Medley

What File to Load

Load the file {MEDLEY}internal>DO-TEST.DFASL

All the symbols mentioned in this document are in both the IL: and XCL-TEST: packages, unless otherwise noted.

Main Testing Entry Points

`(DEFTEST` *name&options forms*`)` [Definer]

This is the definer for tests, allowing them to be saved on file-managed files. The test succeeds if the final *form* returns a non-NIL result. If *name&options* isn't a list, then it's just the name which can be a symbol or string; symbols are preferred for DEFTEST tests. If you specify options, the CAR of *name&options* is the name. If you specify :COMPILED in *name&options, the test will run only when it has been compiled.* Since this test is stored as structure rather than as plain text, any symbols will be package-qualified appropriately. If a test fails or an error occurs during evaluation, a message is printed to '`*ERROR-OUTPUT*`'.

Unless you have DFNFLG set to PROP, the act of defining a test also causes it to be run (so you'll see if your test fails right away).

Examples:

```
(DEFTEST ISSUE-1000  ;use issue number in test name
   (= 3 (+ 1 2)))

(DEFTEST (+-OPT :COMPILED) ; A test of the compiler, only makes sense to run compiled.
   (= 3 (+ 1 1 1)))        ; Checking that +'s optimizer does the right thing.

(DEFTEST (MS-TEST :INTERPRETED)   ; A test of Masterscope, only makes sense interpreted.
   (TEST-DEFUN FN (X) (FOO X))
   (\. IS FOO CALLED BY FN))
```

`(DEFTESTGROUP` *name&options forms*`)` [Definer]

This is the definer for groups of tests, allowing them to be saved on file-managed files. For associating a group of tests. For instance, a group of tests may all require the same setup and cleanup. If there are any options (see below) then the `CAR` of *name&options* is the name and the `CDR` is a keyword/value list. All *forms* must be `DEFTEST` or `DO-TEST` forms.

Unless you have DFNFLG set to PROP, the act of defining a test group also causes it to be run (so you'll see if your tests fail right away).

`:before` allows for a setup form for a group of tests.

`:after` allows a form to be run after the tests without affecting results.

For example, a `DEFTESTGROUP` using all its features is:

```
(DEFTESTGROUP
   (UNWIND-OPCODE-TESTS
     :BEFORE (progn (before-form-1) (before-form-2)...)
     :AFTER  (progn (after-form-1) (after-form-2))
   )
   (DEFTEST "first test" ....)
   (DEFTEST "second test" ....)
)
```

Functions You'll Find Useful When Building Tests

`(EXPECT-ERRORS (`*error-types*`)` *forms*`)` [Macro]

Error-types is a list of errors that may occur while executing the *forms*. If one of the listed errors occurs, EXPECT-ERRORS returns (values t error-that-occurred), otherwise NIL. If all you want to do is make sure that an error is signalled somewhere in the test, you can specify an *error-types* of T. Normal use of this form is:

```
(DEFTEST ERROR-CHECK
   (EXPECT-ERRORS (T)
         (THIS-FORM 'SHOULD 'ERROR)))

(DEFTEST (+-DETECTS-NILS :INTERPRETED)
   (EXPECT-ERRORS (XCL:TYPE-MISMATCH)
         (+ 3 NIL)))
```

(TEST-SETQ *Variable Value*)
(TEST-DEFUN *name* (*arglist*) *forms*)
(TEST-DEFMACRO *name* (*arglist*) *forms*) [Macros]

These work like SETQ, DEFUN, and DEFMACRO, except that if they are executed within a DEFTEST or DEFTESTGROUP, their effects are manually undone (old values are saved and then restored) upon leaving the test. Use these in :BEFORE forms that a whole group of DEFTESTs want to see. **DON'T** use TEST-SETQ on locally-bound variables or in loops.

Commands and Functions for Running Tests

run *Test-name* [EXEC Command]

Once *Test-name* has been defined using DEFTEST or DEFTESTGROUP, you can run the test with the run command.

(DO-TEST-FILE *filename*)

Reads and executes a file of tests. All forms in the file are read before any are executed. The file should be clear text (clearput in TEdit) and terminate with a STOP. The format for test names is

‘‘‘

Chap#[-sec#[-subsec#]]-comment.TEST

```
(DO-ALL-TESTS &key (results *test-batch-results*)
                   (patterns *test-file-pattern*)
                   (sysout-type nil)
                   (resume nil))
```

‘‘‘

Calls DO-TEST-FILE on each file that matches *patterns,* which is a list of directory patterns, and prints the results to a new version of a file named *results*. If *results* is T, results are printed to the window where DO-ALL-TESTS is running. The header of the results file is a message of the date and time the tests are being run and the MAKESYSDATE of the sysout; if *sysout-type* is supplied, a line for it goes out too. If *resume* is non-NIL, DO-ALL-TESTS attempts to resume an interrupted test sequence, appending the results onto the latest version of *results*.

‘*TEST-MODE*‘ [Variable]

Default is :batch, which means to report test failures and errors on *ERROR-OUTPUT* (which is usually a file), and continue. Other values possible are: :interactive which means to print a message before running each test, print another message for test failures, and produce a break window on errors. :batch-verbose which means to generate all the messages of :interactive and do not break on errors.

‘*TEST-BATCH-RESULTS*‘ [Variable]

Defaults to "{MEDLEY}tmp>test>test-results"

‘*TEST-FILE-PATTERN*‘ [Variable]

Defaults to "{MEDLEY}internal>test>*.TEST "

`*TEST-COMPILE*` [Variable]

If this switch is non-nil, DO-TEST compiles its forms before testing them. DO-ALL-TESTS will print a message in its header if this switch is on.

`*ALL-FILES-REMAINING*` [Variable]

While DO-ALL-TESTS is running, this variable contains a list of all the files remaining to be processed; files are removed from it AFTER they are read and executed. To restart a test run that somehow crashes the test driver, first clean up whatever blew up the run (if necessary, dump *ALL-FILES-REMAINING* to a file and get a new sysout), then do
(DO-ALL-TESTS :RESUME T [:RESULTS "wherever"]).

internal Functions

(DO-TEST *name&options forms*) [Macro]

This is the obsolete, plain-test-file testing macro; it is still around so that old tests work (and because DEFTEST uses it). A test succeeds if the final *form* returns a non-nil result. If *name&options* isn't a list, then it's just the name which can be an atom or string; strings are preferred. If you specify options, the CAR of *name&options* is the name. If you specify :COMPILED in *name&options, the test will run only when it has been compiled.* Forms are presumed to be read with the Common Lisp reader in package XCL-TEST, which uses LISP and XCL. If a test fails or an error occurs during evaluation, a message is printed to *ERROR-OUTPUT*.

(DO-TEST-GROUP *name&options forms*) [Macro]

This is the obsolete, plain-test-file testing macro; it is still around so that old tests work (and because DEFTESTGROUP uses it). For associating a group of tests. For instance, a group of tests may all require the same setup and cleanup. If there are any options (see below) then the CAR of *name&options* is the name and the CDR is a keyword/value list. All *forms* must be DO-TEST forms.

:before allows for a setup form for a group of tests.

:after allows a form to be run after the tests without affecting results.

An example of a DO-TEST-GROUP using all its features is:
```
(DO-TEST-GROUP
      ("a test group"
      :BEFORE (progn (before-form-1) (before-form-2)...)
      :AFTER (progn (after-form-1) (after-form-2))
      )
      (DO-TEST "first test" ....)
      (DO-TEST "second test" ....)
)
```

(CL-READFILE *filename*)

Reads all forms in *filename* and returns a list of them. This function is used by DO-TEST-FILE to read test files; test writers who want to see if their files are syntactically valid should first see if CL-READFILE will read them, then see if DO-TEST-FILE will execute them.

(MUNG-TEST-FILES *filepattern* &key (*compiler* 'compile-file)
(*startinglist* NIL))

Compiles test files so they can be run by just loading them. Compiles all files matching *filepattern* (which is fed to directory) using *compiler* and writes them out to the directory they came from with an extension appropriate to *compiler*. If you want to explicitly specify the list of files to compile, hand a list of pathnames to *startinglist*. Prints an error message for files that fail to compile. You have to use this function (instead of just compiling the test files) because it prefaces the test files with

```
```

```
(in-package "XCL-TEST")
(setq *test-file-name* "NAME-OF-FILE")
```

so the compiler will read them properly and the files will know their names for error reporting purposes. **NOTE**: tests that fail should not be compiled; the resulting compiled code may not be a valid test.