

```
:: FILE MANAGER - tests for Common Lisp FILE COMMANDS
;; "FUNCTIONS", "VARIABLES", and "STRUCTURES".
```

```
:: Functions To Be Tested: MAKEFILE, IL:LOAD, CL:LOAD MARKASCHANGED,
;; UNMARKASCHANGED,
;; ADDTOFILE, GETDEF, PUTDEF, HASDEF,
;; COPYDEF, DELDEF
;; RENAME, BCOMPL, BRECOMPILE, COMPILE-FILE
```

```
;; Source: KOTO IRM, [NOTE: Can't find any LYRIC documentation on this]
```

```
;; ;; Created By: Jim Blum
```

```
;;
```

```
;; Creation Date: Jan 9, 1987
```

```
;;
```

```
;; Last Update: Jan 21, 1987
```

```
;;
```

```
;; FEB 16, 1987 - MOVED Into
```

```
{ERIS}<LISPCORE>TEST>FILEMANAGER>CMLFILEMANAGER.TEST
```

```
;;
```

```
;; Filed As: {ERIS}<LISPCORE>TEST>FILEMANAGER>CMLFILEMANAGER.TEST
```

```
;;
```

```
;; 3 new FILE MANGAGER TYPES have been added for COMMON LISP -
;; FUNCTIONS, VARIABLES, & STRUCTURES
;; The tests below test the FILE MANAGER to see if these are being handled correctly
```

```
(do-test "load a test file and make sure it gets noticed"
  (do nil ((null (il:delfile '{DSK}testfile)))) ; delete any old local versions
  (do nil ((null (il:delfile '{DSK}testfile.lcom))))
  (do nil ((null (il:delfile '{DSK}testfile.dfasl))))
  (setq il:dfnflg nil) ; make sure DFNFLG is set to nil
  (il:smashfilecoms 'testfile)
  (il:deldef 'test-function 'il:functions)
  (il:deldef 'test-macro 'il:functions)
  (makunbound 'test-variable)
  (defstruct test-structure) ; redefine test-structure to dummy def
  (il:setproplist 'il:testfile nil) ; remove entire property list
  (IL:load '{eris}<lispcore>test>filemanager>testfile)
  (il:putprop 'il:testfile 'il:makefile-environment '(:readtable "XCL" :package "XCL-TEST"))
  (member 'il:testfile il:filelst)
)

(do-test "define a new function and add to the COMS file"
  (and (eq 'test-function (defun test-function))
    (member 'test-function il:changedfunctionslist)
    (eq 'il:testfile (il:addtofile 'test-function 'il:functions 'il:testfile))
  )
)

(do-test "define a new macro and add to the COMS file"
  (and (eq 'test-macro (defmacro test-macro nil :test))
    (member 'test-macro il:changedfunctionslist)
    (eq 'il:testfile
      (il:addtofile 'test-macro 'il:functions 'il:testfile))
  )
)

(do-test "Define a structure and make sure it gets noticed"
  (and (defstruct test-structure x y)
    (member 'test-structure il:changedstructureslist)
    (eq 'il:testfile
      (il:addtofile 'test-structure 'il:structures 'il:testfile))
  )
)
```

```

)
)

(do-test "Define and set a variable and add to the COMS file"
  (and (defvar test-variable (make-test-structure :x 1 :y 2))
        (member 'test-variable il:changedvariables1st)
        (eq 'il:testfile
            (il:addtofile 'test-variable 'il:variables 'il:testfile))
        )
)

(do-test "MAKEFILE, DELDEF test"
  (and (il:makefile '{DSK}testfile)
        (il:deldef 'test-function 'il:functions)
        (il:deldef 'test-macro 'il:functions)
        (il:deldef 'test-structure 'il:structures)
        (null (il:hasdef 'test-function))
        (null (il:hasdef 'test-macro))
        (makunbound 'test-variable 'il:variables)
        (null (boundp 'test-variable))
        )
)

(do-test "Reload test"
  (and (makunbound 'test-variable)
        (null (boundp 'test-variable))
        (il:load '{DSK}testfile)
        (eql (test-structure-x test-variable) 1)
        (eql (test-structure-y test-variable) 2)
        (equal (il:getdef 'test-function 'il:functions) '(defun test-function))
        (eq (test-macro) :test)
        )
)

(do-test "edit the function definition and see if marked as changed"
  (and (il:putdef 'test-function 'il:functions (append (il:getdef 'test-function
' il:functions) '(a b) (+ a b))))
        (member 'test-function il:changedfunctions1st)
        (equal (il:getdef 'test-function 'il:functions) '(defun test-function (a b) (+ a b)))
        ) ; and
)

(do-test "edit the macro definition and see if marked as changed"
  (and (il:putdef 'test-macro 'il:functions
        (subst ' :new-test ' :test
              (il:getdef 'test-macro 'il:functions))
        )
        (member 'test-macro il:changedfunctions1st)
        )
)

(do-test "edit the structure and see if it gets marked as changed"
  (defstruct test-structure x y z)
  (member 'test-structure il:changedstructures1st)
)

(do-test "edit the variable def and see if it gets marked as changed"
  (defvar test-variable (make-test-structure :x 3 :y 4 :z 5))
  (member 'test-variable il:changedvariables1st)
)

(do-test "makefile, load and execute the new version"
  (and (il:makefile '{DSK}testfile)
        (il:deldef 'test-function 'il:functions)
        (null (il:hasdef 'test-function))
        (il:deldef 'test-macro 'il:functions)
        (null (il:hasdef 'test-macro))
        (makunbound 'test-variable)
        (defstruct test-structure) ; redefine to dummy defstruct
        (equal (il:getdef 'test-structure 'il:structures)
              '(defstruct test-structure)
              )
        (il:load '{DSK}testfile)
        (eql (test-function 3 2) 5)
        (equal (test-macro) :new-test)
        )
)

```

```

    (eql (test-structure-z test-variable) 5)
  )
)

(do-test "rename the function, makefile, reload and execute"
  (setq il:defaultrenamemethod '(il:editcallers))
  (il:rename 'test-function 'new-function 'il:functions '{DSK}testfile)
  (and (null (il:hasdef 'test-function))
        (il:hasdef 'new-function)
        (eql (new-function 2 3) 5))
  )
)

(do-test "copydef"
  (and (il:copydef 'new-function 'newer-function 'il:functions)
        (il:hasdef 'newer-function)
        (member 'newer-function il:changedfunctionslist)
  ) ; and
)

(do-test "test dfnflg set to PROP and ALLPROP"
  (flet ((dfnflg-check (functions-def cell-def)
          (declare (special il:dfnflg))
          (and (equal (il:getdef 'new-function 'il:functions)
                      functions-def ; make sure there is a new functions def
                    )
                (member 'new-function il:changedfunctionslist) ; test
                    marked as changed
                    (equal (symbol-function 'new-function)
                          cell-def ; make sure it hasn't taken effect
                    )
                )
          ) ; and
  ))
  (il:addtofile 'new-function 'il:functions 'il:testfile)
  (and (let ((il:dfnflg 'il:prop))
        (declare (special il:dfnflg))
        (defun new-function (a b) (- a b)); redefine the function
        (dfnflg-check '(defun new-function (a b) (- a b)) '(lambda (a b) (block
new-function (+ a b))))
        (il:makefile '{DSK}testfile)
        (true (setq il:dfnflg nil))
        (defun new-function) ; redefine the function in both places
        (defstruct test-structure) ;redefine test-structure
        (il:load '{DSK}testfile)
        (dfnflg-check '(defun new-function (a b) (- a b)) '(lambda (a b) (block
new-function (- a b))))
        (defun new-function) ; redefine the function
        (il:load '{DSK}testfile 'il:prop) ; load with PROP
        (dfnflg-check '(defun new-function (a b) (- a b)) '(lambda nil (block new-
function))))
        (equal (il:getdef 'test-structure 'il:structures)
                '(defstruct test-structure x y z)
        )
        ) ; let
  (let ((il:dfnflg 'il:allprop)) ; now check dfnflg = ALLPROP
        (declare (special il:dfnflg))
        (defun new-function (a b) (* a b)) ; redefine the function
        (dfnflg-check '(defun new-function (a b) (* a b)) '(lambda (a b) (block
new-function (+ a b))))
        (defstruct test-structure a b c)
        (il:makefile '{DSK}testfile)
        (true (setq il:dfnflg nil))
        (defun new-function) ; redefine the function in both places
        (defstruct test-structure)
        (il:load '{DSK}testfile)
        (dfnflg-check '(defun new-function (a b) (* a b)) '(lambda (a b) (block
new-function (* a b))))
        (defun new-function) ; redefine the function
        (il:load '{DSK}testfile 'il:allprop) ; load with PROP
        (dfnflg-check '(defun new-function (a b) (* a b)) '(lambda nil (block new-
function))))
        (equal (il:getdef 'test-structure 'il:structures)
                '(defstruct test-structure a b c)
        )
        ) ; let

```

```

    ) ; and
  ) ; flet
)

(do-test "test BCOMPL"
  (and
    (defun new-function)
    (defmacro test-macro)
    (defvar test-variable 1)
    (il:delfromfile 'test-structure 'il:structures 'il:testfile) ; get rid of
structure as this will cause a problem later
    (il:defineq (test-fns (a b) (+ a b))) ; define a fns
    (il:addtofile 'test-fns 'il:fns 'il:testfile)
    (il:makefile '{DSK}testfile)
    (il:bcompl '{DSK}testfile nil nil 'il:ST)
    (true (il:smashfilecoms 'testfile))
    (il:deldef 'test-fns 'il:fns) ; delete fns definition
    (il:deldef 'new-function 'il:functions)
    (il:deldef 'test-macro 'il:functions)
    (makunbound 'test-variable)
    (il:load '{DSK}testfile.lcom) ; reload file
    (eq (test-fns 3 4) 7) ; make sure fns got loaded
    (equal (il:getdef 'new-function 'il:functions)
            '(defun new-function)
            ) ; make sure functions and macros didn't compile
    (equal (il:getdef 'test-macro 'il:functions)
            '(defmacro test-macro)
            )
  )
)

(do-test "test makefile, brecompile, & load in a different package environment"
  (il:defineq (test-fns (a b) (- a b))) ; redefine fns
  (il:putprop 'il:testfile 'il:makefile-environment '(:readtable "XCL" :package "XCL-USER"))
  (il:makefile '{DSK}testfile)
  (il:brecompile '{dsk}testfile)
  (il:smashfilecoms 'testfile)
  (il:deldef 'new-function 'il:functions)
  (il:deldef 'test-macro 'il:functions)
  (il:deldef 'test-fns 'il:fns) ; delete fns definition
  (makunbound 'test-variable)
  (and (il:load '{DSK}testfile.lcom)
        (eq (test-fns 4 3) 1)
        (equal (il:getdef 'new-function 'il:functions)
                '(defun new-function)
                )
        )
    (equal (il:getdef 'test-macro 'il:functions)
            '(defmacro test-macro)
            )
    )
  (eql test-variable 1)
)

(do-test "test COMPILE-FILE new compiler"
  (and
    (il:putprop 'il:testfile 'il:makefile-environment '(:readtable "XCL" :package
"XCL-TEST"))
    (il:putprop 'il:testfile 'il:filetype 'compile-file)
    (il:defineq (test-fns (a b) (* a b))) ; redefine the fns
    (defun new-function (a b) (* a b))
    (defmacro test-macro nil :test)
    (defvar test-variable 1)
    (eq 'test-macro (defmacro test-macro nil :test))
    (il:makefile '{DSK}testfile)
    (compile-file 'testfile)
    (true (il:smashfilecoms 'testfile))
    (il:deldef 'new-function 'il:functions)
    (il:deldef 'test-macro 'il:functions)
    (il:deldef 'test-fns 'il:fns) ; delete fns definition
    (makunbound 'test-variable)
    (il:load '{DSK}testfile.dfas1)
    (eql (test-fns 4 3) 12)
    (eq (test-macro) :test)
    (eql (new-function 4 3) 12)
    (true (il:smashfilecoms 'testfile))
    (il:deldef 'new-function 'il:functions)
    (il:deldef 'test-macro 'il:functions)
    (il:deldef 'test-fns 'il:fns) ; delete fns definition
  )
)

```

```

(makunbound 'test-variable)
(cl:load '{DSK}testfile.dfasl) ; test CL LOAD
(eql (test-fns 4 3) 12)
(eq (test-macro) :test)
(eql (new-function 4 3) 12)
)
)

(do-test "test makefile, compile-file, & load in a different package environment"
  (and
    (il:defineq (test-fns (a b)(- a b))) ; redefine fns
    (defun new-function (a b)(- a b))
    (defmacro test-macro nil :new-test)
    (defvar test-variable 2)
    (il:putprop 'il:testfile 'il:makefile-environment '(:readtable "XCL" :package
"XCL-USER"))
    (il:makefile '{DSK}testfile)
    (compile-file '{DSK}testfile)
    (il:smashfilecoms 'testfile)
    (il:deldef 'new-function 'il:functions)
    (il:deldef 'test-macro 'il:functions)
    (il:deldef 'test-fns 'il:fns) ; delete fns definition
    (makunbound 'test-variable)
    (il:load '{DSK}testfile.dfasl)
    (eq (test-fns 4 3) 1)
    (eql (new-function 4 3) 1)
    (eql test-variable 2)
    (il:smashfilecoms 'testfile)
    (il:deldef 'new-function 'il:functions)
    (il:deldef 'test-macro 'il:functions)
    (il:deldef 'test-fns 'il:fns) ; delete fns definition
    (makunbound 'test-variable)
    (cl:load '{DSK}testfile.dfasl)
    (eq (test-fns 4 3) 1)
    (eql (new-function 4 3) 1)
    (eql test-variable 2)
  )
)

(do-test "delete test environment items"
  (il:deldef 'test-function 'il:functions)
  (il:deldef 'new-function 'il:functions)
  (il:deldef 'newer-function 'il:functions)
  (il:smashfilecoms 'il:testfile)
  (do nil ((null (il:delfile '{DSK}testfile)))) ; delete all local files
  (do nil ((null (il:delfile '{DSK}testfile.lcom))))
  (do nil ((null (il:delfile '{DSK}testfile.dfasl))))
  (setq il:filelst (remove 'il:testfile il:filelst))
  (setq il:loadfilelst (remove-if #'(lambda (a)
"TESTFILE" (pathname-name a)) il:loadedfilelst))
  (il:setproplist 'il:testfile nil)
  (il:updatefiles)
  (true)
)

STOP

```