;; This is a listing of the 21-Streams.NoteFile. It tests all of the functions in chapter 21 of Common Lisp the Language by Guy Steele. The individual test files for each of the functions have been appended together in this big file to share routines for testing a stream and to gain diagnostic information by testing the functions in a particular order.
;;
;; The source for this text file is the NoteCards database at {eris}<lispcore>cml>test>21-Streams.NoteFile. Changes are NOT made directly to the listing:
;;Filed As: {eris}<lispcore>cml>test>21-streams.test
;;

```
(do-test "setup stream source and sink names"
    ;; Note: implementation dependent file names below.  For
    ;;portability the stream names at minimum use the file
    ;;name "TEST".  Thus they are a function of the current
    ;;connected directory at the time the test is run!!!
    (test-setq stream-io-names
        (nconc (list "TEST")
            (cond
                ((string-equal (lisp-implementation-type) "Xerox")
                 (when nil
                     (list "{core}test" "{dsk}test")
                          ;ignored for testing test
                 )
                 (when nil
                     (list   "{erinyes}<test>test" " {pele:}<lispcore>test"
                         "{10.0.0.56}"
                     ;SUMEX  requires loading TCP and
                     ;having an account on SUMEX
                         *terminal-io* *debug-io*
                         *query-io*
                     ;testing these means hand typing
                     ;appropriate response
                         "{VAXC}/user/xais/test/test"))))))
    ;;the following may be useful in some tests if set up correctly
    (test-setq stream-source-names
            (append stream-io-names (when nil (list *standard-input* "string"))))
    (test-setq stream-sink-names
            (append stream-io-names
                 (when nil (list *standard-output* *error-output* "string")))))
(do-test "defun setup-input-streams"
    (test-defun setup-input-streams (stream-names)
        (mapcar
            #'(lambda (stream-name)
                  (if     (string-equal stream-name "string")
                      (make-string-input-stream test-string)
                      (let ((astream (open stream-name :direction :output
                                           :if-does-not-exist :create :if-exists
                                           :supersede)))
                          ;;put something in the sources
                          (output-test astream)
                          ;;open them for input
                          (open stream-name)))) stream-names)))
(do-test "define error logger"
      (defun print-stream-error (fun-name &optional (stream-name ""))
           (print (concatenate 'string fun-name " failed"
                       (unless (string-equal stream-name "")
                            (concatenate 'string " on " stream-name)))
                  *error-output*)))
```

;; Functions To Be Tested: streamp input-stream-p
;; output-stream-p and stream-element-type

;;

;; Source: CLtL p. 329-332

;;

;; Chapter 21: Streams Section 21-2&3: Creating New
;; Streams and Operations on Streams

;;

;; Created By: Kirk Kelley

;;

;; Creation Date: 31 October 86

;;

```
;;  Last Update:   >> n MonthName << 86

;;

;;  Filed As:      {eris}<lispcore>cml>test>21-streams.def

;;
;;  Function  To Be Tested: streamp

;;

;;  Source:        CLtL p. 332

;;  Chapter 21: Streams       Section 21-3: Operations on
;; Streams

;;

;;  Syntax: streamp object

;;

;;  Function Description: streamp is true if its argument is a
;; stream, and otherwise is false.  (streamp x) = (typep x
;; 'stream)

;;

;;  Argument(s):  object

;;

;;  Returns: true or false

;;

;;  Function  To Be Tested: input-stream-p

;;

;;  Syntax: input-stream-p stream

;;

;;  Function Description: This predicate is true if its
;; argument (which must be a stream) can handle input
;; operations, and otherwise is false.

;;

;;  Argument(s):  stream

;;

;;  Returns: true or false

;;

;; Function  To Be Tested: stream-element-type

;;

;;  Syntax: stream-element-type stream

;;

;;  Function Description: A type specifier is returned to
;; indicate what objects may be read from or written to the
;; argument stream, which must be a stream.  streams
;; created by open will have an element type restricted to a
;; subset of character or integer, but in principle a stream
;; may conduct transactions using any LISP objects.

;;

;;  Argument(s):  stream

;;

;;  Returns: type specifier
```

```
;;
;; Function  To Be Tested: output-stream-p

;;

;;  Syntax: output-stream-p stream

;;

;;  Function Description: This predicate is true if its
;; argument (which must be a stream) can handle output
;; operations, and otherwise is false.

;;

;;  Argument(s):  stream

;;

;;  Returns: true or false

;;
(do-test "stream predicates"
       (defun input-test (astream &key keep-open dont-test-for-eof)
            (and (streamp astream)
                (input-stream-p astream)
                (or (subtypep (stream-element-type astream) 'integer)
                   (subtypep (stream-element-type astream) 'character))
                (equal (read astream) 'hello)
                (or dont-test-for-eof (read astream nil t))
                (or keep-open (close astream))))
       (defun mult-input-test (streamlist options)
            (if options
              (if (atom streamlist) t
                (and (input-test (car streamlist))
                    (mult-input-test (cdr streamlist) nil)))
              (if (atom streamlist) t
                (and (input-test (car streamlist) :keep-open :dont-test-for-eof)
                    (mult-input-test (cdr streamlist) t)))))
       (defun output-test (astream &key keep-open)
            (and (streamp astream)
                (output-stream-p astream)
                (or (subtypep (stream-element-type astream) 'integer)
                   (subtypep (stream-element-type astream) 'character))
                (print 'hello astream)
                (or keep-open (close astream))))
       (test-setq test-string "hello"))
;; Function To Be Tested: make-string-input-stream
;;
;; Source:        CLtL p. 330
;; Chapter 21: Streams       Section 21-2: Creating New Streams
;;
;; Created By:    Kirk Kelley
;;
;; Creation Date:  31 October 86
;;
;; Last Update:   >> n MonthName << 86
;;
;; Filed As:      {eris}<lispcore>cml>test>21-2-make-string-input-stream.test
;;
;;
;; Syntax: make-string-input-stream string &optional start end
;;
;; Function Description: This returns an input stream.  The input stream will supply, in order,
the characters in the substring of string delimited by start and end; after the last character
has been supplied, the stream will then be at end-of-file.
;;
;; Argument(s):  string, start -- integer, end -- integer
;;
;; Returns: output stream
;;
(do-test-group "make-string-input-stream"
     (do-test make-string-input-stream-simple-test
           (and (test-setq astream (make-string-input-stream test-string))
               (input-test astream)))
     (do-test make-string-input-stream-bounded-test
           (and (test-setq astream (make-string-input-stream test-string 0 5))
               (input-test astream)))
     (do-test make-string-input-stream-bounded-test
           (and (test-setq astream
```

```
                    (make-string-input-stream (concatenate 'string "well "
                                                           test-string " hi")
                                              5 (+ 5 (length test-string))))
              (input-test astream))))
;; Function To Be Tested: make-string-output-stream and get-output-stream-string
;;
;; Source:        CLtL p. 330
;; Chapter 21: Streams       Section 21-2: Creating New Streams
;;
;; Created By:    Kirk Kelley
;;
;; Creation Date:  31 October 86
;;
;; Last Update:   >> n MonthName << 86
;;
;; Filed As:      {eris}<lispcore>cml>test>21-2-make-string-output-stream.test
;;
;;
;; Syntax: make-string-output-stream
;;
;; Function Description: This retruns an output stream that will accumulate all output given it
for the benefit of the function get-output-stream-string.
;;
;; Argument(s):  none
;;
;; Returns: output-stream
;;
;;
;; Syntax: get-output-stream-string string-output-stream
;;
;; Function Description: Given a stream produced by make-string-output-stream, this returns a
string containing all the characters output to the stream so far.  The stream is then reset;
thus each call to get-output-stream-string gets only the characters since the last such call (or
the creation of the stream, if no such previous call has been made).
;;
;; Argument(s):  string-output-stream
;;
;; Returns: string
;;
(do-test-group
    ("make-string-output-stream group" :after
     (progn (close astream)
          (close bstream)))
    (do-test "make-string-output-stream"
        (and (test-setq astream (make-string-output-stream))
            (output-test astream :keep-open t)
            (test-setq bstream
                (make-string-input-stream (get-output-stream-string astream)))
            (print "somemore" astream)
            (input-test bstream)
            (test-setq bstream
                (make-string-input-stream (get-output-stream-string astream)))
            (string-equal (read bstream) "somemore")
            (close astream)
            (read-char bstream nil t)
            (close bstream))))
;; Function To Be Tested: with-input-from-string    [Macro]
;;
;; Source:        CLtL p. 330
;; Chapter 21: Streams       Section 21-2: Creating New Streams
;;
;; Created By:    Kirk Kelley
;;
;; Creation Date:  31 October 86
;;
;; Last Update:   >> day month << 86
;;
;; Filed As:      {eris}<lispcore>cml>test>21-2-with-input-from-string.test
;;
;;
;; Syntax: with-input-from-string (var string {keyword value}*) {declaration}* {form}*
;;
;; Function Description: The body is executed as an implicit progn with the variable var bound to
a character input stream that supplies successive characters from the value of the form string.
with-input-from-string returns the results from the last form of the body.  See CLtL p 330-331
for more info.
;;
;; Argument(s):  var - variable; string -- form;
;;      keyword --      :index -- form of place acceptable to setf
;;                      :start, :end -- form resolving to non-negative integers
```

```
;;
;; Returns: result of last form of the body
;;
(do-test-group "with-input-from-string"
    (do-test with-input-from-string-simple-test
          (with-input-from-string (astream test-string)
                        (input-test astream)))
    (do-test with-input-from-string-book-test
        ;; from the CLtL book
        (and (with-input-from-string (astream "Animal Crackers" :index j :start 6)
                        (read astream))
             (eql j 15)))
    (do-test with-input-from-string-bounded-test
          (and (with-input-from-string (astream (concatenate 'string "well "
                                                  test-string " hi")
                                       :index j :start 5 :end 11)
                                (input-test astream :keep-open t))
               (eql j 11))))
;; Function To Be Tested: with-output-to-string      [Macro]
;;
;; Source:        CLtL p. 331
;; Chapter 21: Streams       Section 21-2: Creating New Streams
;;
;; Created By:    Kirk Kelley
;;
;; Creation Date:  31 October 86
;;
;; Last Update:   >> day month << 86
;;
;; Filed As:      {eris}<lispcore>cml>test>21-2-with-output-to-string.test
;;
;;
;; Syntax: with-output-to-string (var [string]) {declaration}* {form}*
;;
;; Function Description: The body is executed as an implicit progn with the variable var bound to
a character output strema.  All output to that stream is saved in a string.  See CLtL page 331
for more.
;;
;; Argument(s):  var -- variable; string -- form;  declarations;  forms;
;;
;; Returns: if no string is specified, then string.  Otherwise value of last form.
;;
(do-test-group "with-output-to-string"
    (do-test with-output-to-string-simple-test
        (input-test
            (make-string-input-stream
                (with-output-to-string (astream)
                                (output-test astream :keep-open t)))))
    (do-test with-output-to-string-supplied-test
        (let (astring)
            (and (with-output-to-string (astream (setq astring
                                          (make-array 14
                                                  :element-type
                                                  'string-char
                                                  :fill-pointer 0)))
                                (print 'hello astream))
                 (string-equal "
                 hello " astring))))
    (do-test with-output-to-string-supplied-test2
        (let (astring)
            (and (with-output-to-string (astream (setq astring
                                          (make-array 14
                                                  :element-type
                                                  'string-char
                                                  :fill-pointer 0)))
                                (output-test astream :keep-open t))
                 (input-test (make-string-input-stream astring))))))
;; Function To Be Tested: with-open-stream [Macro]
;;
;; Source:        CLtL p. 330
;; Chapter 21: Streams       Section 21-2: Creating New Streams
;;
;; Created By:    Kirk Kelley
;;
;; Creation Date:  31 October 86
;;
;; Last Update:   >> day month << 86
;;
;; Syntax: with-open-stream (var stream) {declaration}* {form}*
;;
```

```
;; Function Description: The form stream is evaluated and must produce a stream.  The variable
var is bound with the stream as its value, and then the forms of the body are executed as an
implicit progn; the results of evaluating the last form are returned as the value of the with-
open-stream form.  The stream is automatically closed on exit from the with-open-stream form, no
matter whether the exit is normal or abnormal.  The stream should be regarded as having dynamic
extent.
;;
;; Argument(s):  local variable, stream form, declarations, and forms
;;
;; Returns: result of last form
;;
(do-test-group
    ("with-open-stream" :after
     (dolist (stream-name stream-source-names)
          (delete-file (probe-file stream-name))))
    (do-test with-open-stream-output-test
        (dolist (stream-name stream-sink-names t)
            (declare (special stream-name))
            (with-open-stream
                (astream (open stream-name :direction :output :if-exists
                               :new-version :if-does-not-exist :create))
                (or   (output-test astream :keep-open t)
                    (print-stream-error
                        "WITH-OPEN-STREAM-OUTPUT-TEST"
                        stream-name)))
            ;;make sure it got closed
            (or (close (open stream-name))
               (print-stream-error "WITH-OPEN-STREAM-OUTPUT-TEST"
                           stream-name))))
    (do-test with-open-stream-input-test
        ;; note this test assumes with-open-stream-output-test
        ;;has been run
        (dolist (stream-name stream-source-names t)
            (declare (special stream-name))
            (with-open-stream (astream (open stream-name))
                (or   (input-test astream :keep-open t)
                    (print-stream-error
                        "WITH-OPEN-STREAM-INPUT-TEST"
                        stream-name)))
            ;;make sure it got closed
            (or (close (open stream-name))
               (print-stream-error "WITH-OPEN-STREAM-INPUT-TEST"
                           stream-name)))))
;; Function To Be Tested: make-broadcast-stream
;;
;; Source:        CLtL p. 329
;; Chapter 21: Streams       Section 21-2: Creating New Streams
;;
;; Created By:    Kirk Kelley
;;
;; Creation Date:  31 October 86
;;
;; Last Update:   >> day month << 86
;;
;; Filed As:      {eris}<lispcore>cml>test>21-2-make-broadcast-stream.test
;;
;;
;; Syntax: make-broadcast-stream streams
;;
;; Function Description: This returns a stream that only works in the output direction.  Any
output sent to this stream will be sent to all of the streams given.  The set of operations that
may be performed on the new stream is the intersection of those for the given streams.  The
results returned by a stream operation are the values resulting from performing the operation on
the last stream in streams, the results of performing the operation on all preceding streams are
discarded.  If no streams are given as arguments, then the result is a "bit sink"; all output to
the resulting stream is discarded.
;;
;; Argument(s):  stream(s)
;;
;; Returns: stream
;;
(do-test-group
    (make-broadcast-stream-test :before
        (test-setq output-streams
               (mapcar #'(lambda (stream-name)
                           (open stream-name :direction :output :if-exists
                                :new-version :if-does-not-exist :create))
                     stream-sink-names)) :after
        (progn (mapcar #'close output-streams)
            (dolist (stream-name stream-sink-names)
```

```
                (delete-file (probe-file stream-name)))
            ;;note each of the following tests must be done in
            ;;sequence
        ))
    (do-test make-broadcast-stream-creation-test
          (test-setq astream (apply #'make-broadcast-stream output-streams)))
    (do-test make-broadcast-stream-list-test
          (expect-errors (error) (make-broadcast-stream '(some random list))))
    (do-test make-broadcast-stream-output-test (output-test astream))
    (do-test make-broadcast-stream-results-test (mapcar #'close output-streams)
          (test-setq output-streams (mapcar #'open stream-sink-names))
          (or (mult-input-test output-streams nil)
             (print-stream-error "MAKE-BROADCAST-STREAM-TEST"
                          (namestring astream)))))
;; Function To Be Tested: make-concatenated-stream
;;
;; Source:        CLtL p. 329
;; Chapter 21: Streams       Section 21-2: Creating New Streams
;;
;; Created By:    Kirk Kelley
;;
;; Creation Date:  31 October 86
;;
;; Last Update:   >> n MonthName << 86
;;
;; Filed As:      {eris}<lispcore>cml>test>21-2-make-concatenated-stream.test
;;
;;
;; Syntax: make-concatenated-stream &rest streams
;;
;; Function Description: This returns a stream that only works in the inut direction.  Input is
taken from the first of the streams until it reaches end-of-file; then that stream is discarded,
and input is taken from the next of the streams, and so on.  If no arguments are given, the
result is a stream with no content; any input attempt will result in end-of-file.
;;
;; Argument(s):  streams
;;
;; Returns: stream
;;
(do-test-group "make-concatenated-stream"
    (do-test make-concatenated-stream-simple-test
        (setq astream
            (open "test" :direction :output :if-exists :new-version
                :if-does-not-exist :create))
        (output-test astream)
        (setq original-stream (open "test"))
        (prog1
            (and (setq astream (make-concatenated-stream original-stream))
                (input-test astream))
            (close original-stream)
            (close astream)
                          ;just in case
            (delete-file (probe-file "test"))))
    (do-test make-concatenated-stream-string-test
          (setq original-stream (make-string-input-stream test-string))
          (and (setq astream (make-concatenated-stream original-stream))
            (input-test astream)))
    (do-test "MAKE-CONCATENATED-STREAM"
        (setq input-streams (setup-input-streams stream-io-names))
        (prog1
            (and (setq concatenated-stream
                    (apply #'make-concatenated-stream input-streams))
                (dolist (astream input-streams t)
                    (or   (input-test concatenated-stream :keep-open t
                                :dont-test-for-eof t)
                          (print-stream-error
                              "MAKE-CONCATENATED-STREAM"
                              (namestring astream))))
                (close concatenated-stream))
            (mapcar #'close input-streams)
            (dolist (stream-name input-streams)
                (delete-file (probe-file stream-name)))))
    (do-test make-concatenated-stream-closed-test
          (and (close (setq closed.file.stream
                      (open "emptyfile" :direction :output :if-exists
                          :new-version :if-does-not-exist :create)))
            (delete-file (probe-file "emptyfile"))
            (setq concatenated-stream
                (make-concatenated-stream closed.file.stream))
            (expect-errors (error) (read-char closed.file.stream))
```

```
            (expect-errors (error) (close concatenated-stream))))
    (do-test make-concatenated-stream-empty-test
          (and (setq empty-stream (make-concatenated-stream))
              (read empty-stream nil t)
              (close empty-stream)))
    (do-test make-concatenated-stream-string-test
          (setq astream (make-string-input-stream test-string))
          (and (setq concatenated-stream (make-concatenated-stream astream))
              (input-test astream))))
;; Function To Be Tested: make-two-way-stream
;;
;; Source:        CLtL p. 329
;; Chapter 21: Streams       Section 21-2: Creating New Streams
;;
;; Created By:    Kirk Kelley
;;
;; Creation Date:  31 October 86
;;
;; Last Update:   >> n MonthName << 86
;;
;; Filed As:      {eris}<lispcore>cml>test>21-2-make-two-way-stream.test
;;
;;
;; Syntax: make-two-way-stream input-stream output-stream
;;
;; Function Description: This returns a bidirectional strem that gets its input from input-stream
and sends its output to output-stream.
;;
;; Argument(s):  input-stream output-stream
;;
;; Returns: stream
;;
(do-test-group "make-two-way-stream"
    (do-test make-two-way-stream-file-test
        (dolist (stream-name stream-io-names t)
            (test-setq instream
                   (open stream-name :direction :output :if-exists
                       :new-version :if-does-not-exist :create))
            (output-test instream)
            (test-setq instream (open instream))
            (test-setq outstream
                   (open "testout" :direction :output :if-exists :new-version
                       :if-does-not-exist :create))
            (unless
                (prog1
                    (and (test-setq two-way-stream
                                (make-two-way-stream instream
                                                     outstream))
                         (streamp two-way-stream)
                         (input-stream-p two-way-stream)
                         (output-stream-p two-way-stream)
                         (equal (read two-way-stream) 'hello)
                         (print "it works" two-way-stream)
                         (expect-errors (end-of-file) (read two-way-stream))
                         (close two-way-stream)
                         ;; should instream and outstream be
                         ;;closed? if so, should test here
                     )
                    (close instream)
                    (close outstream)
                    (delete-file stream-name)
                    (delete-file (probe-file "testout")))
                (print-stream-error "make-two-way-stream-file-test"
                                    stream-name))))
    (do-test make-two-way-stream-string-test
          (test-setq astream (make-string-input-stream test-string))
          (test-setq bstream (make-string-output-stream))
          (and (test-setq two-way-stream
                      (make-two-way-stream astream bstream))
              (streamp two-way-stream)
              (output-stream-p two-way-stream)
              (input-test two-way-stream :keep-open t)
              (prin1 'garbage two-way-stream)
              (string-equal "garbage" (get-output-stream-string bstream))
              (close two-way-stream)
              (close astream)
              (close bstream)))
    (do-test make-two-way-stream-closed-test
          (test-setq astream (make-string-input-stream test-string))
          (close (test-setq closed.file.stream
```

```
                        (open "emptyfile" :direction :output :if-exists
                            :new-version :if-does-not-exist :create)))
          (delete-file (probe-file "emptyfile"))
          (and (test-setq two-way-stream
                      (make-two-way-stream astream closed.file.stream))
               (expect-errors (error) (print "any random thing" two-way-stream))
               (close two-way-stream)
               (close astream)))
     (do-test make-two-way-stream-backwards-test
          (test-setq instream (make-string-input-stream test-string))
          (test-setq outstream (make-string-output-stream))
          (and (test-setq two-way-stream
                      (make-two-way-stream outstream instream))
               (expect-errors (error) (print "backwards" two-way-stream))
               (expect-errors (error) (read two-way-stream))
               (close two-way-stream)
               (close instream)
               (close outstream))))
;; Function To Be Tested: make-echo-stream
;;
;; Source:        CLtL p. 330
;; Chapter 21: Streams       Section 21-2: Creating New Streams
;;
;; Created By:    Kirk Kelley
;;
;; Creation Date:  31 October 86
;;
;; Last Update:   >> n MonthName << 86
;;
;; Filed As:      {eris}<lispcore>cml>test>21-2-make-echo-stream.test
;;
;;
;; Syntax: make-echo-stream input-stream output-stream
;;
;; Function Description: This returns a bidirectional stream that gets its input from input-
stream and sends its output to output-stream.  In addition, all input taken from input-stream is
echoed to output-stream.
;;
;; Argument(s):  input-stream output-stream
;;
;; Returns: stream
;;
(do-test-group "make-echo-stream"
     (do-test make-echo-stream-file-test
          (dolist (stream-name stream-io-names t)
               (test-setq instream
                       (open stream-name :direction :output :if-exists
                           :new-version :if-does-not-exist :create))
               (output-test instream)
               (test-setq instream (open stream-name))
               (test-setq outstream
                       (open "testout" :direction :output :if-does-not-exist
                           :create))
               (unless (prog1 (and (test-setq echo-stream
                                              (make-echo-stream instream
                                                                outstream))
                                   (output-stream-p echo-stream)
                                   (input-test echo-stream :keep-open t
                                           :dont-test-for-eof t)
                                   (output-test echo-stream :keep-open t)
                                   (read echo-stream nil t)
                                   (close echo-stream)
                                   (test-setq outstream (open "testout"))
                                   (input-test outstream))
                              (close echo-stream)
                              (close instream)
                              (close outstream)
                              (delete-file (probe-file stream-name))
                              (delete-file (probe-file "testout")))
                       (print-stream-error "make-ECHO-stream-file-test"
                                           stream-name))))
     (do-test make-echo-stream-string-test
         ;;DEPENDS ON TEST-STRING SETUP WITH
         ;;STREAM PREDICATES
         (test-setq astream (make-string-input-stream test-string))
         (test-setq bstream (make-string-output-stream))
         (prog1
             (and (test-setq echo-stream (make-echo-stream astream bstream))
                  (streamp echo-stream)
                  (output-stream-p echo-stream)
```

```
                (input-test echo-stream :keep-open t)
                (string-equal "HELLO" (get-output-stream-string bstream))
                (close echo-stream)
                (close astream)
                (close bstream))
            ;;just in case
            (close echo-stream)
            (close astream)
            (close bstream)))
    (do-test make-echo-stream-closed-test
          (test-setq astream (make-string-input-stream test-string))
          (close (test-setq closed.file.stream
                      (open "emptyfile" :direction :output :if-exists
                          :new-version :if-does-not-exist :create)))
          (delete-file (probe-file "emptyfile"))
          (and (test-setq echo-stream
                      (make-echo-stream astream closed.file.stream))
              (expect-errors (error) (print "any random thing" echo-stream))
              (close echo-stream)
              (close astream))))
;; Function To Be Tested: make-synonym-stream
;;
;; Source:        CLtL p. 329
;; Chapter 21: Streams       Section 21-2: Creating New Streams
;;
;; Created By:    Kirk Kelley
;;
;; Creation Date:  31 October 86
;;
;; Last Update:   >> n MonthName << 86
;;
;; Filed As:      {eris}<lispcore>cml>test>21-2-make-synonym-stream.test
;;
;;
;; Syntax: make-synonym-stream symbol
;;
;; Function Description: Creates and returns a "synonym stream."  Any operations on the new
stream will be performed on the stream that is then the value of the dynamic variable named by
the symbol.  If the value of the variable should change or be bound, then the synonym stream will
operate on the new stream.
;;
;;
;; Argument(s):  symbol
;;
;; Returns: stream
;;
(do-test-group "make-synonym-stream"
    (do-test make-synonym-stream-test
        (dolist (stream-name stream-io-names t)
            (test-setq original-stream
                   (open stream-name :direction :output :if-exists
                       :new-version :if-does-not-exist :create))
            (or (and (test-setq astream (make-synonym-stream 'original-stream))
                  (output-test astream))
              (print-stream-error "MAKE-SYNONYM-STREAM-output-TEST"
                            stream-name))
            ;;make sure the actual stream did not get closed.
            (or   (write "1" :stream original-stream)
                (print-stream-error
                    "MAKE-SYNONYM-STREAM-output-close-TEST"
                    stream-name))
            (close original-stream)
            (test-setq original-stream (open stream-name))
            (or (and (test-setq astream (make-synonym-stream 'original-stream))
                  (input-test astream :dont-test-for-eof t))
              (print-stream-error "MAKE-SYNONYM-STREAM-input-TEST"
                            stream-name))
            ;;make sure the actual stream did not get closed.
            (or   (string-equal "1" (ignore-errors (read original-stream)))
                (print-stream-error
                    "MAKE-SYNONYM-STREAM-input-close-TEST"
                    stream-name))
            (close original-stream)
            (delete-file (probe-file stream-name))))
    (do-test "MAKE-SYNONYM-STREAM with declare special"
        (dolist (stream-name stream-io-names t)
            (declare (special stream-name))
            (unless
                (and (with-open-file
                         (afilestream stream-name :direction :output
```

```
                                :if-exists :new-version :if-does-not-exist
                                :create)
                            (declare (special afilestream))
                            (and (test-setq astream
                                    (make-synonym-stream 'afilestream))
                                 (output-test astream)))
                        (with-open-file (afilestream stream-name)
                            (declare (special afilestream))
                            (and (test-setq astream
                                    (make-synonym-stream 'afilestream))
                                 (input-test astream))))
                    (print-stream-error
                        "MAKE-SYNONYM-STREAM with-open-file declare special TEST"
                        stream-name))
                (delete-file (probe-file stream-name)))))
;; Function To Be Tested: close
;;
;; Source:        CLtL p. 332
;; Chapter 21: Streams       Section 21-3: Operations on Streams
;;
;; Created By:    Kirk Kelley
;;
;; Creation Date:  31 October 86
;;
;; Last Update:   >> day month << 86
;;
;; Filed As:      {eris}<lispcore>cml>test>21-3-close.test
;;
;;
;; Syntax: close stream &key :abort
;;
;; Function Description: The argument must be a stream.  The stream is closed.  No further i/o
operations may be performed on it.  However, certain inquiry operations may still be performed,
and it is permissible to close an already closed stream.
;; If the :abort parameter is not nil (it defaults to nil), it indicates an abnormal termination
of the use of the stream.  An attempt is made to clean up any side effects ofhaving created the
stream in the first place.  For example, if the stream performs output to a file that was newly
created when the stream was created, then if possible the file is deleted and any previously
existing file is not superceded.
;;
;; Argument(s):  stream -- stream
;;               :abort nil / t
;;
;; Returns: t always?
;; The simple case of close is tested in all the other stream tests.  Here we test the abort
condition.
;;
(do-test-group ("close")
    (do-test "close abort delete output file"
        (dolist (stream-name stream-sink-names t)
            (let ((astream (open stream-name :direction :output :if-exists
                                 :new-version :if-does-not-exist :create)))
                (output-test astream :keep-open t)
                (unless
                    (and (close astream :abort t)
                        ;;make sure the file got deleted
                        (expect-errors (error) (open (pathname astream))))
                    (print-stream-error "close abort delete output file"
                                 stream-name)
                    (delete-file (probe-file (pathname astream)))))))
    (do-test "close abort input"
        (dolist (stream-name stream-io-names t)
            (let ((astream (open stream-name :direction :output :if-exists
                                 :new-version :if-does-not-exist :create)))
                (output-test astream))
            (let ((astream (open stream-name)))
                (unless
                    (and (close astream :abort t)
                        ;;make sure the stream got closed
                        (close (open (pathname astream) :direction :output
                                 :if-exists :append)))
                    (print-stream-error "close abort input" stream-name))
                (delete-file (probe-file (pathname astream)))))))
;; Definition To Be Tested: finish-output, force-output, and clear-output
;;
;; Source:        Xerox LIsp Manual
;; Chapter 22-3-1: Input/Output  Output to Character Streams
;;
;; Created By:    Kirk Kelley
;;
```

```
;; Creation Date:  21 November 86
;;
;; Last Update:   >> day month << 86
;;
;; Filed As:      {eris}<lispcore>cml>test>22-3-1-finish-output.test
;;
;;
;; Syntax: finish-output &optional output-stream
;;
;; Function Description: The function finish-output attempts to ensure that all output sent to
output-stream has reached its destination, and only then returns nil.  force-output initiates the
emptying of any internal buffers but returns nil without waiting for completion or
acknowledgment.  The function clear-output, on the other hand, attempts to abort any outstanding
output operation in progress in order to allow as little output as possible to continue to the
desitnation.
;;
;; Argument(s):  output-stream
;;
;; Returns: nil
;;
;; These tests just test that the functions dont break for a variety of devices.  It could be
improved by putting out a huge string or simulating a slow channel by advising \bufferedbout
(whatever its called) and do some elapsed time tests after each type of output.  Then do an
input-test to see if all the characters made it (or not in the case of clear-output).  Try
calling finish/force-output on a stream to a file server and then killing the connection.
;;
(do-test "finish-output"
    (dolist (stream-name stream-io-names t)
        (with-open-file (astream stream-name :direction :output)
                   (output-test astream :keep-open t)
                   (finish-output astream)
                   (close astream)
                   (with-open-file (astream stream-name)
                              (unless (input-test astream)
                                    (print-stream-error "finish-output"
                                                        stream-name)))))
    ;;cleanup
    (dolist (stream-name stream-io-names t)
         (delete-file stream-name)))
(do-test "clear-output"
    (dolist (stream-name stream-io-names t)
         (with-open-file (astream stream-name :direction :output)
                    (output-test astream :keep-open t)
                    (clear-output astream)
                    (close astream)))
    ;;cleanup
    (dolist (stream-name stream-io-names t)
         (delete-file stream-name)))
(do-test "force-output"
    (dolist (stream-name stream-io-names t)
        (with-open-file (astream stream-name :direction :output)
                   (output-test astream :keep-open t)
                   (force-output astream)
                   (close astream)
                   (with-open-file (astream stream-name)
                              (unless (input-test astream)
                                    (print-stream-error "finish-output"
                                                        stream-name)))))
    ;; cleanup
    (dolist (stream-name stream-io-names t)
         (delete-file stream-name)))
         STOP
```