```
;; The first four lines of this file were added by Dracula.
;; They tell DrScheme that this is a Dracula Modular ACL2 program.
;; Leave these lines unchanged so that DrScheme can properly load this file.
#reader(planet "reader.ss" ("cce" "dracula.plt") "modular" "lang")
#| Team Steele
   Software Engineering I
   Mxmlminidom

   Software that creates a document object model from XML input.
|#
(in-package "ACL2")

(require "../interfaces/Ixmlminidom.lisp")
(require "../interfaces/Ibasiclex.lisp")

;(defconst *whitespace* '(#\Space #\Return #\Newline #\Linefeed #\Tab ))
;(defconst *endtagname* (cons #\> (cons #\/ *whitespace*)))
;(defconst *endattrname* (cons #\= *whitespace*))

(module Mxmlminidom
  (import Ibasiclex)

  (include-book "list-utilities" :dir :teachpacks)

  ;These functions just wrap the basiclex functions of the same name
  ;returning mv's instead of lists because they are a guaranteed size
  ;and I like using mv-let.
  (defun split-on-token-mv (tok xs)
    (let* ((res (split-on-token tok xs)))
      (mv (car res) (cadr res) (caddr res))))
  ;See above
  (defun span-mv (ps xs)
    (let* ((res (span ps xs)))
      (mv (car res) (cadr res))))
  ;See above
  (defun split-at-delimiter-mv (ds xs)
    (let* ((res (split-at-delimiter ds xs)))
      (mv (car res) (cadr res))))
  ;See above
  (defun splitoff-prefix-mv (ps xs)
    (let* ((res (splitoff-prefix ps xs)))
      (mv (car res) (cadr res) (caddr res))))

  ;xml-escape (unescapedchars) → returns string with bad chars replaced
  ;    with entities
  (defun xml-escape (unescapedchars)
    (if (null unescapedchars)
        ""
        (let ((char (car unescapedchars))
              (rest (cdr unescapedchars)))
          (string-append
           (cond
            ((equal char #\&) "&amp;")
            ((equal char #\<) "&lt;")
            ((equal char #\>) "&gt;")
            ((equal char #\') "&apos;")
            ((equal char #\") "&quot;")
            (t (chrs->str (list char))))
           (xml-escape rest)
           ))))

  ;xml-serialize-attributes (attributes) → Returns a string that is
  ;    xml that represents the passed in attribute list.
  (defun xml-serialize-attributes (attributes)
    (if (null attributes)
        ""
        (let ((attribute (car attributes))
```

```
            (rest (cdr attributes)))
        (concatenate
         'string
         " "
         (xml-escape (str->chrs (car attribute)))
         "=\""
         (xml-escape (str->chrs (cadr attribute)))
         "\""
         (xml-serialize-attributes rest)))))


;xml-serizlize-nodes (xmlnodes) → Returns a string containing xml
;    nodes that represents the node list, xmlnodes.
(defun xml-serialize-nodes (xmlnodes)
  (if (null xmlnodes)
      ""
      (let ((node (car xmlnodes))
            (rest (cdr xmlnodes)))
        (mv-let (nodename attributes children)
                node
                (string-append
                 (if (equal nodename 'text)
                     (xml-escape (str->chrs children))
                     (concatenate 'string
                      "<"
                      (xml-escape (str->chrs nodename))
                      (xml-serialize-attributes attributes)
                      (if (null children)
                          "/>"
                          (concatenate 'string
                           ">"
                           (xml-serialize-nodes children)
                           "</"
                           (xml-escape (str->chrs nodename))
                           ">"))))
                 (xml-serialize-nodes rest))))))

;xml-serizlize-dom (xmlnode) → Returns a string containing an xml
;    document that represents the dom passed in through xmlnode.
(defun xml-serialize-dom (xmlnode)
      (string-append
       "<?xml version=\"1.0\"?>"
       (xml-serialize-nodes (list xmlnode)))
  )

;xml-unescape (escapedchars) → string with entities replaced
(defun xml-unescape (escapedchars)
  (if (and
       (consp escapedchars)
       (standard-char-listp escapedchars))
      (let* ((sot (split-on-token '(#\& ) escapedchars))
             (pre (car sot))
             (post (caddr sot))
             (sot1 (split-on-token '(#\; ) post))
             (thechar (car sot1))
             (theend (caddr sot1))
             (thecharstr (chrs->str thechar)))
            (concatenate 'string
             (chrs->str pre)
              (if (string-equal "amp" thecharstr)
                  "&"
                  (if (string-equal "lt" thecharstr)
                      "<"
                      (if (string-equal "gt" thecharstr)
                          ">"
                          (if (string-equal "quot" thecharstr)
                              "\""
```

```
                                (if (string-equal "apos" thecharstr)
                                    "'"
                                    ""))))))
                (xml-unescape theend)))
        ""))

  ;xml-readnodeproperties (xmlchars) →
  ; returns (mv attributes remainingxmlstring)
  (defun xml-readnodeproperties (xmlchars)
    (mv-let (ws1 att1)
            (span-mv *whitespace* xmlchars)
            (if (null att1)
                (mv nil nil)
                (if (member (car att1) *endtagname*)
                    (mv nil att1)
                    (mv-let
                     (propname r1)
                     (split-at-delimiter-mv *endattrname* att1)
                     (mv-let
                      (r2 r3 propvstart)
                      (split-on-token-mv '(#\" ) r1)
                      (mv-let
                       (propvalue r4 rest)
                       (split-on-token-mv '(#\" ) propvstart)
                       (mv-let
                        (props rrest)
                        (xml-readnodeproperties rest)
                        (mv (cons (mv
                                   (xml-unescape propname)
                                   (xml-unescape propvalue))
                                  props) rrest)))))
                    ))))

  ;xml-skipdontcares (xmlchars) → returns next xmlchars sans don't cares
  (defun xml-skipdontcares (xmlchars)
    (mv-let (ws1 tag1)
            (span-mv *whitespace* xmlchars)
            (if (null tag1)
                nil
                (if (equal (car tag1) #\< )
                    (if (equal (cadr tag1) #\? )
                        (mv-let (dontcare dctok therest)
                                (split-on-token-mv '(#\? #\> )
                                                   (cddr tag1))
                                (xml-skipdontcares therest))
                        (mv-let (matched dontcare comment)
                                (splitoff-prefix-mv '(#\! #\- #\- )
                                                    (cdr tag1))
                                (if (= (length matched) 3)
                                    (xml-skipdontcares
                                     (caddr
                                      (split-on-token
                                       '(#\- #\- #\> ) comment)))
                                    tag1)))
                    xmlchars))))

  ;xml-readnodes (xmlchars) → returns (mv nodes remainingxmlstring)
  (defun xml-readnodes (xmlchars)
    (mv-let (ws content)
            (span-mv *whitespace* (xml-skipdontcares xmlchars))
            (if (null content)
                (mv nil nil)
                (if (equal (car content) #\< )
                    (mv-let (ws1 tag1)
                            (span-mv *whitespace* (cdr content))
                            (if (null tag1)
                                (mv nil nil)
```

```
                                (if (equal (car tag1) #\/ )
                                    (mv-let (dontcare dctok therest)
                                            (split-on-token-mv
                                             '(#\> ) tag1)
                                            (mv nil therest))
                                    (mv-let (tagname tagattrs)
                                            (split-at-delimiter-mv
                                             *endtagname* tag1)
                                            (mv-let
                                             (attribs tagend)
                                             (xml-readnodeproperties
                                              tagattrs)
                                             (mv-let
                                              (dc1 dc2 inner)
                                              (split-on-token-mv
                                               '(#\> ) tagend)
                                              (mv-let
                                               (nodes rest)
                                               (if (equal
                                                    (car tagend)
                                                    #\/ )
                                                   (mv nil inner)
                                                   (xml-readnodes
                                                    inner))
                                               (mv-let
                                                (morenodes morerest)
                                                (xml-readnodes rest)
                                                (mv
                                                 (cons
                                                  (mv
                                                   (xml-unescape
                                                    tagname)
                                                   attribs
                                                   nodes) morenodes)
                                                 morerest)))))))))
                    (mv-let (thetext therest)
                            (split-at-delimiter-mv '(#\< ) content)
                            (if (null thetext)
                                (xml-readnodes therest)
                                (mv-let (nodelist restoftext)
                                        (xml-readnodes therest)
                                        (mv
                                         (cons
                                          (mv 'text nil
                                              (xml-unescape
                                               (append ws thetext)))
                                          nodelist)
                                         restoftext)))))))

;xml-readnode (xmlchars) → returns the root node from xmlstring
(defun xml-readnode (xmlchars)
  (let ((result (xml-readnodes (str->chrs xmlchars))))
    (if (null (car result))
        nil
        (caar result))))

;xml-getnodes (node nodename) → returns children of node with type
;  nodename
(defun xml-getnodes (node nodename)
  (if (= (length (caddr node)) 0)
      nil
      (let ((curnode (car (caddr node)))
            (rest (cdr (caddr node))))
        (if (equal (car curnode) 'text)
            (xml-getnodes (mv nil nil rest) nodename)
            (if (string-equal (car curnode) nodename)
                (cons curnode (xml-getnodes (mv nil nil rest) nodename))
```

```
                (xml-getnodes (mv nil nil rest) nodename))))))

;xml-getdeepnodes (node nodename) → returns children of node with type
;  nodename searching recursively using DFS with node as root.
(defun xml-getdeepnodes (node nodename)
  (if (= (length (caddr node)) 0)
      nil
      (let ((curnode (car (caddr node)))
            (rest (cdr (caddr node))))
        (if (equal (car curnode) 'text)
            (xml-getdeepnodes (mv nil nil rest) nodename)
            (if (string-equal (car curnode) nodename)
                (cons curnode
                      (xml-getdeepnodes
                       (mv
                        nil
                        nil
                        (concatenate 'list (caddr curnode) rest))
                       nodename))
                (xml-getdeepnodes
                 (mv
                  nil
                  nil
                  (concatenate 'list (caddr curnode) rest))
                 nodename))))))

;xml-getnode (node nodename) → returns first child node with type
;  nodename
(defun xml-getnode (node nodename)
  (car (xml-getnodes node nodename)))

;xml-getdeepnode (node nodename) → returns first child node with type
;  nodename searching recursively using DFS with node as root.
(defun xml-getdeepnode (node nodename)
  (if (= (length (caddr node)) 0)
      nil
      (let ((curnode (car (caddr node)))
            (rest (cdr (caddr node))))
        (if (equal (car curnode) 'text)
            (xml-getdeepnode (mv nil nil rest) nodename)
            (if (string-equal (car curnode) nodename)
                curnode
                (xml-getdeepnode
                 (mv
                  nil
                  nil
                  (concatenate 'list (caddr curnode) rest))
                 nodename))))))

;xml-getattribute (node attributename) → returns the value of node's
; attribute with name attributename
(defun xml-getattribute (node attributename)
  (if (= (length (cadr node)) 0)
      ""
      (let ((curattr (car (cadr node)))
            (rest (cdr (cadr node))))
        (if (string-equal (car curattr) attributename)
            (cadr curattr)
            (xml-getattribute (mv nil rest nil) attributename)))))


;xml-gettext (node) → returns the composite of all text inside of a node
(defun xml-gettext (node)
  (if (null node)
      ""
      (if (= (length (caddr node)) 0)
          ""
```

```
          (let ((curnode (car (caddr node)))
                (rest (cdr (caddr node))))
            (if (equal (car curnode) 'text)
                (string-append (caddr curnode)
                               (xml-gettext (mv nil nil rest)))
                (string-append (xml-gettext curnode)
                               (xml-gettext (mv nil nil rest))))))))

;xml-isattribute (attribute) → returns true iff attribute is an mv of
;  length 2 with both elements of the mv being strings
(defun xml-isattribute (attribute)
  (and
   (true-listp attribute)
   (equal (length attribute) 2)
   (stringp (car attribute))
   (stringp (cadr attribute))))

;xml-isattributelist (attributes) → returns true iff attributes
;  is nil or a list of attributes
(defun xml-isattributelist (attributes)
  (or
   (null attributes)
   (and
    (true-listp attributes)
    (xml-isattribute (car attributes))
    (xml-isattributelist (cdr attributes)))))

(mutual-recursion
 ;xml-isnode (node) → returns true iff node is actually a node
 (defun xml-isnode (node)
   (and
    (true-listp node)
    (equal (length node) 3)
    (or
     (and
      (equal 'text (car node))
      (null (cadr node))
      (stringp (caddr node)))
     (and
      (stringp (car node))
      (xml-isattributelist (cadr node))
      (xml-isnodelist (caddr node))))))

 ;xml-isnodelist (nodes) → returns true iff nodes is a list of nodes
 (defun xml-isnodelist (nodes)
   (and
    (true-listp nodes)
    (or
     (null nodes)
     (and
      (xml-isnode (car nodes))
      (xml-isnodelist (cdr nodes))))))

 )

;gluekids (nodes)→ Given a list of nodes, glue all nodes' children
;   together in one big list; i.e., if the nodes are rooted in some tree
;   where they're at depth k, then take all of the nodes at depth k+1
;   (cousins or siblings to one another), and put them into a list
;   together.
(defun gluekids (nodes)
  (if (consp nodes)
    (mv-let (nodename atts kids)
            (car nodes)
            (concatenate 'list kids (gluekids (cdr nodes))))
    nil))
```

```
;xml-bfsfindnodes (nodes nodename) → returns a list of children nodes of type
;    nodename using BFS search that are at the shallowest depth.
(defun xml-bfsfindnodes (nodes nodename)
  ; Build a dummy node that looks like xmlminidom, so we can act like
  ; nodes are rooted there as children, and we can just use xml-getnodes
  ; on it.
  (let* ((dummyroot (mv "dummyroot" nil nodes))
         (maybes (xml-getnodes dummyroot nodename)))
      (if maybes
          ; We found them
          maybes
          ; There were no nodes.  Look deeper.
          (xml-bfsfindnodes (gluekids nodes) nodename))))

(export Ixmlminidom))
```