```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; mailing-list.lisp
; Created on February 19, 2013 by Matthew A. Crist.
;
; The data structure that will be allowed for each mailing list.  A
; mailing list defines a group of users that are subscribed to a user's
; mailing list and will be emailed en masse when the user sends an email
; to this "feed".
;
; Data Structure:
; ------------------------------------------------------------------------
; ("My Mailing List" ; The name of the mailing list
;  ; The mailing list recipients
;  (("localhost" "matthew.crist") ("localhost" "wesley.howell"))
;  ; The owner and verification of this list (for sending)
;  (("matthew.crist" "simulation")))
;
; XML File Format: (Store)
; ------------------------------------------------------------------------
; <?xml version='1.0'?>
; <!DOCTYPE mailing-list SYSTEM "../../../dtd/mailing-list.dtd">
; <mailing-list>
;    <list-name>My Mailing List</list-name>
;    <addresses>
;       <address>
;          <domain>localhost</domain>
;          <name>matthew.crist</name>
;       </address>
;       <address>
;          <domain>localhost</domain>
;          <name>wesley.howell</name>
;       </address>
;    </addresses>
;    <owners>
;       <owner>
;          <name>matthew.crist</name>
;          <password>simulation</password>
;       </owner>
;    </owners>
; </mailing-list>
;
; CHANGE LOG:
; ------------------------------------------------------------------------
; 2013-03-17    -   Actually added some code to this file.
; 2013-02-19    -   Initial file conception.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(in-package "ACL2")

; (findOwner owners owner)
; Locates an owner (by name) to ensure they are in this list.
; owners - the list of owners for a list
; owner  - the owner that will be sought.
(defun findOwner (owners owner)
    (if (endp owners)
        nil
        (if (equal (car owner) (cadar owners))
            (car owners)
            (findOwner (car owners) owner))))

; (verifyOwner owners owner)
; Predicate to determine if the owner has correct credentials to use this
; mailing list.
; owners    -   The data structure that houses the list of owners.
; owner  -  The owner that is trying to be verified.
(defun verifyOwner (mailing-list owner)
    (if (or (endp mailing-list) (endp owner))
        nil ; Can't find what isn't
```

```
        (let* ((owners (caddr mailing-list))
                 ; Verify the username
                 (found  (findOwner owners owner)))
            (if (equal found nil)
                nil
                ; Verify the password
                (if (equal (cadr found) (cadr owner))
                    t           ; Success!
                    nil)))))    ; Invalid credentials

; (isOwner mailing-list owner)
; Predicate to determine if the owner has the credentials to modify or
; send to the people contained in this mailing-list.
; mailing-list - the mailing list to be used.
; owner        - the credentials for log in of the owner.
(defun isOwner (mailing-list owner)
    (if (or (endp mailing-list) (endp owner))
        nil
        (let* ((owners (caddr mailing-list)))
            ; Force this check or can hack password if they know a name
            (if (equal (findOwner owners owner) nil)
                nil
                t))))

; (addOwner mailing-list owner)
; Adds an owner to the mailing list, given they do not already exist.
; mailing-list - the list in which to add the owner.
; owner        - the owner that will be added.
(defun addOwner (mailing-list owner)
    (if (or (endp mailing-list) (endp owner))
        mailing-list
        (let* ((name   (car   mailing-list))
                 (subs   (cadr  mailing-list))
                 (owners (caddr mailing-list))
                 (found  (findOwner owners owner)))
            (if (equal found nil)
                (list name subs (append owners (list owner)))
                mailing-list))))

; (removeOwner owners owner)
; Removes an owner from the list of owner, given the name is correct.
; owners - the list of owners in which to remove the owner.
; owner  - the owner that will be removed.
(defun removeOwner (owners owner)
    (if (or (endp owners) (endp owner))
        owners
        (if (equal (caar owners) (car owner))
            (cdr owners)
            (cons (car owners) (removeOwner (cdr owners) owner)))))

; (hasOwners mailing-list)
; Verifies that this mailing list has at least one owner!  This function
; will be used to determine if the list needs to be deleted due to
; mismanagement or intentional removal for list deletion.
(defun hasOwners (mailing-list)
    (if (endp mailing-list)
        nil
        (if (endp (caddr mailing-list))
            nil
            t)))

; (changePassword mailing-list owner new)
; Changes the password for an owner of this mailing list.
; mailing-list - the list that the owner should be able to be verified by
; owner        - the old credentials for the owner
; new          - the new password for the owner
(defun changePassword (mailing-list owner new)
    (if (or (endp mailing-list) (endp owner))
```

```
        mailing-list
        (let* ((name   (car   mailing-list))
               (subs   (cadr  mailing-list))
                (owners (caddr mailing-list))
                (ok     (verifyOwner mailing-list owner)))
            (if ok
                ; Can't change what we cannot verify
                mailing-list
                (let* ((newowners (removeOwner owners owner))
                   (newowner  (list (car owner) new)))
                    (list name subs (append newowners (list newowner)))))))))

; (hasSubscribers mailing-list)
; Predicate to determine if there are recipients that exist in the list.
; mailing-list - the mailing list that will be checked for recipients.
(defun hasSubscribers (mailing-list)
    (if (endp mailing-list)
        nil
        (if (endp (cadr mailing-list))
            nil
            t)))

; (isSubscriber subscribers subscriber)
; Locates a subscriber in the list of subscribers.
; subscribers - the list of subscribers to search.
; subscriber  - the subscriber that is being sought.
(defun isSubscriber (subscribers subscriber)
    (if (endp subscribers)
        nil
        (if (equal (car subscribers) subscriber)
            t
            nil)))

; (removeSubscriber subscribers subscriber)
; Removes a subscriber from a list.
; subscribers - the list in which the subscriber will be removed.
; subscriber  - the subscriber that will be removed.
(defun removeSubscriber (subscribers subscriber)
    (if (or (endp subscribers) (endp subscriber))
        subscribers
        (if (equal (car subscribers) subscriber)
            (cdr subscribers)
            (cons (car subscribers)
                    (removeSubscriber (cdr subscribers) subscriber)))))

; (unsubscribe mailing-list subscriber)
; Removes a subscriber from a mailing list.
; mailing-list - the list in which the subscriber will be removed.
; subscriber   - the subscriber that will be removed.
(defun unsubscribe (mailing-list subscriber)
    (if (or (endp mailing-list) (endp subscriber))
        mailing-list
        (let* ((subscribers (cadr  mailing-list))
                 (name        (car   mailing-list))
             (owners      (caddr mailing-list))
                 (ok          (isSubscriber subscribers subscriber)))
            (if ok
                (let* ((newsubs (removeSubscriber subscribers subscriber)))
                    (list name newsubs owners))
                mailing-list))))

; (subscribe mailing-list subscriber)
; Adds a subscriber to a mailing list.
; mailing-list - the list in which the subscriber will be added.
; subscriber   - the subscriber that will be added to the mailing list.
(defun subscribe (mailing-list subscriber)
    (if (or (endp mailing-list) (endp subscriber))
        mailing-list
```

```
        (let* ((subscribers (cadr  mailing-list))
                (name        (car   mailing-list))
                 (owners      (caddr mailing-list))
                 (subbed      (isSubscriber subscribers subscriber)))
            (if subbed
                mailing-list
                (list name (append subscribers (list subscriber)) owners)))))

; (renameMailingList mailing-list new-name owner)
; Renames a mailing list, given the user has the credential to do so.
; mailing-list - the mailing list that will be renamed.
; new-name     - the new name of the mailing list.
; owner        - the credentials to verify the user has access.
(defun renameMailingList (mailing-list new-name owner)
    (if (or (endp mailing-list) (endp owner))
        mailing-list
        (let* ((subscribers (cadr  mailing-list))
                 (owners      (caddr mailing-list))
                 (ok          (verifyOwner mailing-list owner)))
            (if ok
                (list new-name subscribers owners)
                mailing-list))))
```