

my first Tck/Tk program... and an XML question

Source: <http://coding.derkeiler.com/Archive/Tcl/comp.lang.tcl/2006-12/msg00022.html>

- *From:* "jkj" <kevin@xxxxxxxxxx>
 - *Date:* 30 Nov 2006 10:06:38 -0800
-

Ok, appended to this (can you attach files to these posts?) is a program which takes an XML Schema file (something.xsd), converts it to a basic XML tree and allows the user to save it. I'll post it because I learned a lot by messing with it and perhaps it will generate some critical feedback that will be useful to me from either a Tcl/Tk or XML viewpoint. I rarely touch the mouse, know only a little about XML and a very little about Tcl/Tk but, hey, FORTRAN is pretty cool! :-)

I think the next step will be to use a package called "tile" and create different tabs to show different files and/or features since the application that ultimately needs to be modified uses "tile". The Tcl/Tk syntax will take some getting used to, to say the least. In the end, I want to wrap my head around Tcl/Tk, XML, and SOAP.

Regarding XML, I do wonder why implicit endtags are not recognized consistently as start AND end tags. In other words, if the XML contains:

```
<Catalog/>
```

rather than:

```
<Catalog></Catalog>
```

....then the parsings, from the my very inexperienced point of view, of these two instances are different even though they are syntactically equivalent (please correct me if I am wrong about syntactic equivalence). The statement used in the appended code "[`$default_xml asXML`]" produces output with "implicit end tags" – can it be modified so that explicit tags are produced when a node's definition is empty? is this a common hassle when working with XML??

Code follows (I'm on a UNIX system),
–Kevin

```
#!/bin/sh
# the next line restarts using wish \
exec wish "$0" "$@"

# needed for mktclapp/wish
if { [catch { package require tdom 0.8 } err_msg] } {
  puts stderr "tdom load error...$err_msg"
```

my first Tck/Tk program... and an XML question

```
exit -1
}
#####
# November 2006
# JKJ

# Open up the XML Schema file.
proc open_schema { schemaFile } {
    global schema_doc

    if { $schemaFile == "newfile" } {
        set file_types {
            { "XSD Files" "*.xsd *.XSD" }
            { "All Files" "*" }
        }
        set schemaFile [tk_getOpenFile -filetypes $file_types -parent .]
    } else {
        set file $schemaFile
    }

    if { [catch { set schema_fd [open $schemaFile r+] } err_msg] } {
        schema_file_error "open error" $err_msg $schemaFile
    }
    set schema_data [read $schema_fd]
    close $schema_fd

    set schema_doc [dom parse $schema_data]
    schema_convert $schema_doc
}

# Handles errors related to the schema file
proc schema_file_error { err_type err_msg filename } {
    puts stderr "XML Schema filename: \"$filename\""

    switch $err_type {
        "open error" {
            puts stderr "schema open error...$err_msg"
            exit -1
        }
        default {
            puts stderr "$err_type...$err_msg"
            exit -1
        }
    }
}

# Extract notes from documentation nodes.
proc extract_doc_notes { node } {
    set notes ""
```

my first Tck/Tk program... and an XML question

```
if { [$node localName] eq "annotation" } {
  set node [$node firstChild]
}
if { [$node localName] eq "documentation" } {
  set notes [$node text]
}

return $notes
}

# Given an XML node, extract out any
# elements which define a matching
# attribute name ("AttrName=") and
# return the list.
proc extract_field_names { node attrName } {
  set names ""

  # We are after the list of list of complex
  # types, so drill down to the next node
  # if needed.
  if { [$node localName] eq "complexType" } {
    set node [$node firstChild]
  }
  if { [$node localName] eq "sequence" && \
    [$node hasChildNodes] } {
    set child_nodes [$node childNodes]
    foreach c_node $child_nodes {
      if { [$c_node localName] eq "element" && \
        [$c_node hasAttribute $attrName] } {
        lappend names [$c_node getAttribute $attrName]
      }
    }
  } else {
    puts stderr { "\"extract_field_names\": expecting "\
      "a sequence node with children!" }
  }

  return $names
}

# Given a node, return the element
# defined by it, or the element
# defined by its child.
proc get_first_element { root } {
  # We want to get the names of the references
  # from the first child node for which
  # "name=" is defined...
  if { ![$root hasAttribute name] } {
```

my first Tck/Tk program... and an XML question

```
set node [$root firstChild]
} else {
set node $root
}

if { ![$node hasAttribute name] } {
puts stderr "Not an element starting point??"
return
}

# case-sensitivity check to see if
# "name=" value matches what was
# requested:
set firstElement [$node getAttribute name]

return $firstElement
}

# Initialize the top-level reference
# list from the given XML Schema.
# This initializes any notes provided
# as well as identifiers the names and
# limitations of the top-level References.
#####
proc schema_list { root id input_list input_doc \
input_mins input_maxs } {

upvar 1 $input_list return_list
upvar 1 $input_mins return_mins
upvar 1 $input_maxs return_maxs
upvar 1 $input_doc return_doc

set return_list ""
set return_mins ""
set return_maxs ""
set return_doc ""

# We want to get the names of the references
# from the first child node for which
# "name=" is defined...
if { ![$root hasAttribute name] } {
set node [$root firstChild]
} else {
set node $root
}

if { ![$node hasAttribute name] } {
puts stderr "Not an element starting point??"
return
}
}
```

my first Tck/Tk program... and an XML question

```
# case-sensitivity check to see if
# "name=" value matches what was
# requested:
set lc_id [string tolower $id]
set chk_id [string tolower [$node getAttribute name]]
if { $chk_id != $lc_id } {
  puts stderr "first child is \"[$node getAttribute name]\""
  puts stderr " ...expected \"$id\""
  exit -1
}

set child_nodes [$node childNodes]
foreach c_node $child_nodes {
  switch [$c_node localName] {
    "annotation" {
      set return_doc [extract_doc_notes $c_node]
    }
    "complexType" {
      set return_list [extract_field_names $c_node "ref"]
      set return_mins [extract_field_names $c_node "minOccurs"]
      set return_maxs [extract_field_names $c_node "maxOccurs"]
    }
    default {
      puts stderr "...child node has local name: \"[$c_node
localName]\""??"
    }
  }
  if { [$c_node hasAttribute name] } {
    puts "Name attribute is \"[$c_node getAttribute name]\""
  }
}

return return_list
}
```

```
# At this point we have list of
# reference names (i.e. we tracked
# down a "name=" and extracted out
# all of the "ref=" values found
# under complexType), so now bundle
# up those reference names in such
# a way that invoking "asXML" will
# correctly output them.
proc element_setup { ref_names } {
  set return_string ""
```

```
foreach c_node $ref_names {
  set openBrace " { "
  set closeBrace " } "
```

my first Tck/Tk program... and an XML question

```
set newLine "\n"
set r_node $c_node
set r_node [append newLine $r_node]
set newLine "\n"

# We have to invoke "createNodeCmd" on each node...
dom createNodeCmd elementNode $c_node
# We are building up a command string that,
# to the best I can tell, requires the
# presence of newlines, so if the node
# name is "Catalog", the the following line
# appends: "
#Catalog { " ... }
# ...ok – now why does Tcl care about
# balancing braces in comments!?!
# to the string being returned..., then if
# Catalog ends up with no children the final
# entry in the return string for Catalog winds
# up being: "
#Catalog { } "
#####
set return_string [append return_string [append r_node $openBrace]
]

# Check to see if the current node
# has an element definition (i.e.
# checking for "name=$c_node").
set curr_root [schema_search $c_node]
if { [llength $curr_root] > 0 } {
# Now obtain a list of all the references
# defined for this "ref=blah" for $c_node.
schema_list $curr_root $c_node \
new_list top_notes \
top_mins top_maxs
if { [llength $new_list] > 0 } {
set newLine "\n"
# Ok, this guy has children, so recursively
# add then to the return string before
# appending the closing brace for the
# current node.
set return_string [append return_string \
[append newLine \
[element_setup $new_list] ] ]
}
}
# ...append the closing brace.
set return_string [append return_string $closeBrace]
}

set newLine "\n"
if { [llength $return_string] > 0 } {
```

my first Tck/Tk program... and an XML question

```
# ...seems cleaner to add another newline.
return [append return_string $newLine]
}

return $return_string
}

# Search through the Schema for a node
# containing a specific "name="
# definition and return the ID of that
# node. As far as I understand, a
# given node should only have one
# instance of "name=".
proc schema_search { name } {
    global schema_root
    set returnvals ""

    set node [$schema_root find name $name]

    if { $node != "" } {
        if { [llength $node] > 1 } {
            puts stderr "Doubly-defined node: \"$node\"!!"
        } else {
            return $node
        }
    }

    return $returnvals
}
```

```
# We want to take the Schema file and
# generate a bare XML tree.
proc schema_to_xml { top_list } {
    global default_xml

    if { [llength $top_list] < 1 } {
        return
    }

    # For each name in the top_list listing,
    # generate a node in the XML template
    # file, then find any children of the
    # current node, recursively generate a
    # node for each of them, etc.
    # default_xml is a domDoc
    # root is a domNode
    # ...both relate to the XML which is
    # to be generated.
    set root [$default_xml documentElement]
```

my first Tck/Tk program... and an XML question

```
foreach c_node $top_list {
  dom createNodeCmd elementNode $c_node
  set newLine "\n"
  set openBrace " { "
  set closeBrace " } "
  set baseCmnd "$root appendFromScript"

  set cmndNode [element_setup $c_node]
  set cmndNode [append newLine $cmndNode]
  set cmndNode [append openBrace [append cmndNode $closeBrace]]
  set cmndNode [append baseCmnd $cmndNode]
  eval $cmndNode
}
}

# If there are unsaved changes, warn the user.
proc promptSave { } {
  global dirtyFlag

  if { $dirtyFlag > 0 && \
  [tk_messageBox -type yesno \
  -icon warning -message \
  "Discard unsaved changes?"] == no } {
    return -1
  }

  return 0
}

proc quit { } {
  global dirtyFlag

  if { $dirtyFlag > 0 } {
    if { [promptSave] == -1 } {
      return
    }
  }
  exit
}

proc save { option } {
  global dirtyFlag default_xml outputFilename

  set file_types {
    { "XML Files" "*.xml *.XML" }
  }

  switch $option {
    save {
```

my first Tck/Tk program... and an XML question

```
if { $outputFilename == "" } {
save "saveas"
return
} else {
set file $outputFilename
}
}
saveas {
set outputFilename [tk_getSaveFile -defaultextension ".xml" \
-filetypes $file_types]
}
default {
return
}
}
if { $outputFilename == "" } {
return
}

set fd [open $outputFilename "w"]
puts $fd [$default_xml asXML]
close $fd

set dirtyFlag 0
}
```

```
# Given a "domDoc", extract out the
# references from the top-level
# element, then use that list as
# the basis for converting the
# Schema to an XML tree.
proc schema_convert { schema_doc } {
global schema_root default_xml dirtyFlag

set schema_root [$schema_doc documentElement]

# Acquire the top-level references
# for this schema.
set firstElement [get_first_element $schema_root]
if { $firstElement eq "" } {
puts stderr "Element ID failure!"
exit 0
}
schema_list $schema_root $firstElement \
top_list top_notes \
top_mins top_maxs
if { [llength $top_list] < 1 } {
puts "No primary references identified!?! ([llength $top_list])"
exit -1
}
}
```

my first Tck/Tk program... and an XML question

```
# Convert the Schema to an XML
# version which can be used
# to populate the GUI.
set default_xml [dom createDocument $firstElement]
schema_to_xml $top_list
set dirtyFlag 1

displayXML ""
}

# By now, the Schema has been converted
# to an XML tree which can now be
# displayed using "[$default_xml asXML]",
# or simply display a default string.
proc displayXML { flag } {
    global default_xml

    set defaultString "Converted Schema to be displayed here\n\tas a bare
XML tree"

    destroy .xmltext

    frame .xmltext
    pack .xmltext -side left -expand 1 -fill y
    label .xmltext.label -text "XML Treeview of Schema"
    pack .xmltext.label

    set log [text .xmltext.log -yscrollcommand { .xmltext.sb set}]
    scrollbar .xmltext.sb -command { .xmltext.log yview } -orient
vertical
    pack .xmltext.sb -side right -expand 1 -fill y
    pack .xmltext.log -side left -expand 1 -fill y

    if { $flag eq "default" } {
        .xmltext.log insert 1.0 $defaultString
        .xmltext.log config -state disabled
    } else {
        .xmltext.log insert 1.0 [$default_xml asXML]
        .xmltext.log config -state disabled
    }
}

# Simple window with simple windows to
# which is added some default text.
proc make_top_window { } {
    global default_xml aboutString

    wm protocol . WM_DELETE_WINDOW {quit}
```

my first Tck/Tk program... and an XML question

```
set fileBar ".menubar.file"
set helpBar ".menubar.help"

menu .menubar -type menubar
. configure -menu .menubar

.menubar add cascade -label "File" -menu $fileBar
.menubar add cascade -label "Help" -menu $helpBar

menu $fileBar -tearoff 0
$fileBar add command -label "Open" -command { open_schema newfile }
$fileBar add command -label "Save" -command { save save }
$fileBar add command -label "Save As" -command { save saveas }

$fileBar add separator
$fileBar add command -label "Quit" -command {quit}

menu $helpBar -tearoff 0
$helpBar add command -label "About" \
-command { tk_dialog .d "About" \
$aboutString info 0 OK}
displayXML "default"
}

#####
# Given an XML Schema file, output the
# contents as a bare XML tree.
proc main {} {
global schema_doc default_xml current_xml \
schema_root dirtyFlag outputFilename \
aboutString

wm protocol . WM_DELETE_WINDOW {quit}
wm title . "Schema to XML Converter"
set aboutString "Schema to XML Converter\nNovember 2006, JKJ"

set top_notes ""
set top_list ""
set top_mins ""
set top_maxs ""
set dirtyFlag 0

set outputFilename ""
set firstElement ""

# Ok, build a window:
make_top_window
}
```

my first Tck/Tk program... and an XML question

```
# invoke main:  
main
```

.