;; Functions To Be Tested: XCL:def-define-type, XCL:defdefiner
;;
;; Source: {ERIS}<LispCore>CML>DOC>DEF-DEFINE-TYPE.TEDIT
;;   {ERIS}<LispCore>cml>doc>defdefiner.tedit
;;
;; Created By: Jim Blum
;;
;; Creation Date: Jan 9, 1987
;;
;;
;; Last Update: FEB 2/16/87 Moved into
{ERIS}<LISPCORE>TEST>FILEMANAGER>DEFDEFINE.TEST
;;
;;
;; Filed As:
{ERIS}<LISPCORE>TEST>FILEMANAGER>DEFDEFINE.TEST
;;
;; Function: defdefinetype
;;
;; Syntax: (defdefinetype name &optional description &key undefiner)
;;
;; Function Description: New kinds of file manager objects can be defined with defdefinetype.
;;
;; Aruments: NAME should be the name of the define type in plural, e.g., FUNCTIONS, VARIABLES, STRUCTURES.

;; DESCRIPTION is the documentation of this definition type, and should be a string suitable for the sentence

;; "The following <description> have not been saved on any file: "

;; The only keyword currently defined is a global "undefiner" for this definition type.
;; Each individual defdefiner is allowed to define how to "undefine" a given name,
;; but def-define-type also has a shot at removing a definition for all instances of this type, if there is such.

;; Function: def-definer
;;
;; Syntax: (def-definer name-and-options type arg-list . body)

;;

;; Function Description: DefDefiner creates macro named name that creates definitions of type type.
;; DefDefiner arranges that:
;;   -- the body will be evaluated if and only if IL:DFNFLG is not one of IL:PROP or IL:ALLPROP
;;   -- the form returned by the body will be evaluated in a context in which the file manager has been disabled
;; (so that subordinate definitions like the accessor defun's of defstruct will not be noticed by the file-manager)
;;   -- macro-calls to the new definer will return the name of the thing being defined
;; (as DEFUN, DEFMACRO, and others are defined to do)
;;
;;
;; Arguments: name-and-options is a defstruct-style name.   That is, it is either a symbol, name, or
;; a list, ie, (name (option . value) ...).
;; type must be a file-manager type previously defined using def-define-type.

;; The following options are supported:

;; (:name name-fn)
;;   name-fn should be a form acceptable as the argument to cl:function. When name-fn is
;; applied to any form representing a
;; macro-call on the new definer, it should return a Lisp value to be used as the name of the thing
;; being defined, for the purposes of
;; saving the definition with the file-manager and returning the name as the value of the
;; macro-call. name-fn should have no
;; side-effects nor should its workings depend upon any data outside of that provided as an
;; argument. The default value for name-fn is cl:second.

;; (:prototype-fn defn-fn)

;;   defn-fn should be a form acceptable as the argument to cl:function. When defn-fn is applied to any Lisp value, it should
;; return either NIL or a form that, when evaluated, would create a dummy definition of type type named by that Lisp value.
;; This function can be used by SEdit to provide dummy definitions for names that have no other definition.
;; For example, the defn-fn for DEFUN might be
;;
;;

```
;;              (lambda (name)
;;                      (and  (symbolp name)
;;                              '(defun ,name ("args") "body")))
;; The default value for defn-fn is
;;              (lambda (name) nil)


;; (:undefiner function)
;;       a function which will clear any definition of the name given to it. This
is an "incremental" undefiner, in that when DELDEF
;; is given the type, it calls all undefiners for all of the types. The undefiner
function should be undoable, if at all possible.



;;
;; Returns:   name of definer if successful or, error if not.
;;

;; ----------------------------------------------------------------------------

;;    Use DEF-DEFINE-TYPE to define a new file manager type.
;;    Give it a recognisable description string and an undefiner.
;;    The undefiner will take a name and remove a certain property
;;    (call it PROPERTY-ONE) from that name.
(do-test "define new file manager type"
        (and (def-define-type definer-tests "Definer Tests"
                :undefiner (lambda (name)
                                (remprop name 'property-one)))))

;;    Use DEFDEFINER to define a definer of the new type.
;;    Use the :NAME option in some non-trivial way to make a new
;;    name.  The effect of the definer will be to put T onto the
;;    properties PROPERTY-ONE and PROPERTY-TWO of the name.  Use
;;    the :UNDEFINER option to remove only PROPERTY-TWO from the
;;    name.  In conjunction with the undefiner on the type, this
;;    will clear the whole effect of the definer.

(do-test "define a new definer of the new type"
        (and (defdefiner (def-test-one
                        (:name (lambda (whole)
                           (intern (concatenate 'string
                                                "FOO--"
                                                (string (second whole))))))
                        (:undefiner (lambda (name)
                                    (remprop name 'property-two))))
                definer-tests
```

```
                    (proto-name value-one value-two)
             (let ((name (intern (concatenate 'string "FOO--" (string proto-
name)))))
             '(progn (setf (get ',name 'property-one) ',value-one)
              (setf (get ',name 'property-two) ',value-two))))))

;;   Also use DEFDEFINER to definer another definer for the new
;;   type using neither :NAME nor :UNDEFINER.  The effect of this
;;   definer would be to only give the name the property PROPERTY-ONE.

(do-test "use DEFDEFINER to definer another definer for the newtype
using neither :NAME nor :UNDEFINER"
  (and (defdefiner def-test-two definer-tests (name value-one)
       '(setf (get ',name 'property-one) ',value-one))))

;;   With DFNFLG bound to NIL, use both definers to make objects
;;   of the new type.  These definitions should take effect.  Use
;;   SEdit-style comments to test that they get properly stripped.

(do-test "make objects of the new type which take effect"
 (and (let ((il:dfnflg nil))
   (declare (special il:dfnflg))

   (def-test-one (il:* il:|;| "An SEdit-style comment")
            one-1
            (il:* il:|;;| "An SEdit-style comment")
            1
            (il:* il:|;;;| "An SEdit-style comment")
            2)

   (def-test-two (il:* il:|;| "An SEdit-style comment")
            two-1
            (il:* il:|;;| "An SEdit-style comment")
            (il:* il:|;;;| "An SEdit-style comment")
            3))))

;;   With DFNFLG bound to PROP, again use both definers.  Neither
;;   of these should take effect.

(do-test "make objects of the new type with DFNFLG = PROP which should
not take effect"
 (and (let ((il:dfnflg 'il:prop))
   (declare (special il:dfnflg))

   (def-test-one (il:* il:|;| "An SEdit-style comment")
            one-2
            (il:* il:|;;| "An SEdit-style comment")
            1
            (il:* il:|;;;| "An SEdit-style comment")
            2)

   (def-test-two (il:* il:|;| "An SEdit-style comment")
            two-2
            (il:* il:|;;| "An SEdit-style comment")
            (il:* il:|;;;| "An SEdit-style comment")
            3))))

;;    With DFNFLG bound to ALLPROP, once again use both definers.
;;    Neither of these should take effect either.

(do-test "make objects of the new type with DFNFLG bound to ALLPROP
which should not take effect"
 (and (let ((il:dfnflg 'il:allprop))
   (declare (special il:dfnflg))

   (def-test-one (il:* il:|;| "An SEdit-style comment")
            one-3
            (il:* il:|;;| "An SEdit-style comment")
            1
            (il:* il:|;;;| "An SEdit-style comment")
            2)

   (def-test-two (il:* il:|;| "An SEdit-style comment")
            two-3
            (il:* il:|;;| "An SEdit-style comment")
            (il:* il:|;;;| "An SEdit-style comment")
            3))))

;;     Check that the define-type, both definers, and all six uses
;;    of the definers got marked as changed.

(do-test "Check that the define-type, both definers, and all six uses of the
definers got marked as changed"
(and (flet ((is-changed (name type)
             (let ((changes-var (first (find type il:prettytypelst
                                             :key 'second))))
               (member name (symbol-value changes-var)))))
   (and (is-changed 'definer-tests 'il:define-types)
       (is-changed 'def-test-one  'il:functions)
       (is-changed 'def-test-two  'il:functions)
```

```
        (is-changed 'foo--one-1    'definer-tests)
        (is-changed 'foo--one-2    'definer-tests)
        (is-changed 'foo--one-3    'definer-tests)
        (is-changed 'two-1         'definer-tests)
        (is-changed 'two-2         'definer-tests)
        (is-changed 'two-3         'definer-tests)))))

;;    Check that the define-type got installed with the
;;    right description name.

(do-test "Check that the define-type got installed with the right description
name"
   (equal "Definer Tests" (third (find 'definer-tests il:prettytypelst
                                                         :key 'second))))

;;   Check that all six uses of the definers got putdef'd correctly.


(do-test "Check that all six uses of the definers got putdef'd correctly"
  (and (equal (il:getdef 'foo--one-1 'definer-tests)
                   '(def-test-one (il:* il:|;| "An SEdit-style comment")
                                  one-1
                                  (il:* il:|;;| "An SEdit-style comment")
                                  1
                                  (il:* il:|;;;| "An SEdit-style comment")
                                  2))
         (equal (il:getdef 'two-1 'definer-tests)
                   '(def-test-two (il:* il:|;| "An SEdit-style comment")
                                  two-1
                                  (il:* il:|;;| "An SEdit-style comment")
                                  (il:* il:|;;;| "An SEdit-style comment")
                                  3))
        (equal (il:getdef 'foo--one-2 'definer-tests)
                   '(def-test-one (il:* il:|;| "An SEdit-style comment")
                                  one-2
                                  (il:* il:|;;| "An SEdit-style comment")
                                  1
                                  (il:* il:|;;;| "An SEdit-style comment")
                                  2))
        (equal (il:getdef 'two-2 'definer-tests)
                   '(def-test-two (il:* il:|;| "An SEdit-style comment")
                                  two-2
                                  (il:* il:|;;| "An SEdit-style comment")
                                  (il:* il:|;;;| "An SEdit-style comment")
                                  3))
```

```
    (equal (il:getdef ’foo--one-3 ’definer-tests)
            ’(def-test-one (il:* il:|;| "An SEdit-style comment")
                    one-3
                    (il:* il:|;;| "An SEdit-style comment")
                    1
                    (il:* il:|;;;| "An SEdit-style comment")
                    2))
  (equal (il:getdef ’two-3 ’definer-tests)
            ’(def-test-two (il:* il:|;| "An SEdit-style comment")
                    two-3
                    (il:* il:|;;| "An SEdit-style comment")
                    (il:* il:|;;;| "An SEdit-style comment")
                    3))))

;; Check that only the first two uses took effect.

(do-test "Check that only the first two uses took effect"
  (and (= 1 (get ’foo--one-1 ’property-one))
     (= 2 (get ’foo--one-1 ’property-two))
     (= 3 (get ’two-1 ’property-one))
     (null (get ’two-1 ’property-two))
     (null (get ’foo--one-2 ’property-one))
     (null (get ’foo--one-2 ’property-two))
     (null (get ’two-2 ’property-one))
     (null (get ’two-2 ’property-two))
     (null (get ’foo--one-3 ’property-one))
     (null (get ’foo--one-3 ’property-two))
     (null (get ’two-3 ’property-one))
     (null (get ’two-3 ’property-two))))

;; Use DELDEF on each of the first two uses and check that all of the
appropriate REMPROP’s
;; happened.  Also check that those two uses are no longer marked as
changed and that HASDEF returns NIL for both.

(do-test "DELDEF test"
  (and (il:deldef ’foo--one-1 ’definer-tests)
     (il:deldef ’two-1 ’definer-tests)
     (null (get ’foo--one-1 ’property-one))
     ; (null (get ’foo--one-1 ’property-two))
     (null (get ’two-1 ’property-one))
     (null (get ’two-1 ’property-two))
     ; (null (il:hasdef ’foo--one-1 ’definer-tests))
     ; (null (il:hasdef ’two-1 ’definer-tests))))
STOP
```