The following paper was originally published in the
Proceedings of the Sixth Annual Tcl/Tk Workshop
San Diego, California, September 14–18, 1998

XML Support for Tcl

Steve Ball
Zveno Pty Ltd

For more information about USENIX Association contact:

1. Phone: 510 528-8649
2. FAX: 510 548-5738
3. Email: office@usenix.org
4. WWW URL: http://www.usenix.org/

XML Support For Tcl

Steve Ball
Zveno Pty Ltd
http://www.zveno.com/
Steve.Ball@zveno.com

Abstract

XML is emerging as a significant technology for use on both the World Wide Web and in many other application areas, such as network protocols. Documents written in XML have a rich, hierarchical structure, the document tree. An application which is to process XML documents must be able to access and manipulate the document tree in order to be able to examine and change the structure.

The DOM is a language-independent specification of how an application accesses and manipulates the document structure. TclDOM is a Tcl language binding for the DOM. The TclDOM specification provides a standard API for Tcl applications to process a XML or HTML document.

TclXML is a Tcl package which provides a sample implementation of TclDOM. It provides XML parsers along with the tools needed to create a hierarchical representation of documents which can be conveniently processed by a Tcl script. There are also facilities to check the validity of a document, along with commands to produce document output. TclXML provides a framework for parser and validator modules which allows some or all of the various components to be implemented in an extension language.

Keywords: Tcl, World Wide Web, WWW, XML, DOM, Parsing

A Brief Introduction To XML and DOM

XML

The e**X**tensible **M**arkup **L**anguage, XML [XML], is a Recommendation from the World Wide Web Consortium (W3C) [W3C] which is a significant simplification of SGML, intended to make a subset SGML suitable for use on the Web. XML provides a much more meaningful and rich way in which to represent semantic structure in a document when compared to HTML. With XML, a document author can "call a Spade a <Spade>". That is to say, elements can be given names which are meaningful to the author and convey more of the semantics of the document. These elements are defined using a Document Type Definition (DTD), along with allowed general entities. The DTD includes rules which govern what elements or text are allowed to occur inside each element. In contrast, HTML gives the author a fixed set of tags to use, and those tags have fixed semantics and behaviour.

XML has been designed with the goal of being easier for tool developers to write software for processing documents, so that there will be plentiful applications available for handling XML. The syntax is far more restricted than SGML, with most optional features removed. For example, tag minimisation and omission are not allowed so it is easy to detect the end of an element. Empty elements must be explicitly marked.

There are two levels of conformity for a XML document. At the very least, a XML document must be *well-formed*. A well-formed XML document adheres to all of the syntactic rules of the XML specification. All elements must have an end tag, unless they are empty elements in which case the tag must include a trailing slash, such as
. Element attributes must have a value and the value must be quoted. There may be no occurrances of illegal characters, and so on. XML has been designed so that a program can check that a document is well-formed without having to use the document's DTD.

In addition to being well-formed, a XML document may also be *valid*. A valid XML document conforms to the rules laid out in its associated DTD. This means that all elements have content which is allowed according to the element's definition of its content model in the document's DTD. All attributes used in elements must also be permitted for that element and their value must be of the correct type. All attribute values which are used for identification must be unique, and so on. Validity is a much stronger assertion for a document than well-formedness, but requires access to the document's DTD and more processing time to check the various rules.

XML is internationalised and support for Unicode is

a requirement of XML, so Tcl version 8.1 is well positioned to be used for software applications which handle and process XML documents.

The following document is a small example of a XML document instance:

```
<?xml version="1.0">
<!DOCTYPE paper SYSTEM "paper.dtd">
<paper>
<title>TclXML Example</title>
<abstract>This is an example of a
XML document instance.
</abstract>
<introduction>XML uses the
<acronym><short>DTD</short>
<full>Document Type
Definition</full></acronym>
to define classes of documents.
Such a document is known as a
<definition>document
instance</definition>.
</introduction>
<point>A DTD is used to define the
elements and entities that are
allowed to appear in a document.
Empty elements have a trailing
slash:
<figure image="figure-1.png"/>
</point>
<conclusion>XML is way better than
HTML.</conclusion>
<references></references>
</paper>
```

DOM

The Document Object Model (DOM) [DOM] is a language-neutral specification of a standard Application Programming Interface (API) for both accessing the features and the content of a document and creating or modifying documents. Documents are in the form of a tree and the DOM has methods of accessing properties of the tree nodes. DOM provides a core set of features and specific features may be provided for certain markup languages. In the first instance, DOM is providing a specification of features for XML and HTML. The aim of the DOM is to allow application developers to write software which manipulates a document and to use whichever programming language is suitable for the task. The same underlying constructs will be available in whatever language the developer chooses.

DOM already has language bindings for Java and ECMAscript. Many tools are being written in Java for processing XML, and standard interfaces are starting to emerge for Java components, such as XAPI-J and SAXDOM. A language binding for Tcl is discussed in this paper.

Other Proposed Standards

XML will never replace HTML because XML does not supply any of HTML's semantic features, in particular the elements which provide presentational and/or behavioural semantics such as STRONG, EM, UL and FORM. HTML elements perform a function when they are used in a document, whereas XML elements do not. XML elements only provide syntactic declaration of a document's structure. Of course, this is what makes XML useful for a wider range of applications than HTML.

In order to provide certain semantics for the elements used in a XML document other facilities are required. A number of languages have been proposed for these purposes. The XML Link Language, XLL, provides the hyperlinking model for XML. XLL has been derived from TEI and HyTime and as a result will be a much richer model than HTML's. For example, XLL will allow bidirectional and multi-destination links, as well as allowing link targets to specify and use the structure of the destination resource. In order to render XML documents for viewing or printing a stylesheet language will be used. Either Cascading Style Sheets (CSS) [CSS] will be used or the new proposed XML Stylesheet Language (XSL). XSL will also include scripting.

Tcl Support For XML

Tcl can be very useful for processing and generating XML documents, just as Tcl is useful for handling HTML documents in CGI scripts. There are many applications which may be able to use a scripting language for document processing, both in GUI and non-GUI modes and in standalone, server and client environments.

Since applications will have varying requirements, different supporting libraries may be used to process documents. Some of these may be implemented entirely in Tcl, for cross-platform portability, some may use C or C++ extensions to improve performance and some may interface with Java components. Ideally, there will be a standard programming interface for applications written in Tcl to access the document structure for processing. It is the goal of TclDOM to provide that standard interface, based upon the DOM specification.

XML support in Tcl has two distinct parts. Firstly, TclDOM is a language binding for the DOM.

TclDOM provides an implementation-independent specification of a Tcl API of the DOM for Tcl scripts to access and manipulate XML and HTML documents. Secondly, TclXML is a sample implementation of TclDOM which provides XML parsers and generators.

Zveno is making the TclDOM specification and TclXML freely available. Both the TclDOM specification and the TclXML distribution may be found on the Zveno website [TCLXML]: `http://www.zveno.com/zm.cgi/in-tclxml/`

TclDOM

TclDOM is a Tcl language binding for the DOM. The specification details how to access DOM features using Tcl constructs. The goals for TclDOM are: to provide access to all features of the DOM from a Tcl script, to allow features from other DOM-compliant languages (such as Java) to be used in conjunction with Tcl scripting and finally to provide an interface which uses conventions familiar to Tcl developers.

Access to all of the features of the DOM has the advantage of familiarity to developers who have a previous knowledge of the DOM. The DOM is also a collaborative design effort which ensures that the interface will be comprehensive, allowing sufficient expressiveness of the interface for a general-purpose document scripting library such as TclXML.

Providing all of the DOM features goes a long way towards allowing access to DOM implementations in other languages, because there is then a straightforward mapping between the implementations. It is fair to say that the majority of work in writing XML processors is being done using Java, so an interface to Java is essential for TclXML to leverage this activity. Fortunately, TclBlend and Jacl give easy access to Java classes from Tcl scripts.

TclDOM Interface Design

Tcl and Tk use a number of conventions to make Tcl scripting easier for developers. TclDOM will adopt these conventions in the design of its API so that Tcl developers will find it easy to use. For example, using Tcl commands to manipulate objects and the use of configuration options. The design of the TclDOM API must take into consideration that an application may need to process more than one document concurrently, so a single hierarchical model, such as Tk's widget hierarchy, would be unsuitable. For example, a XML document browser may need to access the document's tree structure while at the same time accessing the document's XSL stylesheet, which is itself a XML document.

Previous Work

Uhler's `html_library`

Stephen Uhler wrote an all-Tcl HTML parser called `html_library` [Uhler95]. This library was sufficiently generalised internally to be easily adapted for parsing XML documents. Indeed, the tokeniser which is part of the Tcl XML parser in TclXML is derived from `html_library`.

`html_library` translates a HTML or XML document into a series of calls to a Tcl procedure. Hence, it is an event-based parser where the application is notified of the start and end of elements. The parser does no checking for well-formedness or validity - all further processing is left to the application. This parser does not provide a good model for TclDOM, since the DOM uses a tree-based view of the document structure.

Plume

Plume [Ball98A] is a general-purpose WWW browser. Plume version 1.0 (never officially released publically) extended the `html_library` parser to build a tree representation of the parsed HTML or XML document. This representation was then presented to the application with the tree structure, in a format known as "XAPI-Tcl", the XML API for Tcl. XAPI-Tcl used a nested Tcl list structure to represent the document tree, with certain commands used to distinguish between elements, character data, processing instructions, and so on. For example, the document example given above would be represented as:

```
parse:pi xml {version 1.0} {}
parse:pi DOCTYPE {SYSTEM paper.dtd}
{}
parse:element paper {} {
 parse:element title {} {
  parse:text {TclXML Example} {} {}
 }
 parse:element abstract {} {
  parse:text {This is an example of
a XML document instance.}
 }
 parse:element introduction {} {
  parse:text {XML uses the }
  parse:element acronym {} {
   parse:element short {} {
    parse:text DTD {} {}
```

```
   }
   parse:element full {} {
    parse:text {Document Type
Definition}
   }
  }
  parse:text {to define classes of
documents. Such a document is known
as a } {} {}
  parse:element definition {} {
   parse:text \
    {document instance} {} {}
  }
  parse:text . {} {}
 }
 parse:element point {} {
  parse:text {A DTD is used to
define the elements and entities
that are allowed to appear in a
document. Empty elements have a
trailing slash: }
  parse:element figure {image
figure-1.png} {}
 }
 parse:element conclusion {} {
  parse:text {XML is way better
than HTML.}
 }
 parse:element references {} {}
}
```

This format may be interpreted as a (nested) Tcl list or evaluated as a Tcl script. Internally the parser, `xml::parse`, manipulates the document as a Tcl list. Since in Tcl version 8.0 (and above) list traversal is relatively fast the parser also returns the parsed data structure as a Tcl list. However, this format is not suitable for evaluation so a command, `xml::cvtscript`, is provided to convert this representation into one which may be evaluated, as shown above.

To facilitate access to the data structure using Tcl list commands, in particular the `foreach` command, dummy arguments are appended to the entries used for text and processing instructions. This allows a construct to be used such as:

```
foreach {type arg1 arg2 arg3}
[xml::parse $MyDocument] {
  switch $type {
    parse:element {
      #   Process element
      # arg1 is the tag name
      # arg2 is the attribute list
      # arg3 is the element content
    }
    parse:text {
      #   Process character data
      # arg1 is the data
      # arg2 and arg3 are unused
    }
    parse:pi {
      #   A processing instruction
      # arg1 is the PI name
      # arg2 is data for the PI
      # arg3 is unused
    }
  }
}
```

As discussed above, an alternative way to process a document is to evaluate the parsed data structure. All arguments are appropriately quoted, so this is a safe operation to perform. In this case, the application defines procedures with the same name as those names used to distinguish the features of the document. By default, these are:

`parse:element`

To denote an element, option `-elementcommand`.

`parse:text`

To denote character data, option `-textcommand`.

`parse:pi`

To denote a processing instruction, option `-picommand`.

`parse:comment`

To denote a comment, option `-commentcommand`.

Different names may be used by specifying options to the `xml::parse` command. This processing method may be used as such:

```
proc parse:element {name attributes
content} {
    eval $content
}
proc parse:text {text unused
unused} {
    puts $text
}

eval [xml::cvtscript [xml::parse
$MyDocument]]
```

The disadvantage of using an explicit Tcl representation for the parsed data structure is that the opportunity is lost to implement the document tree using another language, such as C or Java. Accessing

and manipulating the tree structure may be quite slow using Tcl. Also, it can be difficult to dynamically modify the document structure, for example for document editing purposes, and to navigate the tree from an arbitrary starting point, for example if a tree node is passed as an argument.

CoST

CoST [English96] is a system for processing SGML documents written by Joe English based on James Clark's `sgmls` library. It has been adapted for use with Tk in an application called CoSTWish [PM-R96], and is being used for processing XML documents for the DOM specification [Nicol98].

CoST's API is not directly suitable for use with TclDOM. CoST has the notion of a document node, but nodes are not exposed to the application and there is no equivalent to a DOM "Iterator" for traversing sequences or trees of nodes. Instead, CoST supplies explicit methods for performing iterative operations on document tree nodes, such as `withNode` or `foreachNode`. In addition, CoST does not provide support for creating or modifying documents, nor does it provide access to the document's DTD.

Another difference between DOM and CoST is the notion of a current node. In CoST there is always a node which is current, and operations may be performed upon that node by CoST methods. However, in the DOM no actual node is specified as being current, instead there is a pointer to the position between two nodes. The DOM then allows the node before or after the current pointer to be retrieved. The DOM specifies node positions in this way to allow traversal of a dynamically changing document tree. With the DOM there is no danger of the current node being deleted and so the current node reference becoming invalid.

The TclDOM API

A preliminary API for TclDOM has been designed to satisfy the constraints outlined above. The namespace `dom` shall be reserved for use by the TclDOM package. Layered packages may also be used for supporting specific markup languages. The namespaces `xml` and `html` are initially reserved for XML and HTML respectively.

DOM is defined using the OMG IDL interface specification, a language-neutral definition language. This presents some difficulties for defining the TclDOM specification because IDL is object-oriented, whereas Tcl is not. Common Tcl conventions are used to overcome this problem, by defining class creation commands and object instance commands. TclDOM defines a number of commands within the `dom` namespace which correspond to the IDL interfaces defined in the DOM specification. These commands produce and accept "tokens" as arguments for referring to nodes in the document tree. The DOM implementation may use these tokens to lookup the node in an internal data structure. This allows for efficient implementation in extension languages.

The DOM specification provides a class for accessing a list of nodes and a class to manipulate strings. These are unnecessary in Tcl, as Tcl already has a rich set of primitives for manipulating lists and strings, but these interfaces may be emulated for the sake of compatibility. TclDOM defines methods for retrieving a list of the children of a node, the parent of a node, the list of attributes for an element, and so on. An application may use these methods to traverse the document tree, much like a Tk script traverses the widget hierarchy. Tcl lists are ordered, which is an important property for representing the children of an element node. One potential problem with this approach is that in an application where the document is being dynamically updated the document tree may change after a list of nodes has been generated. However, this may also be seen as an advantage when compared to Plume's nested list approach where changing the document tree can be cumbersome.

Attribute lists are also defined in terms of a Tcl list, but are represented as name/value pairs. This is convenient for use in conjunction with the `array set` Tcl command for accessing attributes via a Tcl array. Attribute lists are unordered, so storing these in a Tcl array is satisfactory.

The command necessary to perform parsing and serialisation a XML document instance is not explicitly provided by the DOM specification. For TclDOM these functions may be made implicit by defining that XML documents are stored as an internal representation of a Tcl Object. In this way, a XML document will be stored initially as a string, but when accessed by a DOM function it will be parsed into the implementation's internal data structure. When the string representation is required the internal structure is serialised. The only problem with this approach is that an implementation would be difficult to write as a pure Tcl script, since the internal representation of a Tcl Object cannot be accessed

from the Tcl script level.

Example

The following is an example of creating a document using TclDOM. This example creates the first few elements of the example given earlier, modifies the document and then saves the XML text in a file. Note that this interface is a preliminary one based on the 20th July 1998 draft version of the DOM Core specification.

```
# Implicitly parse a document
set text [dom::text cget -nodeValue
    [lindex [dom::node children
{<Example>Sample Text</Example>}]
0] ]

# Create a document in memory

set docRoot [dom::document
 createDocumentFragment]
set paper [dom::document
 createElement $docRoot paper]
set title [dom::document
 createElement $paper title]
dom::document createTextNode $title
 {TclXML Example}
dom::document createTextNode
 [dom::document createElement
$paper abstract] {This is an
example of a XML document instance}
set in [dom::document
 createElement $paper introduction]
dom::document createTextNode $in
 {XML uses the }
set acronym [dom::document
 createElement $in acronym]
dom::document createTextNode
 [dom::document createElement
 $acronym short] DTD
dom::document createTextnode
 [dom::document createElement
 $acronym full] {Document Type
 Definition}
dom::document createTextNode $in
 {to define classes of documents.
 Such a document is known as a }
dom::document createTextNode
 [dom::document createElement
 $in definition] {document
 instance}
dom::document createTextNode $in .

# Add an ID attribute to <abstract>

# Search for the element
foreach child [dom::node children
 $paper] {
 if {[dom::node cget -nodeType] ==
"element"} {
   if {[dom::node cget -nodeName]
== "abstract"} {
    dom::element configure -
attributes {ID abc123}
     break
   }
  }
}

# Write out the XML document

set ch [open example.xml w]

# No explicit serialisation
puts $ch $docRoot

close $ch
```

TclXML

TclXML is a Tcl package which provides the facilities needed by a Tcl script to parse XML documents, traverse and manipulate their structures and to generate XML documents. The package provides an implementation of TclDOM (see below) for accessing and manipulating the document structure. It also provides a framework for the various components needed to parse and generate documents, allowing different implementations to be used together in a seamless fashion.

Applications wishing to use TclXML will have different requirements, both in terms of functionality and in terms of performance. Some applications may require an event-based parser, others may require a tree-based representation. An application may only need to check that a document is well-formed, whereas another may need to validate its documents. Figure 1 outlines the various requirements needed.

Application Requirement	Performance	Non-Validating	Validating
Event-based	Critical	TclExpat	Java Parser
	Non-Critical	TclXML	TclXML + Validator
Tree-based	Critical	TclExpat + TclXML Tree Builder	Java Parser
	Non-Critical	TclXML Tree Builder	TclXML + Tree Builder + Validator

Figure 1: XML Processing Requirements of Applications

At present TclXML provides two non-validating XML parsers. A "native" parser written entirely in Tcl and a Tcl interface to James Clark's `expat` [Clark98] parser called TclExpat. Both of these parsers are event based, ie. they produce a stream of "document events", such as the start and end of elements. TclXML also provides a utility to construct a tree based representation of the document, the Tree Builder. This utility uses the event stream produced by either of the parsers to construct the document tree. Finally, TclXML will provide a document validator. The validator will include a DTD parser and will check the complete tree representation of a document for validity according to the document's DTD.

All of these facilities have been exposed to the application developer, since the different parts may be useful for different applications. Some application may process a XML document as a stream, in which an event-based parser is most useful. Other applications may need to traverse the document structure, in which case a tree-based parser is required. Also, the modular construction of the TclXML toolkit, makes it a simple matter to replace components of it with packages written in other languages. TclXML already includes expat, which is written in C, as an example.

An event-based parser which is used within the TclXML framework is configured to issue calls to the TclXML Tree Builder, which then issues calls to the TclDOM module to construct the document tree. A tree-based parser would make calls directly to the TclDOM module. The Tcl application can call the same commands to construct a document in-memory itself.

Compliant Parsers

As can be seen from Figure 2, TclXML aims to allow any compliant parser to be used as a component of the framework. An example might be to replace the built-in Tcl parser with a SAXDOM-compliant parser written in Java. Once a driver is written for the SAXDOM interface, any parser written using that interface can be used with TclXML.

TclExpat

An Tcl extension, called `TclExpat`, has been created to provide an interface to James Clark's `expat` XML parser. This extension has been written for Tcl version 8.0 and 8.1 (alpha2) as a dynamically loadable library. The extension has been tested on Linux, Solaris and HP/UX. Compiling the extension on other platforms supported by Tcl should present no difficulties. An `expat` parser is created using the `expat` command. When a parser is created a corresponding Tcl command is registered to access the parser, in the usual Tcl fashion. `expat` allows callbacks to be registered for various "document events" such as the start of an element, the end of an element, character data, processing instructions, and so on. The purpose of `TclExpat` is to allow Tcl scripts to be invoked as these callbacks. An application may configure a parser with various options to set the callback scripts. Certain callback scripts have arguments appended to them before evaluation, such as the name of an element and its attribute list for the start of an element.

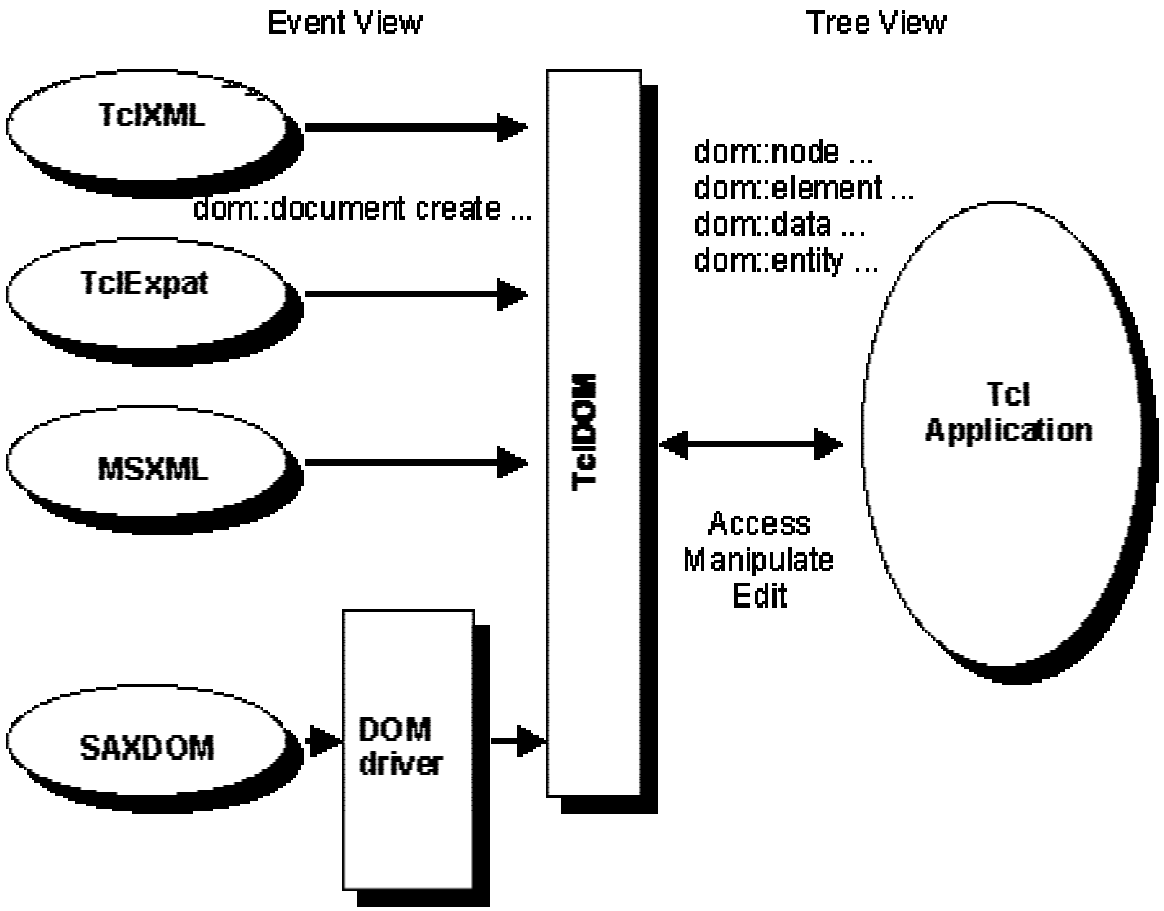

Figure 2: TclXML Modular Structure

The main callback options for the `TclExpat` parser are as follows:

`-elementstartcommand name attributes`

> Invoked when an element starts. Arguments are the name of the element and its attribute list, given as a name-value paired list.

`-elementendcommand name`

> Invoked when an element ends. The name of the element is appended.

`-datacommand data`

> Invoked when character data is encountered. The data is appended.

`-processinginstructioncommand name data`

> Invoked when a processing instruction is encountered. Arguments are the name of the processing instruction and the instruction's data.

`-defaultcommand data`

> This callback is invoked when data is encountered for which there are no other handlers registered.

In addition to supporting straightforward callback scripts, `TclExpat` also allows these scripts to use the `continue` and `break` commands to alter how the document is processed. If the callback script returns a `TCL_CONTINUE` code then callback invocation is suspended until the currently open element is closed. A return code of `TCL_BREAK` causes all further callback invocation to be skipped. If a callback script returns an error condition all further callbacks are skipped and the parser also returns an error condition. `TclExpat` handles aborting the processing of document callbacks even though `expat` itself does not support this feature.

An example of using `TclExpat` is as follows:

```
package require expat

set parser [expat xmlParser]
$parser conf -elementstartcommand
```

```
  CountElements
$parser conf
  -processinginstructioncommand PI

proc CountElements args {
    incr ::count
}

proc PI {name args} {
    if {[regexp
{break|continue|error} $name]} {
        return -code $name "$name
due to processing instruction"
    }
}

set result {}

set count 0
$parser parse {<?xml
version="1.0"?>
<!DOCTYPE Document SYSTEM
"Document.dtd">
<Document>
<Visible>This element is
processed</Visible>
<Interrupted>This element is
skipped <?break?>
<NotCounted/>
</Interrupted><NotCounted/>
</Document>
}
lappend result $count

set count 0
$parser reset
$parser parser {<?xml
version="1.0"?>
<!DOCTYPE Document SYSTEM
"Document.dtd">
<Document>
<Visible>This element is
processed</Visible>
<Interrupted>This element is
skipped <?continue?>
<NotCounted/>
</Interrupted><Counted/>
</Document>
}
lappend result $count

puts $result
```

The output of this script would be `3 4`.

Document Generation

When data stored in an internal data structure is converted to a XML representation, the data is said to be serialised. In this way, a XML document may be generated by a program.

It is proposed that TclDOM-compliant implementations store their documents as the internal representation of a Tcl Object. When a Tcl script requires such an object in a string format, the object will serialise the document, thus generating a XML document. This process may apply to an entire document, a document fragment or an element hierarchy. After serialisation the document is just a plain Unicode string and the application may write it to a file, a network channel and so on.

Performance

Tests have been conducted to investigate the performance of the `TclExpat` extension and the all-Tcl TclXML parser. The `expat` XML parser source code distribution includes a well-formedness checker. The output of this program is a normalised version of the XML input file. A Tcl version of this program, using either the `TclExpat` extension or TclXML parser, which accepts the same inputs and produces the same output, was written for the purpose of running comparison tests. Figure 3 shows the performance comparison.

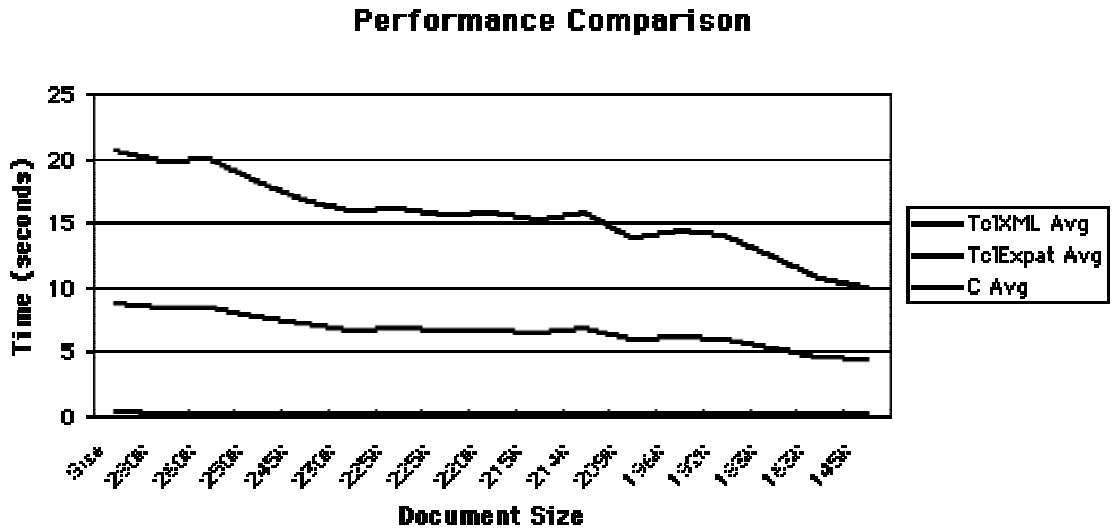

Figure 3: Performance Comparison of C Program with TclExpat and TclXML.

The timing data was calculated using a version of `TclExpat` and TclXML compiled against Tcl version 8.0, which does not support Unicode. Tcl version 8.1 alpha2 was not used because its regular expression engine is known to have poor performance as it has not yet been performance optimised. The tests were conducted on a Sun UltraSPARC 2.

The tests indicate that the Tcl script using the `TclExpat` extension is approximately 10 times slower than the equivalent C code, and the full Tcl version is 20 times slower. Given that the only difference between the two versions of the Tcl program are that the all-Tcl version must also parse the XML document in Tcl, it would appear that the performance loss in the `TclExpat` version is due to the task of writing the normalised XML document. A 10 to 20 times performance loss for Tcl scripts when compared to C code is acceptable. From this we can conclude that the `TclExpat` extension adds minimal overhead to `expat` for parsing XML documents. Further testing on more complex applications will be required to draw conclusions on the feasibility and/or efficiency of using Tcl to process XML documents, but this comparison test is encouraging.

An important point to note is that the C program included with the `expat` distribution is 653 lines of code. The Tcl script developed using the `TclExpat` extension or TclXML package is only 89 lines of code. This indicates that writing document processing applications is significantly more productive using Tcl.

Application Examples

Plume's XML support and TclXML are being used in two applications. Firstly, Steve Ball has written a book, titled "Web Tcl Complete" [Ball98B]. The publisher, McGraw-Hill, required the manuscript to be delivered as plain, double-spaced text, along with code and script examples to be included on a CD-ROM. However, it was also desirable to produce drafts of the chapters for the Web.

To meet all of these requirements the book was written in XML, with markup used to identify code examples along with other structural features. Two simple scripts were written, one to translate the documents into HTML and another to output plain text while at the same time extracting the code examples. The tree-based document representation allowed these scripts to be written quickly and easily. These scripts used the Plume XML support library, which provided some valuable experience to guide the development of the TclDOM specification.

Another application is an Intranet development for the Department of Foreign Affairs and Trade of the Australian Federal Government. For this project it was decided to store corporate documents in XML format, to allow rich structuring of the information for improved searching and vendor- and platform-independence of the data. To support the current generation of Web browsers, the XML documents are translated into HTML on-the-fly at the time of

document delivery. If the translation required is straightforward an event-based parser is used. An event-based parser allows large documents to be delivered incrementally, thus reducing the latency of document delivery and improving the perceived responsive of the system to the user. More complex translations use the tree-based representation of the document, which allows advanced formatting depending on the content of elements. These technologies are augmented with Tcl microscripting, using the Tcl Web Server [Tclhttpd], to provide a highly dynamic Web site.

Conclusion

An Application Programming Interface standard is being developed to provide a Tcl language binding for the DOM, known as TclDOM. TclDOM provides the means to create XML documents, as well as access and traverse their document structures. DOM supports both XML and HTML, and so too will TclDOM. A single interface to both standards will simplify the development of Tcl applications which wish to process both XML and HTML documents.

A Tcl package has been developed, called TclXML, which provides a sample implementation of the TclDOM. TclXML includes a non-validating, event-based XML parser written in Tcl, as well as a Tcl interface to the `expat` parser. Modules are, or will be, provided to create the document tree, validate documents and produce XML or HTML output. TclXML is being designed to take advantage of the significant number of Java packages that are available for processing XML documents.

XML, DOM and other related standards are, or will be, Recommendations of the World Wide Web Consortium which will have a major impact on the World Wide Web, as well as making inroads into application areas which currently use SGML. Many network protocols and document formats are proposing to use XML as their data syntax, such as CDF, RDF and PGML to name but a few. This means that it is very important for the Tcl language to be able to efficiently process XML documents, in order to be able to support the development of applications which make use of XML. Tcl 8.1 has the advantage that it supports Unicode, though Tcl 8.0 may also be used for western character sets.

Given the increasing importance of XML for a wide range of application development it is the author's opinion that XML support of some form must be included in the core Tcl distribution. TclXML is being distributed under a license that permits it to be added to Tcl. It is interesting to note that Perl is also undergoing development to include Unicode and XML support [Perl]. Tcl has a significant advantage in that it is already Unicode-enabled, and so has a lead on Perl development.

References

[TCLXML]
Zveno Pty Ltd. *Tcl Support For XML*.
http://www.zveno.com/zm.cgi/in-tclxml/

[XML]
Tim Bray, et al. *Extensible Markup Language*.
World Wide Web Consortium.
http://www.w3c.org/TR/

[W3C]
World Wide Web Consortium. http://www.w3c.org/

[DOM]
Lauren Wood, et al. *Document Object Model*.
World Wide Web Consortium.
http://www.w3c.org/TR/

[CSS]
Hakon Lie, et al. *Cascading Style Sheets*.
http://www.w3c.org/Style/

[Ball98A]
Steve Ball. *The Plume WWW Browser*.
http://plume.browser.org/

[Ball98B]
Steve Ball. *Web Tcl Complete*
McGraw-Hill, New York.
http://www.zveno.com/zm.cgi/in-wtc/

[Uhler95]
Uhler, S. *html_library*.
ftp://ftp.smli.com/research/tcl/html_library.tar.gz

[Clark98]
James Clark. *Expat XML Library*.
http://www.jclark.com/xml/

[English96]
Joe English. *CoST*. http://www.art.com/cost/

[PM-R96]
Peter Murray-Rust. *CoSTWish*.
http://www.venus.co.uk/omf/costwish/

[Nicol98]
Gavin Nicol, Inso Corporation. mailto:gtn@inso.com

[Tclhttpd]
Brent Welch, Scriptics Corporation.
http://www.scriptics.com/

[Perl]
Larry Wall. http://perl.oreilly.com/