemented using the system language (C, C++, Java). Now, software engineering using TcIXSLT is three-tiered, with application logic implemented as an XSL stylesheet, lower-level components (accessible as XSLT extensions) implemented using the scripting language and finally lowest-level components implemented using the system language. Note that other scripting languages also have wrappers for `libxslt` (or other XSLT processors), so this phenomenon is by no means peculiar to Tcl.

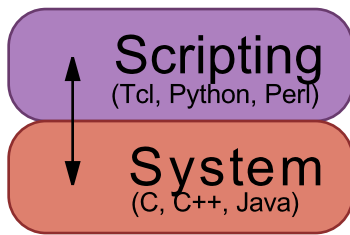


Figure 3. Two Tiered Application

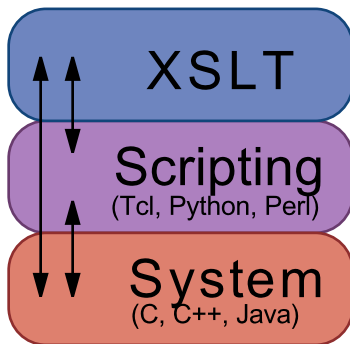


Figure 4. Three Tiered Application

6. Applications

Now that a high-performance, efficient framework is available for processing XML documents a number of applications are being developed to leverage the TclXML family of packages. These applications include a document authoring tool and entry-level content management systems. xmltool is a simple GUI application that drives XML parsers, XSL transformers and comparators. Simple CMS is a more sophisticated Web-based application that also drives parsing and transformation processes, but as part of a workflow. waX Me Lyrical (waX) is an information authoring tool that uses XML as its underlying save format and employs a DOM tree for editing.

6.1 waX

The primary design goal of the waX Me Lyrical application (waX) is to enable a document author to create and maintain information, independent of the eventual use of that application - whether it be for print publishing, Web publishing or other purposes. To achieve this goal waX uses XML as the document medium. A specific non-goal is for waX to be an XML editor. Initially, waX is primarily aimed at supporting

authoring of DocBook documents.

Under-the-hood, waX is, in fact, a DOM-based editor. DOM Events are used to synchronise the various GUI components.

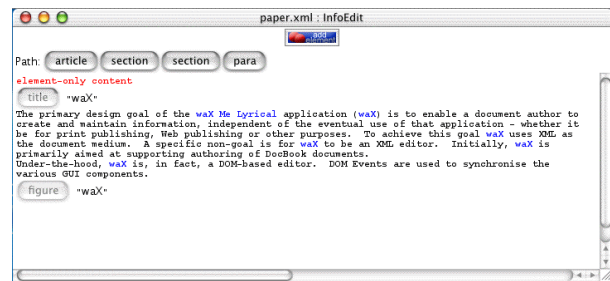


Figure 5. waX

6.2 xmltool

xmltool (fancier, more marketable name pending) is a very simple tool for performing XML processing tasks upon XML documents. These tasks include checking well-formedness, checking validity, transforming documents and comparing documents.

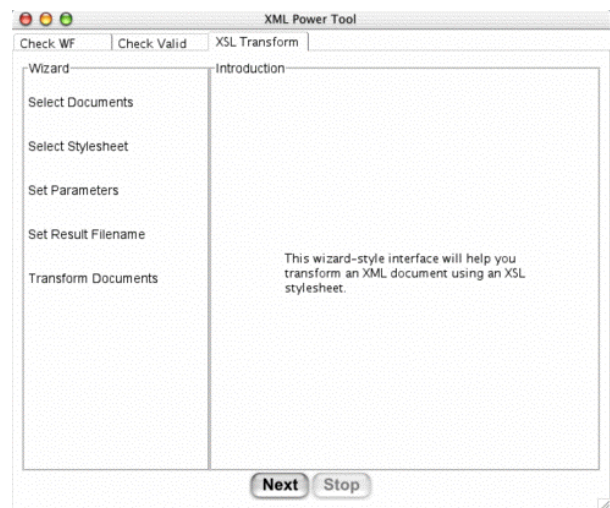


Figure 6. xmltool

6.3 Simple CMS

The Simple Content Management System (SCMS) takes xmltool to the next level by adding processing workflows. Workflows are specified using the XML Pipeline Definition Language, a W3C Note. The implementation of SCMS is mostly achieved using XSLT. Currently, SCMS uses xmltool as the host application interface but other interfaces are planned, in

particular a Web server application.

The XML Pipeline Definition Language [9] is an XML schema for specifying how resources are processed. The rules defining the processing workflow are contained in a "pipeline document". Software that interprets a pipeline document is known as a "pipeline controller". At the heart of SCMS is a pipeline controller.

For example, this paper, written in DocBook, must be transformed into an XSL Formatting Objects (XSL-FO) document and then the XSL-FO document is rendered to PDF. A pipeline document that implements this workflow is as follows:

```
<pipeline
xmlns='http://www.w3.org/2002/02/xml-pipeline'>
  <processdef name='transform.p'
    definition='org.xmlpipeline.xslt' />
  <processdef name='format.p'
    definition='org.apache.xml.fop' />

  <process type='format.p'>
    <input name='document' label='paper.fo' />
    <output name='result' label='paper.pdf' />
  </process>

  <process type='transform.p'>
    <input name='document' label='paper.xml' />
    <input name='stylesheet'
      label='xsl/fo/docbook.xsl' />
    <output name='result' label='paper.fo' />
  </process>
</pipeline>
```

Example 11. A Pipeline Document

It is tempting to think of XML Pipeline as "Make on XML Steroids". The pipeline controller is expected to check whether the target resource exists and if it does whether it is out-of-date with respect to its dependencies. Only if the target does not exist or is out-of-date does the stipulated process need to be performed.

It is not possible to implement a pipeline controller using XSLT version 1.0 alone. In particular, XSLT has no way of testing the existence of a resource and certainly no means by which to find the last modification date of a resource. SCMS solves this problem by providing a set of XSLT extensions, implemented as Tcl procedures, via TclXSLT. The goal of the SCMS project is to find the minimum set of extensions that allow the implementation of a pipeline controller.

7. Conclusion

The TclXML family of packages now has three members: TclXML (for parsing), TclDOM (for tree manipulation) and TclXSLT (for transformations). TclDOM and TclXSLT provide Tcl wrappers for the Gnome libxml2 and libxslt libraries respectively. The Gnome libraries give a significant performance boost to applications using TclDOM, as well as pro-

viding a range of functionality.

Future work on these packages will see TclXML/TclDOM/TclXSLT version 3.0 all providing libxml2/libxslt wrappers. TclXML v3.0 will provide access to libxml2's SAX interface. All of the packages will better support preserving Tcl and XPath data types. The aim is to provide high-performance, and eventually for Tcl to host all aspects of XML processing and to be able to interpose on all operations, including parsing and resolving external entities.

Both the TclDOM and TclXSLT packages value-add to the wrapped Gnome libraries. TclDOM/libxml2 implements the same API as its Tcl implementation, including the DOM Level 2 Event model. TclXSLT incorporates a binding to the libxslt extension mechanism, allowing XSLT extension elements and functions to be implemented as Tcl scripts.

XSLT is a special-purpose, high-level language for handling XML documents. TclXSLT allows XSLT to be used for engineering sophisticated applications, with Tcl providing components and an interface to external resources and legacy applications and data. This represents a radical departure from the traditional use of Tcl as a high-level control language.

8. References

- [1] *TclXML Project*. Steve Ball, et al. <http://tclxml.sourceforge.net/>
- [2] *XML Support For Tcl*. Steve Ball. Proceedings of the 6th Tcl/Tk Conference. September 1998, San Diego CA USA.
- [3] *eXtensible Markup Language, Second Edition*. Tim Bray (Ed). W3C Recommendation [<http://www.w3.org/TR/>] October 2000.
- [4] *libxml2, libxslt libraries*. Daniel Veillard, et al. <http://xmlsoft.org/>
- [5] *XML Path Language (XPath)*. James Clark, et al. W3C Recommendation [<http://www.w3.org/TR/>] November 1999.
- [6] *XSL Transformations (XSLT)*. James Clark. W3C Recommendation [<http://www.w3.org/TR/>] November 1999.
- [7] *Document Object Model (DOM)*. Philippe Le Hegaret, Lauren Wood, Arnaud Le Hors, et al. W3C Recommendation [<http://www.w3.org/TR/>] November 2000.
- [8] *Simple API for XML (SAX)*. David Megginson, XML-DEV mailing list. <http://www.megginson.com/>.
- [9] *XML Pipeline Definition Language*. Norm Walsh, Eve Maler. W3C Note [<http://www.w3.org/TR/>] February 2002.

