Processing XML documents with Tcl and tDOM

FREEDOM

There's nothing to stop Tcl programmers from processing XML files with their own software. Carsten Zerbst takes a look at the tDOM package, which provides a fast, streamlined DOM implementation

XML is fast becoming the *lingua franca* of file formats. If you want to use XML within your own programs there is no need to write a parser, as many ready-made solutions are available for this purpose, not least for Tcl, which offers extensions enabling you to access XML document elements. One of these is tDOM, a package that is not only powerful but also unusually lean and fast.

Access to the actual parser is usually via one of two types of API. With SAX (*S*imple *A*PI for *X*ML) the parser reads the data and executes commands as soon as it reaches certain points in the text, even if it has not finished loading the document. In other words, this is an event-driven API. The alternative to SAX is DOM (Document Object Model). Here, the parser loads the complete document into memory first and stores its structural details in a tree structure. DOM offers more powerful document retrieval and editing facilities than SAX. This flexibility comes at the price of greater memory usage.

Implementations of this API are available in various programming languages, for example C and Java. The DOM API basically uses three different classes: one for creating documents, one for the documents themselves and one class for the different elements. The idea behind DOM is that it uses element objects to represent each XML tag, all attributes and every text section. The elements have methods that enable them to inherit elements located above, below and next to them. Other methods provide information about the element.

Table 1: Document commands

Command	Description
documentElement	Returns the root element
createElement *tagname*	Creates an element called *tagname*
createTextNode *text*	Creates a text element
createCommentText *text*	Creates a comment
createCDATASection *data*	Creates a CDATA element
createProcessingInstruction *data*	Creates a PI element

DOM and XPath in a single package

The two main DOM packages available under Tcl are Steve Ball's TclDOM and Jochen Löwers tDOM. In the following examples we are using tDOM; this package has an impressive functionality range and is very fast. Apart from the DOM specification it contains other useful things, such as an XPath implementation and a graphical query tool.

Compiling and installing tDOM is pretty simple: unpack the sources, change to the *unix* sub-directory and perform the usual routine (*configure, make, make install*).

tDOM in action

The *dom* command in tDOM creates new documents. *dom parse string* instructs the package to analyse a string of XML data and to create an XML document object from it. New empty documents are created using *dom createDocument name*.

You will frequently come across files which at first glance appear as though they should be easy to process with DOM, but are not pure XML; HTML being a case in point. There is a solution to this. The option *-html* allows tDOM to read many HTML pages without requiring any manual intervention.

The tDOM syntax is object-oriented in a similar way to Tk. The *dom* command's return value is a *domDoc* class object representing the entire document.

```
% set fd [open filename]
% set doc [dom parse [read $fd]]
% close $fd
```

Although this looks simple, it does in fact involve a lot of work. The raw data needs to be turned into a data structure with XML tags and attributes. tDOM does a pretty quick job of this. On a computer with a Duron processor (800 MHz, 512 Mb RAM) the normal parser takes only 80 milliseconds to read the 370Kb Mondial database. The Tcl interpreter and the XML document together take up a mere 2.5Mb of memory. By comparison, Java and JDom require 1900 milliseconds for the first read and 500 milliseconds for each subsequent read on the same

machine, with the JVM taking up 15Mb of resident memory.

The tDOM package uses James Clark's Expat parser and therefore supports Unicode, as is normal for Tcl. However, as technical data, for example, often does not require Unicode the ISO-Latin-1 character set will in many cases be sufficient. tDOM contains a second, faster parser for this eventuality, which can be selected using the *–simple* option. The data now takes only 45 milliseconds to read.

The memory requirement also depends on the size of the original file. tDOM needs about double to four times the size of the original file, other solutions in C and Java require up to 30 times.

Step by step

Once the document has been read, the *dom* command returns a *domDoc* object. Using this element's sub-commands, various element types can be created or the root element retrieved (see Table 1). The root element represents the top node, like the *<html>* tag in an HTML document. The type of the root node is *domNode*, the same as that of any other DOM element.

The most important commands for the (*domNode*) elements are listed in Table 2. Using these commands, the program can navigate through the branches of the DOM tree and retrieve the information it contains.

Creating new elements is a little more complicated. The DOM uses a two-step process. First, a document command creates a new element, for instance *createTextNode* creates a new element of the type *TEXT_NODE*. In the second step, this is appended to an element of the tree.

```
% set doc [dom createDocument html]
% set root [$doc documentElement]
% set node [$doc createTextNode "Hello"]
% $root appendChild $node
% puts [$root asHTML]

<html>Hello</html>
```

To the point

The commands mentioned up to now are pretty much part of the normal DOM environment but rather unwieldy for addressing XML elements. Should you require the first *td* in the third *table* element with an attribute setting of *rowspan=2* this will need quite a bit of work. Fortunately there is a far more elegant way in tDOM. The magic word is XPath, which enables you to search for nodes using a handy query language. This is another W3C specification.

The command *selectNodes query* can search the sub-tree starting at any node within a document. Depending on the type of query the results are nodes, attributes or their values. To search for all elements of a particular type, use *//TYPENAME*. To address the third element specifically use *//TYPENAME[3]*. As the query may contain square brackets, these must be protected in Tcl (either with backslashes or curly brackets). Table 3 shows some examples of XPath queries.

To get a quick overview of the XPath features, Jochen Löwer has written the xe tool, which displays the results of XPath queries in graphical format. The interface takes a bit of getting used to: the query is typed into to the upper window, highlighted with the mouse and submitted with the *execute <sel.>* button. The xe input consists of the desired file followed by the XPath query. xe displays the result, including child nodes, in the lower window, the branches of the tree can be expanded using the + symbol (see Figure 1).

Web-grabbing

XPath also makes Web-grabbing (the extraction of data from HTML files) very easy. The following example uses the BBC sports news ticker, from which we are going to read a news item. The last instalment of this series described how Tcl loads Web pages from the Internet. Here, we are concentrating on processing their HTML contents.

As soon as the page is available, we want tDOM to retrieve the summary of the latest news item and the

Table 2: Element commands

Command	Description
ownerDocument	Returns a reference to the document containing the element
Single node	
nodeName	Returns the name of the node
nodeType	Type of node, e.g. element or attribute
nodeValue	Node content for text nodes
Attributes	
attributes	Lists attributes
getAttribute *attrName*	Retrieves single attribute
@attrName	Short for *getAttribute*
setAttribute *attrName attrValue*	Sets attribute
Child nodes	
parentNode	Returns the node one level above in the tree
childNodes	Lists all child nodes
appendChild *child*	Appends element as child node
replaceChild *old new*	Replaces child element
removeChild *child*	Deletes child element
getElementsByTagName	Searches the entire sub-tree for elements with a specific name
XPointer	
descendant *count type*	Descendant elements by position and type
descendant all *type*	Descendant elements by type
XPath	
selectNode *query*	XPath query
Outputting the DOM tree	
asXML	Outputs sub-tree as XML
asHTML	Outputs sub-tree as HTML

Table 3: XPATH examples

Query	Description
/option	The *option* element directly below the root node
//option	All elements in the document called *option*
//option[3]	The third *option* element
/table/*	All elements below *table*, where *table* must be located directly below the root node
//table[1]	The first *table* element in a document
//table[last()]	The last *table* element
//@colspan	All *colspan* attributes in a document
//td[@colspan]	All *td* elements with the attribute *colspan*
//table[@width]	All *table* elements that have a *width* attribute
//table[@width=690]	All *table* elements with a *width* attribute that has a value of 690
//*[count(tr)=2]	All elements with two *tr* child nodes
//tr/tdlth	All *td* and *th* elements contained within a *tr* element
//table//img	All *img* elements contained within a *table*–element
//table[1]//img[2]	Second *img* element in the first *table*

Figure 1: The BBC sports news pages in a Web browser. The DOM tree can be displayed via the context menu, with the selected element on the HTML page also highlighted in the DOM viewer

reference to the complete story. But which element is the right one? We could either use xe and keep trying until the desired result appears, or we could get some help from Mozilla. If you press the right mouse button over the relevant text and select *Inspect* from the context menu a new window opens, displaying the DOM viewer (Figure 1 and 2). You can also reach the DOM inspector via *Tasks/Tools/DOM Inspector* in the SuSE 7.3 supplied Mozilla

A simple count is sufficient in this case. The latest text is in the seventh table, in the second line, in the second column, within a *<a href="">* element. It is advisable to save this page and design the relevant query in xe. Now you have all the information required for writing Listing 1. This calls the page and extracts the news item using tDOM. The script can be easily extended, for instance to load the full version of the news item.

Info

TclDOM
http://tclxml.sourceforge.net/tcldom.html
TDOM
http://sdf.lonestar.org/~loewerj/tdom.cgi

Listing 1: Web-grabbing with tDOM

```
#!/bin/sh
#
# \
exec tclsh8.3 $0 $@

package require tdom
package require http

set server "http://news.bbc.co.uk"
set path "sport/"
set url "$server/$path"

if {[catch {http::geturl $url -timeout 15000} ➥
token]} {
    puts stderr "Problem with network: $token"
    exit 1
}

if {[http::ncode $token] != 200} {
    puts stderr "Problem with server, ➥
[http::code $token]"
    exit 1
}

set doc [dom parse -html [::http::data $token]]
set root [$doc documentElement]
set node [$root selectNodes ➥
{//table[7]/tr[2]/td[2]/a}]
set text [[$node childNode] nodeValue]

puts "Latest news: $text"
```

tDOM, the universal XML tool

Simple DOM, parser extensions and XPath are by no means all that tDOM offers. Recently tDOM has also learnt XSLT, confirming its position as the XML equivalent of the Swiss army knife. XSLT support is almost complete in the current version 0.7test, and DOM 2 with name spaces is also supported.

Should you be interested in finding out more about the development of tDOM or have specific questions you will get to appreciate the tDOM mailing list. Overall, tDOM is a solid base for XML processing.

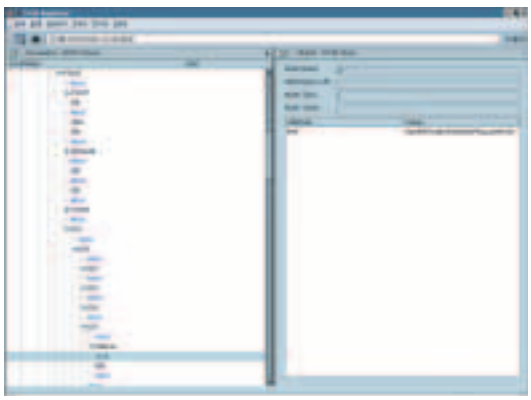

Figure 2: Mozilla shows the document's DOM structure. The A element is selected in the left tree view; on the right the DOM inspector shows the *href* attribute with its value (a relative link).