THE INTERLISP-D TESTING SYSTEM - USER GUIDE

This document should be used as a guide for users of the testing system, and it assumes that the reading of "The InterlispD Testing System" document.

LOADING THE TESTING SYSTEM

The testing system resides on the directory {Eris}<test>tools>. To load the testing system, LOAD the file TESTERLOADER. This file contains the correct loading sequence for the testing system files.

FILES:

TESTER.DCOM - Contains the main part of the tester program.

TESTERVARS - Contains most of the tester global variables.

TEST-REMOTE-EVAL.DCOM -Contains the functions for executing remote eval.

RANDOM-GENERATOR.DCOM - Contains functions and variables for random generation.

VARBROWSER.DCOM - A user package for manipulating the programs global variables.

THERMOMETER.DCOM - A user package for displaying the progress of a program in execution.

INTERLISPD-SYSTEM.CONCEPTSPACE - The file contains the global concept space of the system.

EXECUTING TESTS

Usually execution of tests will be done through the Concept Space Browser. The basic function which is called by the browser is:

(TEST.PERFORM-TEST TEST TIMES LOCATION TRACE-FILE TRACE-MODE) [function]

Only the first argument is necessary. TEST must be of type TEST. TIMES is the number of times that the test will be performed and is defaulted to the value of the field TIMES of TEST. If this value is NIL it will be executed once. LOCATION can be either the atom "Local" or the atom "Remote" and is defaulted to the value of the global variable TEST.DEFAULT-LOCATION. TRACE-FILE is the name of the file on which the test execution process should be traced. The default name is the value of the global variable TEST.TRACE-FILE-NAME. TRACE-MODE can be the atom "On" or the atom "Off". It is defaulted to the value of the global vcariable TEST.DEFAULT-TRACE-MODE. When TRACE-MODE is "On" every testing step will be recorded on the trace file as soon as possible. The tracing will be done in a "careful" way - as soon as a new information is available (The input was generated, the result from the tested expression evaluation is returned, etc.) , the trace file will be opened the information will be written and the file will be cvlosed. If the trace mode is Off, the function will check at the end of each iteration of the function to see if the outcome was a failure and only in such a case it will write it down on the trace file.

There are two ways in which the PERFORM-TEST function can iterate. One way is if the TIMES argument is greater then one (or the times field of the test is greater then one). The other way is when the input expression is a list starts with the atom SYSTEMATIC. In such a case each of the elemnets of the CDR of the list will be evaluated. The tester expect them to produce sets (lists) which are the ranges of the arguments. It will then collect all the possible combinations of the elements in these finite ranges and perform the test on each of these combinations.

The function returns the name of the trace file.

(TEST.HARDCOPY-TRACE-FILE TRACE-FILE OUTPUT-FILE FAILURES-ONLY)

The default for TRACE-FILE is the value of TEST.TRACE-FILE-NAME. The default for OUTPUT-FILE is the value of TEST.DEFAULT-HARDCOPY-DEVICE (usually {LPT}). The trace file is written in a way that is hard for reading. This function prints the trace file in a "pretty" way. if FAILURES-ONLY is non-NIL only the trace of tests that failed will be printed out.

Manipulating Concept Spaces

Concept spaces are stored in files, each concept space in its own file. usually the name of the file can be XXX.CONCEPTSPACE where XXX is the name of the concept space. This name can be retrieved by calling

(TEST.CANONICAL-CONCEPT-SPACE-FILE-NAME CONCEPT-SPACE-NAME)

Returns XXX.CONCEPTSPACE where XXX is the value of CONCEPT-SPACE-NAME.

A concept space is of type CONCEPTSPACE which is record with the fields CONCEPTSPACENAME ROOTCONCEPT and CONCEPTLIST. ROOTCONCEPT is the name of the root concept. CONCEPT list is a list of concepts. A concept is an instance of the record CONCEPT which has the fields CONCEPTNAME, TESTS, SUBCONCEPTS and SUPERCONCEPTS. SUBCONCEPTS and SUPERCONCEPT are NAMES of concepts. TESTS is a list of tests ids. Usually the initial concept space will be created by the function

(TEST.CREATE-NEW-CONCEPT-SPACE CONCEPT-SPACE-NAME ROOT-CONCEPT-NAME)

which returns an instance of CONCEPTSPACE with one concept. The rest of the concept space will be most conveniently built using the Concept Space Browser.

The system maintains a global list of the concept spaces that are "known" to the system. It is convenient to work with concept spaces that appear in that list since this will enable the user to perform certain operations on the concept space using the background menu. The global list is stored in the global variable TEST.CONCEPT-SPACES. This variable can be manipulated directly or by calling the function

(TEST.ADD-CONCEPT-SPACE-TO-CONCEPT-SPACES CONCEPTSPACE)

If a concept space with the same name is already in the list, it will be removed and the new one will be added. The function

(TEST.GET-CONCEPT-SPACE NAME)

Returns the concept space with name NAME (if there is one in TEST.CONCEPT-SPACES).

Concept spaces that are on the TEST.CONCEPT-SPACES can be stored by calling the function

(TEST.STORE-CONCEPT-SPACE CONCEPTSPACENAME)

The function will prompt for a file name, suggesting the canonical name. If the concept space was loaded before through the TEST.LOAD-CONCEPT-SPACE function then the candidate file name will be the same as the one that it was loaded from (but with higher version). The user can also choose the subitem "Store Concept Space" from the background menu.

(TEST.LOAD-CONCEPT-SPACE CONCEPT-SPACE-FILE-NAME)

Loads the concept space from the specified file, adds it to TEST.CONCEPT-SPACES and keeps the file name in the property CONCEPTFILE of the concept space name.

Building and manipulating tests

(TEST.CREATE-NEW-TEST) [function]

Creates an instance of the record TEST with the default values which are obtained by calling the function TEST.GET-DEFAULT-FIELD-VALUE. The new instance is then added "officially" to the world by calling TEST.ADD-TEST . Then the test editor is called on the new test to allow the user insert values to its fields. It does NOT store the test in a file, and this should be done later by calling either TEST.STORE-TEST or TEST.STORE-CHANGED-TESTS. The test, after created is not assigned to any concept, and it is recommended to assign it as soon as possible to at list one concept.

(TEST.ADD-TEST TEST-RECORD) [function]

Assigns a new ID to the test record instance by calling TEST.GET-AND-INCREASE-NEXT-TESTID and adds the test to the list of loaded tests.

(TEST.EDIT-TEST TEST) [function]

TEST should be a test or a test number. If the global flag TEST.OBTAIN-LOCk-WHEN-EDIT is non NIL the function will try to obtain write lock on the test. If it fails to obtains such a lock it will exit without editing. The user can get the name of the locking users and automatically send release requests as described in the "accessing the database" section.

The test inspector will be called and the test will be marked as changed.

(TEST.GET-TEST TEST-NUMBER)

This function is the user interface to the database management system. The user calls this function and gets the test with the TESTID TEST-NUMBER. Actually the system will search for the test in the list of the loaded tests. If it will not be found a "test fault" will occur and the system will look for the file for this test and load the test from there. If by adding the test to the list of loaded tests an "overflow" occurs (the number of tests is more then the maximum allowed), the last test on the list (the least recently used will be deleted from the list.

(TEST.GET-FIELD-VALUE FIELD DATUM)

Gets the "Actual value" of a test field. If the field value is indirect reference to other test, the field value will be retrieved from there (using TEST.GET-FIELD-VALUE thus enable chaining), otherwise it will return the field value of DATUM.

(TEST.GET-DEFAULT-FIELD-VALUE FIELD-NAME)

FIELD-NAME is a TEST field name. The function will return the global default value for this field.

The following functions can be useful when building a test for use within the actual test fields:

(TEST.TEST-SINGLE-TIME TEST-NUMBER)

There are tests which have the TIMES field greater then one. Such tests will be executed more then one time. there are also tests which have as their INPUT field a list starts with the atom SYSTEMATIC. Such tests will be applied on all the combinations of the elements of sets in their INPUT field. Such tests can run for hours only to discover that something in the test itself was not correct, and the trace file is full of garbage. To avoid such a cases it is recommended to create for such tests a "Testing test" which will perform the tested test once and will only check that the outcome is meaningful (i.e. The result is either "success" or "failure" etc). Such a test can be built easily by having as its Expression field a call to this function with the tested test as TEST-NUMBER. It may be also useful to add a WEAK or STRONG link in the tested test PRETESTS field.

(TEST.ERRORP EXPR) [function]

If an error occured during the evaluation of the evaluated expression, the returned result will be a list of two elements, the first is the atom ERROR!, and the second is the error number. This is a simple predicate that checks if an expression is of the form described above. It is useful for building the success predicate. For example, a test may have as the EVALEXPR the function ADD1, as its INPUT field some non numeric atom, and as its success predicate, the expression (LAMBDA (RES ARGS) (IF (AND (TEST.ERRORP RES)(EQP (CADR RES) 10)) THEN 'SUCCESS ELSE 'FAILURE] . This test tests whether the function ADD1 breaks with non-numeric arg error.

(TEST.ALL-COMBINATIONS SET-OF-SETS) [function]

Produces the Cartesian product of the sets in the list SET-OF-SETS . (TEST.ALL-COMBINATIONS '((a b)(1 2 3))) will return the list ((a 1)(a 2)(a 3)(b 1)(b 2)(b 3)). This function is used by the tester when it encounter an INPUT field starts with the atom SYSTEMATIC. It is useful in any case that the user wants to build tests that tries all the combinations of possible values of a function arguments, or to have all the possible settings for flags and global variables for some subsystem or library package.

(TEST.LOCAL-EVAL-FORM FORM) [function]

Evaluates the form FORM using ERRORSET, thus the avaluation will not break even if error condition occurs. If error did not occur the function will return the result of the evaluation of form. If error condition was entered, the function will return a list with the first element ERROR!, and the second element the number of the error (as described in Interlisp-D manual.

(TEST.PERFORM-TIMED-EVALUATION FORM TIMEOUT.ms) [function]

Evaluates FORM using TEST.LOCAL-EVAL-FORM (thus it will not break). If the evaluation will take time which is longer then the value of TIMEOUT.ms, the function will return the error expression (ERROR! TIMEEXPIRED). This function is used by the test driver if the TIMEOUT field of a test is non NIL, thus an evaluation that will take more time that the designated time will be considerd as returning with error condition. The function create a seperate process to perform the evaluation, and set a timer for the designated time (plus some time for overhead). The user should note that since the Interlisp-D process schedualing algorithms are non-preemptive, the function is not guarantied to halt. An infinite loop may not release the CPU and then only keyboard interrupt will work.

(TEST.GET-NEXT-AVAILABLE-TESTID) [function]

The test ids should be unique. That's why the system maintains a file which holds the next available test id. This function access this file and returns the id.

(TEST.GET-AND-INCREASE-NEXT-TESTID) [function]

This functions returns the next available id as the one above, but also increase this number on the file. this function is called by the function TEST.ADD-TEST which adds a test "officially" to the world.

The Random Generator

The random generator resides on the file RANDOM-GENERATOR.DCOM. The main function is

(TEST.GENERATE-RANDOM OBJECT-SPECIFICATION) [function]

This function is planned to be constantly expanded by the tests builders according to their needs. The OBJECT-SPECIFICATION can be an atom, which should be one of the objects known by the random generator. It can also be a list where the first element is an atom which is one of the known objects, and the rest of the list are modifiers according to the object type. The current list of known objects is : (INTEGER, SPECIAL-INTEGER, BOUND-INTEGER, LARGE-INTEGER, SMALL-INTEGER, BIGNUM, POSITIVE-BIGNUM, SPECIAL-BIGNUM,POSITIVE-POWEROF10-BIGNUM, WINDOW, REGION, SHORT-SIMPLE-LIST, SHORT-SIMPLE-NON-NULL-LIST, SHORT-LIST, LIST-OF-CHARACTERS, CHARACTER, LIST-OF-ITEMS and some more.

Some of the objects have modifiers, like short list which can have a maximum depth as a modifier. A very important object is LIST-OF-ITEMS which can have as its modifier another object specification, thus enable recursive use of the function. Thus you can write (TEST.GENERATE-RANDOOM '(LIST-OF-ITEMS REGION 100 200)) which will produced betwwen 100 and 200 random regions.

Database access

(TEST.GET-LOCKING-USERS TEST-LIST) [function]

TEST-LIST is a list of test numbers or the atom DATABASE. Will return the list of all the users that kas locks to tests in TEST-LIST together with the tests in TEST-LIST that they are locking. If TEST-LIST is the atom DATABASE, the function will return the names of all the users that have locks to any test.

(TEST.OBTAIN-DATABASE-WRITE-LOCK TEST-LIST) [function]

TEST-LIST is as above. The function tries to obtain locks on the list of tests in TEST-LIST. If TEST-LIST is the atom DATABASE, it will try to obtain lock on the whole data base. A lock on a test can be obtained if there is no other user locking the test. A lock to the whole data-base can be obtained if there is no user that locks any test. The function returns the list of all tests that it was able to lock.

(TEST.RELEASE-DATABASE-WRITE-LOCK TEST-LIST) [function]

As above, but releases the locksto the tests in TEST-LIST that are locked by the user. Returns all the tests that it suceeded to release.

(TEST.SEND-RELEASE-REQUESTS TEST-LIST) [function]

TEST-LIST is as above. Sends automatic messages to all the users with locks to the tests in TEST-LIST to release their locks.

(TEST.MARK-AS-CHANGED TEST-NUMBER) [function]

The number of thetests that are being modified using the programs editor are added to the global list TEST.LIST-OF-MODIFIED-TESTS. This can be done by calling this function .

(TEST.UNMARK-AS-CHANGED TEST-NUMBER) [function]

As above, but remove the test from the list.

(TEST.STORE-CHANGED-TESTS) [function]

Stores all the tests in the list of modified tests.

(TEST.STORE-TEST TEST-NUMBER) [function]

Stores the test TEST-NUMBER in the file wit h the name returned by the function TEST.TEST-NUMBER-TO-FILE-NAME, and removes it from the list of changed tests.

(TEST.TEST-NUMBER-TO-FILE-NAME TEST-NUMBER) [function]

Returns a file name on which the test TEST-NUMBER is stored. The directory is the value of the global variable TEST.TEST-DATA-BASE-DIRECTORY. The the root name for test number 45 will be TEST00045.

The Concept Space Browser

To browse a concept space you can either select the submenu "Browse Concept Space", or by calling the function

(TEST.DISPLAY-CONCEPT-SPACE-BROWSER CONCEPT-SPACE REGION/POSITION DEPTH INCLUDE-TESTS) [function]

CONCEPT-SPACE must be of type CONCEPTSPACE. REGION/POSITION can be either a region or a position for the browser window. DEPTH is the depth of the lattice that will be displayed. If INCLUDE-TESTS is non NIL, the tests will be included as part of the displayed graph. Only the first argument is neccessary.

All the operations on concepts are done in PREFIX form - first you select the operation and then the argument (like all Lisp operations). Copy selection from a node will push the list of test numbers belong to this node into the current tty stream. The operations that are available using the concept space browser are:

Copy subtree : Allow you to copy a subtree from one displayed concept space to another one. Will prompt for selection of the new parent node and the root of the subtree.

Add Concept : Prompts for the parent of the new concept and for the name of the new concept.

Delete concept : Deletes the concept selected, and all its children which have the deleted concept as they only parent (and so on recursively).

Add Link : prompts for the superconcept and the subconcept and creates a link between them.

Delete Link : prompts for the superconcept and the subconcept and deletes the linkl between them.

Add test : Adds a test to a concept. Prompts for selection of the concept and for a test number. The test number can be a list of numbers, thus you can copy select tests from any node on any browser window.

Delete test : Will ask you to select a node and will add a menu with all the tests of the node so that you can select those you want to delete.

Edit test : Will ask you for selection of a node and will pop up a menu of all the tests in the selected concept. Will apply the test editor on the selected test.

Display - Display tests on/off : will switch the mode of display. You can either display the graph with the tests as part of it or only with the concepts.

Display - Browse subtree : Will ask you to select a concept and will aply the concept space browser on the subgraph for which the selected node is the root.

Display - Change depth : Will pop up a menu of integers. You can select the depth of the graph being displayed.

Display - Update : Recomputes the graph of the concept space and redisplays it.

Execute tests : Will execute all of the tests or part of the tests of the selected concept. Will pop up a menu to set the execution modes.

Hardcopy tests ; Sends a pretty printed hardcopy of all the tests (or the selecetd tests) of the selected concept.

data base - obtain lock : Tries to obtain lock on all the tests of the selected concept.

data-base - Release lock : Releases all the locks that the user has to tests belongs to the selected concept