```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Client-email.lisp
;
; Created on March 7, 2013 by Wesley R. Howell.
;
; Purpose:
; This file will execute the email generation from a client.
;
; CHANGE LOG:
; 3/11/2013 - Added the getEmailText function
; 3/7/2013 - Created file to handled the client logic for email messages
; ---------------------------------------------------------------------
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(in-package "ACL2")

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Base Lines
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;(break-at-set delimiters xs)
;Function copied from the list utiliites package. This function is copied
;since it is not necessary to load all the functions in this package and
;have ACL2 certify each function.
(defun break-at-set (delimiters xs)
  (if (or (not (consp xs))
          (member-equal (car xs) delimiters))
      (list '() xs)
      (let* ((first-thing (car xs))
             (break-of-rest (break-at-set delimiters (cdr xs)))
             (prefix (car break-of-rest))
             (suffix (cadr break-of-rest)))
          (list (cons first-thing prefix) suffix))))

;(break-at delimiter xs)
;Function copied from the list utiliites package. This function is copied
;since it is not necessary to load all the functions in this package and
;have ACL2 certify each function.
(defun break-at (delimiter xs)
   (break-at-set (list delimiter) xs))

;(getEmailXML email)
;returns the XML formatted code for the email object passed
;param email - the email structure
;return string
(defun getEmailXML (email)
  (if (endp email)
      nil
      (let* ((hd (concatenate 'string
                              "<?xml version='1.0'?>"
                              "<!DOCTYPE user SYSTEM '../../../dtd/messages.dtd'>"
                              "<email>"))
             (to (concatenate 'string
                              "<to>" (car (car (car email))) "@"
                              (car (cdr (car (car email))))
                              "</to>"))
             (from (concatenate 'string
                                "<from>" (car (cadr email)) "@"
                                (car(cdr (cadr email)))"</from>"))
             (sub (concatenate 'string
                               "<subject>" (cadr (cdr email)) "</subject>"))
             (msg (concatenate 'string
                               "<content>" (cadr (cdr (cdr email)))
                               "</content>"))
             (ft (concatenate 'string "</email>"
                              )))
              (list hd to from sub msg ft))))
```

```
;(getContactStructure str)
;gets the contact structure for a given string
;name@domain
;param - str - the string to parse
(defun getContactStructure (str)
  (let* ((chrs (coerce str 'list))
         (name (car (break-at #\@ chrs)))
         (domain (cdr (break-at #\@ chrs))))
        (list  (coerce name 'string)
               (coerce (cdr (car domain))
                       'string))))

;(getEmailStructure xml)
;Gets the email data structure for the tokenized XML file passed
;param - xml - the tokenized list of XML contents
(defun getEmailStructure (xml)
  (let* (;(xml (cdr (getEmailXMLTokens file)))
         (to (car (car (cdr (cdr xml)))))
         (from (car (car (cdr (cdr (cdr (cdr (cdr xml))))))))
         (sub (car (car (cddddr  (cddddr xml)))))
         (msg (car (car (cddddr (cddddr (cdddr xml)))))))
    (list (list (getContactStructure to))
                (getContactStructure from) sub msg)))
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;(splitToString toString)
;Returns the string value of a list of characters
;param - toString - the list of characters
(defun splitToString (toString)
   (coerce toString 'string)
)

;(splitToStruct toList)
;The recursive fuction to separate a string based to field
;separated by commas ',' to a list of contact structures
(defun splitToStruct (toList)
        (if (consp (car (cdr toList)))
            (cons (splitToString (car toList))
                    (splitToStruct
                     (break-at #\, (cdr (car (cdr toList))))))
            (list (splitToString (car toList)))
          )
)

;(splitToField toString)
;Entry point for the 'to' field parsing. This function will break apart
;the to field into a list of characters and then call the splitToStruct
;param - toString - the string to be parsed
(defun splitToField (toString)
  (let* ((tos (break-at #\, (coerce toString 'list))))
    (splitToStruct tos)
  )
)

;(generateEmailFromStrings to from sub msg)
;This will be the main logic point for the user input.
;taken string values entered from the user, this function will
;gereate the XML format for an email message.
(defun generateEmailFromStrings (to from sub msg)
   (let* (
         (tos (getContactStructure to))
         (fr (getContactStructure from))
         (subject sub)
         (message msg))
         (list (list tos) fr subject message))
)
```

```
;(multRecip toList from sub msg)
;This function will generate a list of Email XML objects for
;each recipient specified in the to field of an email message
(defun multRecip (toList from sub msg)
    (if (consp (cdr toList))
        (append (getEmailXML (generateEmailFromStrings (car toList)
                                                       from sub msg))
                (multRecip (cdr toList) from sub msg))
        (append (getEmailXML (generateEmailFromStrings (car toList) from sub msg)))
     )
)

;(email to from sub msg)
;Generates the email XML for email messages based on user string input.
(defun email (to from sub msg)
   (if (consp (cdr (splitToField to)))
     (multRecip (splitToField to) from sub msg)
    (getEmailXML (generateEmailFromStrings (car (splitToField to)) from sub msg))
   )
)

;(getEmailText tokxml)
;This function will be the base for the client output. This fuction will
;return the contents of an email message to the client
(defun getEmailText (tokxml)
  (let* ((xml (getEmailStructure tokxml))
            (to (concatenate 'string
                             "To: " (car (car (car xml))) "@"
                             (car (cdr (car (car xml))))
                             ))
            (from (concatenate 'string
                               "From: " (car (cadr xml)) "@"
                               (car(cdr (cadr xml)))))
            (sub (concatenate 'string
                              "Subject: " (cadr (cdr xml))))
            (msg (concatenate 'string
                              "Message: " (cadr (cdr (cdr xml)))
                              ))
       )
             (list to from sub msg))
)

;(getEmailText tokxml)
;This function will be the base for the client output. This fuction will
;return the contents of an email message to the client
(defun getEmailHTML (tokxml)
  (let* ((xml (getEmailStructure tokxml))
         (hd (concatenate 'string "<html><body>"))
            (to (concatenate 'string
                             "<h2>To: " (car (car (car xml))) "@"
                             (car (cdr (car (car xml)))) "</h2>"
                             ))
            (from (concatenate 'string
                               "<h2> From: " (car (cadr xml)) "@"
                               (car(cdr (cadr xml))) "</h2>"))
            (sub (concatenate 'string
                              "<h1>" (cadr (cdr xml)) "</h1><hr><br>"))
            (msg (concatenate 'string
                              "<p>" (cadr (cdr (cdr xml))) "</p>"
                              ))
            (ft (concatenate 'string "</body></html>"))
       )
             (list hd sub to from msg ft))
)
```