

Lyric Manual Test procedures

Filed as: {Eris}<Test>Admin>ManualManual.tedit

This document is a part of the procedures describing how to run tests on the Xerox Lisp Environment. The following is a list of the tests that must be run by hand. These are of two types. 1. Those that are run via the do-test software and thus automatically log errors. These are denoted by a .u extention. 2. Those that must be manually logged.

Running interactive regression tests using do-test

1. Bring up the LISP.SYSOUT to be tested.
2. In an XCL executive window, load DO-TEST by typing (LOAD "{ERIS}<LISPCORE>INTERNAL>LIBRARY>DO-TEST.DFASL")
3. To run tests using DO-TEST, type (DO-ALL-TESTS :PATTERNS '("filenames") :RESULTS "{Eris}<Test>SubSystemName>SubsystemName.log")

For example, to run all the interactive tests for the debugger and put the results in a file named debugger.log, do the following:

```
(DO-ALL-TESTS
  :PATTERNS '("{Eris}<Test>Env>Debugger>Hand>*.u;")
  :RESULTS "{Eris}<Test>Env>Debugger>Logs>Debugger.log")
```

Note the importance of using the terminating semicolon on test file names. Not supplying the semicolon makes it run all versions of that file name!

See {eris}<lispcore>internal>library>do-test.tedit for the full list of features.

Helpful utilities:

(IL:FILESLOAD WHO-LINE) ;each field is active to help change them.
(IL:FILESLOAD FILEWATCH) ;use background menu to turn on a monitor of what files you have open

SUCCESS / FAILURE prompt windows

Some tests will pop up a prompt window requesting certain actions and to look for certain results. At the top of the window are the words SUCCESS and FAILURE. If the results of the test are as prompted, select SUCCESS otherwise select FAILURE. You may move and shrink this window, but do not close it.

Before this window is popped up, you will be asked if you want a SHORT, MEDIUM, or LONG test and you will be presented with a menu. A short test will only test high priority items. LONG will test all cases.

Reporting results

After running a test for a subsystem, shift-select the log file into a message and address it to: John Sybalsky, the appropriate developer, and the test writer.

Here is a lafite-form for this purpose:

Subject: Test results for >>subsystem name<<
To: >>developer<<, >>test writer<<, >>documenter<<
cc: Sybalsky.pa

>> test log <<

LIST OF MODULES TO BE TESTED:

EXEC - Tests written by John Park

Location of Old Detailed Test Procedures: {eris}<test>env>exec>hand>test.proc
 Approximate time to run test: < 1 hr.

Note:

You cannot use do-all-tests with the Exec test suite. You must instead:

(DO-TEST-FILE '{Eris}<Test>Env>exec>Hand>FOO.u)

where FOO is the name of each .u file in the directory.

These are automatic tests, which feed the input into exec via bksysbuf. Test results are automatically logged in {eris}<lispcore>test>exec>test.report, but you must make sure a new version of the file exists before you run the tests.

Be sure the exec is in the XCL-TEST package.

While running these tests you must not place the caret outside of the exec or do any other work while the tests are running.

1108 Regression tests assigned to Norm Schuster, 3/18/87.

DEBUGGER - Tests written by Kirk Kelley

Relevant developer: Andy Daniels

(DO-ALL-TESTS

:PATTERNS '("{Eris}<Test>Env>Debugger>Hand>*.u;")

:RESULTS "{Eris}<Test>Env>Debugger>Logs>Debugger.log")

Approximate time to run test: < 1 hr.

Note: These are a mixture of automatic tests and those that require interaction. Some of them intentionally pop up break windows since this tests break windows. If in the process of testing you get a break window that is not obviously part of the test, uparrow out of it (type an ↑ or use the one from the title menu) and the next test should appear.

These tests were generated (and run) on an 1108.

Regression tests assigned to Masa Tateno, 3/18/87.

DEDIT - Tests written by Henry Cate

Location of test file: {eris}<test>env>DEdit>hand>*.u

Location of log file: {eris}<test>env>DEdit>logs>DEdit.log

Approximate time to run test: < 1 hr.

1108 Regression tests assigned to Albert Sahim, 3/18/87.

DISPLAY - Tests written by Peter Reidy

Location of test procedure: {eris}<test>i/o>display>hand>cursor.proc

Location of test source code (used by both cursor.proc and cursor.test:

{eris}<test>i/o>display>hand>cursor.test

Location of log file: {eris}<test>i/o>display>logs>cursor.log

HARDCOPY - Tests written by Peter Reidy

Location of test plans: {erinyes}<test>lisp>lyric>plans>fx80driver.plan, 4045xlpstream.plan, press/interpress.plan

Location of test procedure files: {eris}<test>i/o>hardcopy>hand>fx80driver.proc, 4045xlpstream.proc, press/interpress.proc

Location of log files: {eris}<test>i/o>hardcopy>hand>fx80driver.log, 4045xlpstream.log, press/interpress.log

Location of test code: {eris}<test>i/o>hardcopy>hand>streamtests.u

Location of test cases: {eris}<lispcore>test>streams>

Regression tests assigned to >>NAME<<, >>DATE<<.

KEYBOARD - Tests written by Henry Cate

Location of test file: {eris}<test>i/o>Keyboard>hand>*.u
 (As of march 10th, there are 4 tests.)
 Location of log file: {eris}<test>i/o>Keyboard>logs>Keyboard.log
 Approximate time to run test: about 15 minutes.

Regression tests assigned to Albert Sahim, 3/26/87.

**PROGRAM ANALYSIS - Tests written by John Park
(Subsystems: Masterscope, Databasefns, Browser, and Inspector)****Masterscope**

Location of test files: {eris}<test>env>program-analysis>hand>masterscope.u
 Location of log file: {eris}<test>env>program-analysis>logs>masterscope.report
 Approximate time to run test: 7 minutes.

Databasefns

Location of test files: {eris}<test>env>program-analysis>hand>databasefns.u
 Location of log file: {eris}<test>env>program-analysis>logs>databasefns.report
 Approximate time to run test: 3 minutes.
 Note: The data file used by this test is in {eris}<test>env>program-analysis>hand>databasefns.data

Browser

Location of test files: {eris}<test>env>program-analysis>hand>browser-part1.u
 {eris}<test>env>program-analysis>hand>browser-part2.u
 Location of log file: {eris}<test>env>program-analysis>logs>browser.report
 Approximate time to run test: 8 minutes.
 Note: There are two test files for browser testing. Part 2 must be executed after Part 1.
 The data file used by this test is in {eris}<test>env>program-analysis>hand>browser. graph

Spy

Location of test files: {eris}<test>env>program-analysis>hand>spy.u
 Location of log file: {eris}<test>env>program-analysis>logs>spy.report
 Approximate time to run test: 5 minutes.

Inspector

Location of test files: {eris}<test>env>program-analysis>hand>inspect.u
 Location of log file: {eris}<test>env>program-analysis>logs>inspect.report
 Approximate time to run test: 8 minutes.
 Note: A fatal bug was discovered - Inspect never returns when *random-state* is inspected (AR # 8203).

Regression tests assigned to Norm Schuster, 3/26/87.

**PROGRAM SUPPORT - Tests written by John Park
(Subsystems: DWIM, and PRETTYPRINT)****DWIM**

Location of test files: {eris}<test>env>program-support>hand>dwim.u
 Location of log file: {eris}<test>env>program-support>logs>dwim.report
 Approximate time to run test: 2 minutes.
 Note: The DWIM test is executed by entering (DWIM-TEST). SEE the test file for more info.

Printing Out Function Definitions (PRETTYPRINT)

This test is covered by PP and other subsystems of Executative Test .
 (See {eris}<test>env>exec>hand>pp.u and also see.u, see-without-comment, ty.u and type.u in the same directory)

PROCESS CONTROLS (PSW) - Tests written by John Park

Location of test files: {eris}<test>env>process-controls>hand>psw.u
 Location of log file: {erinyes}<test>env>process-controls>logs>psw.report
 Approximate time to run test: 2 - 10 minutes.

SEdit - Tests written by Henry Cate

For SEdit test, load TEdit. Then change packages with: (cl:in-package 'xcl-test)
 For do-all-tests, it may work better if the semicolon is not used.

Location of test file: {eris}<test>env>code-editor>hand>*.u
 (As of march 2nd, there are 21 tests.)

Location of log file: {eris}<test>env>code-editor>logs>SEdit.log

Approximate time to run test: about a day.

1108 Regression tests assigned to Albert Sahim, 3/18/87.

INSPECTOR - Tests written by Lois Lew

Location of test files: {eris}<test>env>env>inspector>hand>inspect-allrec.tedit
 {eris}<test>env>inspector>hand>inspect-defstruct.tedit
 {eris}<test>env>inspector>hand>inspect-macro.tedit
 {eris}<test>env>inspector>hand>inspectw.tedit
 {eris}<test>env>inspector>hand>inspect-code.tedit
 {eris}<test>env>inspector>hand>inspectfieldflg.tedit

Location of log file: create one at:

{eris}<test>env>inspector>logs>inspect-allrec.log
 {eris}<test>env>inspector>logs>inspect-defstruct.log
 {eris}<test>env>inspector>logs>inspect-macro.log
 {eris}<test>env>inspector>logs>inspectw.log
 {eris}<test>env>inspector>logs>inspect-code.log
 {eris}<test>env>inspector>logs>inspectfieldflg.log

Approximate time to run test: ?

Note: use shift select to copy the material marked to be typed into the appropriate exec window.

Regression tests assigned to Masa Tateno, 3/26/87.

Running AR Test Cases & Recording the Results

The Pass and Fail commands

The "Pass" and "Fail" commands are defined on <Lispcore>Internal>Library>RELEASETOOLS. The form of use is

Pass *AR#1 AR#2 AR#3 . . .* ;Records the fact that AR Test Cases for the ARs named were run and succeeded.

Fail *AR#4 AR#5 AR#6 . . .* ; Records the fact that AR Test Cases for the ARs named were run and failed.

These two commands record each AR in a log file, along with your user name (so we know who ran it), and the current time (so we know when). These log entries will be used to

- 1) Track the current status of the AR test cases, and
- 2) Prepare the alpha-test log for Beta-test readiness review.

Running a test

When you run an AR test case, please use the Pass or Fail command to record the fact!

AR Test Case Status Summary

As of >>Date<<

>>AR Test &
Status<<

Special Files on <Test>ARs>

AR-TEST-CASE.Auto-log;1

This is the file where the "Pass" and "Fail" commands record the results of running an AR test case.
See {Eris}<Test>Admin>Running-AR-Test-Cases.TEdit for details.

>><<
>><<

>><<
>><<

>><<
>><<

>><<
>><<

SEdit test report

This report is for tests written and executed up to February 28, 1987 on the <Lyric>Basics>Full.Sysout generated 21-Jan-87.

The following tests are for the integration of the new error system into the Interlisp environment.

The test plan for this report is {Erinyes}<Test>Lisp>Lyric>Plans>SEdit.NoteFile

Groups of tests were written and executed on the 1186 for the following commands:

INTERRUPTS

ABORT
BASE
COMMENT
EVAL
EXPAND
EXTRACT
FIND
HELP
JOIN
MENU
EDIT DEFINITION
MUTATE
PACKAGE
PARENTHE SIZE
REDO
SKIP-NEXT
SUBSTITUE
UNDO

DELETE PREVIOUS WORD
DONE
REDISPLAY

These require user interaction. They are stored at {Eris}<Lispcore>Test>SEdit>*.u

Still to be formally tested

The mouse, (shift selection (copy, delete, move) to and from each source:

1. The same SEdit
2. A different SEdit
3. TEdit
4. Exec (each type/profile)

Window. Test each window operation with SEdit in each of stand selection/caret situations. SEdit should reappear in the same situation.

Completion. Make sure SEdit completes properly for each type of definition being edited.

1. def-types
2. fns
3. vars
4. property lists
5. records and datatypes.

Edit interface:

1. ed and editdef
2. df, dv, dc, dp
3. multiple edit drivers: editfromfile, editcallers, masterscope, etc...

New ARs generated

7553 Not able to parenthesize part of an litatom
 7604 SEdit, Meta-B does not work with decimals
 7607 SEdit, problem with Base, change base and try placing edit caret, break
 7624 SEdit, Attach Menu, come up with incorrect Print-Base

 7629 SEdit, meta-E has trouble evaluating a number, breaks
 7631 SEdit, meta-E, evaluates all, even when only part is selected
 7642 SEdit, meta-E, something ought to happen with extended selection
 7682 SEdit, control-W after single quote breaks
 7688 SEdit, CASE on a string or litatom doesn't change the case

 7699 SEdit, help, if nothing is selected, nothing happens, should give some feedback
 7703 SEdit, meta-H, no response against a string
 7705 SEdit, meta-H, break window when have an extended selection
 7717 SEdit, meta-J, new, nothing selected, type meta-J, break window
 7729 SEdit, documentation says prompt window, confusing

 7731 SEdit, try meta-J against an extended selection of numbers, breaks
 7733 SEdit, meta-J can join atoms and strings, different types
 7783 SEdit, meta-O, optimizers, cann't use cl:optimize
 7784 SEdit, meta-O, lets the user get started then says no way
 7792 SEdit, meta-O some times the message overflows

 7842 SEdit, meta-; documentation on old format, need il:
 7851 SEdit, combination of meta-0, & meta-/ builds break window
 7878 SEdit, meta-s, cann't substitue in extra items in a list
 7879 SEdit, meta-S cann't substitute in a comment
 7880 SEdit, meta-s, cann't substitute for an extended selection of a number

 7889 SEdit-, meta-n give some feedback when don't do anything
 7907 SEdit, meta-n, select vs extended selection give different results

Debugger, Error System, and Unwinder test report

This report is for tests written and executed up to March 24, 1987 on the <Lyric>Basics>Full.Sysout generated 11-Mar-87.

The following tests are for the integration of the new error system into the Interlisp environment.

The source for this report is {Erinyes}<Test>Lisp>Lyric>Plans>Debugger.NoteFile

The print version of this test report is filed at {Eris}<Lispcore>Test>Debugger>Report.IP

The test plan is filed at {Erinyes}<Test>Lisp>Lyric>Plans>Debugger.NoteFile and IP.

Groups of tests were written and executed on the 1109 for all the commands in the Debugger document in the Xerox Common Lisp implementation notes.

These are stored on {eris}<Lispcore>Test>Debugger>BreakWindow.u

Groups of tests were written and executed on the 1109 for most of the functions in IRM chapter 15: Breaking, Tracing, and Advising.

These are mostly automatic tests but some require user interaction. They are stored in {Eris}<Lispcore>Test>Debugger>Debugger.u

For tests of the Xerox extensions to the CML error system, see {Eris}<Lispcore>cml>test>24-*. {Eris}<Lispcore>cml>test>cl-error.x and {Eris}<Lispcore>cml>test>errorsystem.notefile which is the source for {Eris}<Lispcore>cml>test>24-errorsystem.x.

Regression Test for ARS

ARS tested 21-Jan-87 <Lyric>basics>full.sysout

7152 passed
7780 new
7797 new
6503 passed

New ARs generated

Several problems were discovered and reported as ARs. Each of these have tests. The following are only new ARs generated up to February 28.

DEBUGGER.NEWARS

7486 debugger "eval:" undocumented

7522 missing second param causes random error reporting

7601 unnamed proceed cases break compute-proceed-cases

7679 Version .01 Error System documentation edits

7780 argument names of broken fns unbound in debugger

7797 breaking/tracing advised fns does not update brokenfns

7845 exec il:settopval il:helpflag serious-condition attempt-to-change-constant

7848 (IL:NLSETQ (CL:SIGNAL 'ERROR)) breaks

7868 bad package fix readtable change screws up fix

7873: Common functions should be safe to break

AR 7923 il:brkinfolst no longer exists

AR 7908 untrace also unbreaks

AR 7919 TRACE no longer works for undefined subfunctions

AR 7932: (unbreak (sub-fn in super-fn)) has two problems

Original List of ARs

ERRORSYSTEM.ARSUMMARY

AR Summary generated on 17-Feb-87 10:33:56

Generated with Query Spec: (AND (Submitter: HAS Kelley))

Sorted with Sort Spec: (Status:)

| Numbe | Date: | System: | Subsystem: | Status: | Attn: | Subject: | Priority: | Difficulty | Impact: |
|-------|-----------|----------------|----------------|------------|---|--|-----------|------------|---------|
| 6787 | 4-Nov-86 | Common Lisp | File System In | New | Jellinek.pa | CL:OPEN says FILE NOT FOUND for BUSY FILE | | | |
| 6810 | 5-Nov-86 | Common Lisp | Streams and I/ | New | Jellinek | make-synonym-stream core file read fails | | Absolutely | |
| 6847 | 10-Nov-86 | Common Lisp | Other | New | vanMelle, W | Making straight common-lisp text files using SEdit | | Unlikely | |
| 6987 | 2-Dec-86 | Common Lisp | Streams and I/ | New | Jellinek | make-broadcast-stream should check for list arg | | | |
| 7033 | 9-Dec-86 | Programming En | Code Editor | New | Wozencraft | SEdit global replace breaks | | | |
| 7067 | 15-Dec-86 | Common Lisp | Streams and I/ | New | Jellinek | force-output should flush pages buffered in vmem | | | |
| 7118 | 29-Dec-86 | Programming En | Code Editor | New | woz | Sizing to fit SEdit window region length | | Perhaps | |
| 7168 | 8-Jan-87 | Common Lisp | Other | New | | Portable DO-TEST needs expect-errors | | | |
| 7268 | 19-Jan-87 | Language Suppo | Storage Format | New | vanmelle | SPELLFILE should use FILEDATES prop, but doesn't | | | |
| 7308 | 21-Jan-87 | Programming En | Break Package | New | Daniels | CL:READ should be in the list of break warning fns | | | |
| 7436 | 4-Feb-87 | Communications | Other | New | LispCore↑.p | Constant requests for passwords | | | |
| 7439 | 4-Feb-87 | Common Lisp | Error System | New | Biggs, Dani | Error system documentation needs re-writing | | | |
| 7448 | 4-Feb-87 | Programming En | Code Editor | New | Wozencraft, | SEdit loses edits of whole lists | | | |
| 7451 | 4-Feb-87 | | New | Fischer | Old-Interlisp-Exec comes up in current pkg&rdtbl | | | | |
| 7486 | 5-Feb-87 | Programming En | Break Package | New | Daniels | debugger "eval:" undocumented | | | |
| 7522 | 6-Feb-87 | | New | Daniels | CERROR missing second param causes random error reporting | | | | |
| 7601 | 10-Feb-87 | Common Lisp | Error System | New | Daniels | unnamed proceed cases break compute-proceed-cases | | | |
| 7679 | 12-Feb-87 | Common Lisp | Error System | New | Daniels,Big | Version .01 Error System documentation edits | | | |
| 7680 | 12-Feb-87 | Documentation | Product Descr/ | New | Biggs | WHO-LINE mention missing from overview | | | |
| 7681 | 12-Feb-87 | Text | TEDIT | New | Sybalsky, S | TEDIT not calling EDITBM ? | | | |
| 7686 | 12-Feb-87 | Documentation | Interlisp Refe | New | Biggs,Sybal | IL => CL function map needed | | | |
| 7687 | 12-Feb-87 | Programming En | Code Editor | New | Wozencraft | SEdit should reflect a change in readtable | | | |
| 7696 | 12-Feb-87 | Programming En | Code Editor | New | SeditSuppor | SEdit quits refreshing after soft stack overflow | | | |
| 5717 | 27-May-86 | Windows and Gr | Window System | Open | Wozencraft. | ATTACHWINDOW JUSTIFY should work for thin | | | |
| 6789 | 4-Nov-86 | Common Lisp | Streams and I/ | Open | Jellinek.pa | make-concatenated-stream core file problem | | | |
| 6919 | 18-Nov-86 | | Open | Jellinek | SETFILEINFO does not take FileName in all cases | | | Unlikely | |
| 4847 | 5-Dec-85 | Communications | Other | Fixed | | FileCache breaks when running init.firsttime | | Hopefully | |
| 7098 | 22-Dec-86 | Common Lisp | Streams and I/ | Fixed | | make-string-input-stream breaks on printed double | | | |
| 4890 | 10-Dec-85 | Operating Syst | Virtual Memory | Declined | | Changing CPE and memory boards gives MP9335 | | | |
| 6797 | 4-Nov-86 | | Declined | | | KEYACTION does not work for MOVE. | | | |
| 6799 | 5-Nov-86 | Common Lisp | Streams and I/ | Declined | Jellinek | make-string-input-stream calls OPENSTRINGSTREAM | | | |
| 7529 | 9-Feb-87 | Language Suppo | Stack and Inte | Incomplete | | il: fetch dwimification problems | | | |

DEBUGGER.ARSUMMARY

AR Summary generated on 16-Feb-87 16:23:08

Generated with Query Spec: (AND (Subsystem: IS Break Package))

Sorted with Sort Spec: (Status:)

| Numbe | Date: | System: | Subsystem: | Status: | Attn: | Subject: | Priority: | Difficulty | Impact: |
|-------|-------|---------|------------|---------|-------|----------|-----------|------------|---------|
|-------|-------|---------|------------|---------|-------|----------|-----------|------------|---------|

| | | | | | | | |
|------|-----------|----------------|--|-----|-------------|--|------------|
| 6851 | 10-Nov-86 | Programming En | Break Package | New | masinter | If a u.d.f. has a functions definition, want the d | |
| | 23:07:31 | vironment | Feature | | | efinition to be unsaved, ala Interlisp-D environme | |
| 6981 | 1-Dec-86 | Programming En | Break Package | New | Fischer.PA | Trace Window Overflow | Hopefully |
| | 08:56:50 | vironment | Annoying Bug | | | | |
| 6993 | 3-Dec-86 | Programming En | Break Package | New | daniels | Bug in debugger: Buttoning "display edit" from the | |
| | 15:57:33 | vironment | Hopefully Moderate Annoying Bug | | | frame window edits the function cell | |
| 7016 | 6-Dec-86 | Programming En | Break Package | New | jellinek, d | Want -> = commands for debugger | Unlikely |
| | 07:40:10 | vironment | Serious Design - Impl | | | | |
| 7079 | 17-Dec-86 | Programming En | Break Package | New | daniels | Debugger should skip SI::*UNWIND-PROTECT* frames | |
| | 13:26:10 | vironment | i Hopefully Easy Annoying Bug | | | n reporting errors | |
| 7084 | 17-Dec-86 | Programming En | Break Package | New | daniels, va | BREAK..OK doesn't work under Interlisp exec/profil | |
| | 18:07:17 | vironment | Absolutely Bug | | | e--uses wrong evaluator, gets uba BROKEN | |
| 7085 | 17-Dec-86 | Programming En | Break Package | New | daniels.pa | UNBREAK (foo :in bar) doesn't find uses in subfunc | |
| | 20:36:10 | vironment | Absolutely Moderate Serious Bug | | | tions | |
| 7089 | 19-Dec-86 | Programming En | Break Package | New | daniels.pa | OPENWP should be on list of unsafe functions to br | |
| | 10:23:56 | vironment | Absolutely Easy Fatal Bug | | | eak | |
| 7097 | 19-Dec-86 | Programming En | Break Package | New | Daniels | Wrong frame current in broken function in debugger | |
| | 22:36:05 | vironment | Hopefully Moderate Moderate Bug | | | | |
| 7122 | 29-Dec-86 | Programming En | Break Package | New | Daniels | Break window BT frame window doesn't show arg name | |
| | 13:47:11 | vironment | Absolutely Bug | | | s for broken fn. | |
| 7213 | 13-Jan-87 | Programming En | Break Package | New | daniels, pa | Trace replacing old fn defn. | Hopefully |
| | 02:10:08 | vironment | Serious Bug | | | vel | |
| 7215 | 13-Jan-87 | Programming En | Break Package | New | daniels | Arguments displayed during trace are random. | |
| | 02:16:43 | vironment | Perhaps Moderate Feature | | | | |
| 7236 | 14-Jan-87 | Programming En | Break Package | New | daniels | Information in breakpoint, backtrace, and frame wi | |
| | 16:07:14 | vironment | Absolutely Moderate Moderate Bug | | | ndow is printed with inconsistent package / readta | |
| 7267 | 16-Jan-87 | Programming En | Break Package | New | woz | Bad bahavior of backtrace window inspect menu | |
| | 16:06:54 | vironment | Unlikely Easy Annoying Design - UI | | | | |
| 7295 | 20-Jan-87 | Programming En | Break Package | New | Daniels, Bi | Need to finish documentation for the new DEBUGGER | |
| | 17:45:26 | vironment | Absolutely Hard Serious Documentation | | | | |
| 7296 | 20-Jan-87 | Programming En | Break Package | New | Fischer, Bi | New Step and trace need documentation | |
| | 17:48:23 | vironment | Absolutely Moderate Serious Documentation | | | | |
| 7308 | 21-Jan-87 | Programming En | Break Package | New | Daniels | CL:READ should be in the list of break warning fns | |
| | 16:14:01 | vironment | Absolutely Design - UI | | | | |
| 7355 | 26-Jan-87 | Programming En | Break Package | New | Pavel, Dani | Want better support for debugging interpreted code | |
| | 23:27:01 | vironment | Design - UI | | | | |
| 7371 | 28-Jan-87 | Programming En | Break Package | New | Daniels | Stack overflow condition should get normal debugge | |
| | 12:01:49 | vironment | Absolutely Moderate Design - UI | | | r window | |
| 7383 | 29-Jan-87 | Programming En | Break Package | New | Daniels | ENTER-DEBUGGER-P should say yes for STORAGE- | |
| | 11:41:06 | vironment | CONDIT Hopefully Easy Moderate Design - Impl | | | IONS | |
| 7384 | 29-Jan-87 | Programming En | Break Package | New | | STACK-OVERFLOW errors should get a new window | |
| | 11:43:08 | vironment | Hopefully Easy Annoying Design - Impl | | | | |
| 7402 | 30-Jan-87 | Programming En | Break Package | New | Daniels | Closing a debugger window does not always abort | |
| | 12:56:11 | vironment | Moderate Bug | | | | |
| 7441 | 4-Feb-87 | Programming En | Break Package | New | Daniels | Can't break Interlisp NLAMBDA's | Absolutely |
| | 14:57:56 | vironment | Serious Bug | | | | |
| 7445 | 4-Feb-87 | Programming En | Break Package | New | Daniels | INSPECTCODE from debugger should always inspect se | |
| | 16:51:54 | vironment | Hopefully Moderate Bug | | | lected frame | |
| 7474 | 4-Feb-87 | Programming En | Break Package | New | Daniels | The debugger should rebind *READ-SUPPRESS* to NIL | |
| | 21:28:01 | vironment | Absolutely Easy Serious Bug | | | | |
| 7486 | 5-Feb-87 | Programming En | Break Package | New | Daniels | debugger "eval:" undocumented | |

[illegible]

| | | | | | | | |
|----------|--------------------|----------------|-------------------------------|------------|---|--|------------|
| 20:39:12 | vironment | | | | acting like top-level typescript window | | |
| 5556 | 24-Apr-86 | Programming En | Break Package | Fixed | AUTOBACKTRACEFLG vs. TRACE | | |
| | Absolutely | | Bug | | | | |
| 11:48:47 | vironment | | | | | | |
| 5807 | 11-Jun-86 | Programming En | Break Package | Fixed | AUTOBACKTRACEFLG = ALWAYS breaks TRACE | | |
| | Absolutely | | Serious Bug | | | | |
| 09:38:52 | vironment | | | | | | |
| 5940 | 28-Jun-86 | Programming En | Break Package | Fixed | biggs BT in break windows isn't very useful. | | Absolutely |
| | Moderate | | Design - Impl | | | | |
| 02:06:18 | vironment | | | | | | |
| 6007 | 8-Jul-86 | Programming En | Break Package | Fixed | Break handling under CMLEXEC should be CMLreading | | |
| | Absolutely | | Annoying Design - Impl | | | | |
| 11:03:28 | vironment | | | | | | |
| 6128 | 19-Jul-86 | Programming En | Break Package | Fixed | Want to be able to include 1-cell arglist debuggin | | Absolutely |
| | Design - Impl | | | | | | |
| 04:17:22 | vironment | | | | g info in compiled code, have printcode etc know a | | |
| | | | bout them | | | | |
| 6445 | 8-Sep-86 | Programming En | Break Package | Fixed | Can't revert from the menu any more | | Absolutely |
| | Moderate | | Bug | | | | |
| 17:57:29 | vironment | | | | | | |
| 6712 | 27-Oct-86 | Programming En | Break Package | Fixed | Break package obscures system variables | | |
| | Absolutely | | Moderate Bug | | | | |
| 09:44:31 | vironment | | | | | | |
| 6796 | 4-Nov-86 | Programming En | Break Package | Fixed | EVAL in debugger doesn't print result | | Absolutely |
| | Moderate | | Design - Impl | | | | |
| 17:59:31 | vironment | | | | | | |
| 6817 | 6-Nov-86 | Programming En | Break Package | Fixed | Backtrace windows (and printed backtraces) show to | | |
| | Hopefully | | Annoying Design - UI | | | | |
| 12:33:52 | vironment | | | | o many calls | | |
| 6818 | 6-Nov-86 | Programming En | Break Package | Fixed | Stack frame display for CL EVAL frames should show | | |
| | Hopefully | | Moderate Annoying Feature | | | | |
| 12:40:05 | vironment | | | | more | | |
| 6927 | 19-Nov-86 | Programming En | Break Package | Fixed | debugger windows pop up on top of one another. | | |
| | Absolutely | | Annoying Design - UI | | | | |
| 11:54:28 | vironment | | | | | | |
| 6952 | 24-Nov-86 | Programming En | Break Package | Fixed | PAGEHEIGHT remains debugger window height after de | | |
| | Absolutely | | Annoying Bug | | | | |
| 12:01:36 | vironment | | | | bugger exit | | |
| 7121 | 29-Dec-86 | Programming En | Break Package | Fixed | ?= under breakpoint shows broken fn, rather than a | | |
| | Absolutely | | Bug | | | | |
| 13:44:59 | vironment | | | | rgs | | |
| 7155 | 7-Jan-87 | Programming En | Break Package | Fixed | Fischer Debugger BTV! command :print-junk option broken | | |
| | Absolutely | | Easy Annoying Bug | | | | |
| 11:23:37 | vironment | | | | | | |
| 7329 | 23-Jan-87 | Programming En | Break Package | Fixed | Help info on debugger menu is "NIL" | | Hopefully |
| | Annoying | | Bug | | | | |
| 18:09:49 | vironment | | | | | | |
| 7598 | 10-Feb-87 | Programming En | Break Package | Fixed | ↑ out of break under "EVAL" unwinds too far | | |
| | Absolutely | | Serious Design - Impl | | | | |
| 19:18:26 | vironment | | | | | | |
| 7599 | 10-Feb-87 | Programming En | Break Package | Fixed | "PROCEED" shows too much, starts at wrong frame | | |
| | Absolutely | | Moderate Design - Impl | | | | |
| 19:22:56 | vironment | | | | | | |
| 6237 | 28-Jul-86 | Programming En | Break Package | Closed | Change break package uses a special variable to * | | |
| | Easy | | Annoying Design - Impl | | | | |
| 14:01:45 | vironment | | | | WINDOW-BREAK* from WBREAK to decide whether to sta | | |
| | | | | | rt a new window or to use the same one instead of | | |
| | | | | | MOVDs when WBREAK(T) or WBREAK(NIL) | | |
| 2347 | 8-Oct-84 | Programming En | Break Package | Declined | Break of (REPLACE IN FOO) gives erroneous error me | | |
| | Perhaps | | Annoying Design - UI | | | | |
| 14:23:41 | vironment | | | | ssage (REPLACE was undefined) | | |
| 4102 | 17-Jul-85 | Programming En | Break Package | Declined | want ~ to be treated equivalent to ↑ in break wind | | Perhaps |
| | Annoying | | Design - UI | | | | |
| 19:13:39 | vironment | | | | ow | | |
| 744 | 17-Apr-84 | Programming En | Break Package | Superseded | (Superseded by AR 1035) ↑ out of break can switch | | |
| | Perhaps | | Moderate Annoying Design - UI | | | | |
| 16:25:26 | vironment | | | | TTY stream to TRACE window | | |
| 1740 | 2-Aug-84 | Programming En | Break Package | Superseded | editing under a break (superseded by AR 162) | | |
| | Moderate | | Bug | | | | |
| 14:00:53 | vironment | | | | | | |
| 2809 | 6-Dec-84 | Programming En | Break Package | Superseded | (Superseded by AR 5556) Setting | | |
| | AUTOBACKTRACEFLG = | | Perhaps | | Annoying Bug | | |
| 09:56:17 | vironment | | | | ALWAYS causes a break when a traced fn is called: | | |
| | | | | | "TRACE - UNBOUND ATOM" | | |
| 2863 | 12-Dec-84 | Programming En | Break Package | Superseded | BURTON.PA Harmony: tracing functions (superceded by AR | | |
| | 2863) | | | | | | |
| 16:10:44 | vironment | | | | | | |
| 2967 | 29-Dec-84 | Programming En | Break Package | Superseded | (Superseded by AR 1035) BREAK windows can be left | | |
| | Unlikely | | Minor Bug | | | | |
| 08:52:57 | vironment | | | | on screen after REVERT to broken fn. | | |

| | | | | | |
|------|-----------|------------------------|------------------------|--|--|
| 3487 | 14-Mar-85 | Programming En | Break Package | Superseded | (Superseded by AR 1035) Display stream switched to |
| | 09:13:59 | Perhaps | Annoying Design - Impl | | |
| | | viroment | | | break window when HELPFLAG set to break! |
| 3540 | 25-Mar-85 | Programming En | Break Package | Superseded | (Superseded by AR5556) When |
| | 08:39:38 | AUTOBACKTRACEFLG=ALWAY | Hopefully | Moderate Bug | |
| | | viroment | | SI, any TRACEd function stops with UNBOUND ATOM TR | |
| | | | ACE | | |
| 5219 | 14-Feb-86 | Programming En | Break Package | Superseded | superseded by AR494: HELPFLAG = BREAK! doesn't |
| | 11:41:33 | alw Absolutely | Moderate Bug | | |
| | | viroment | | ays work | |
| 6099 | 16-Jul-86 | Programming En | Break Package | Superseded | (Superseded by AR 5556) AUTOBACKTRACEFLG of |
| | | ALWAYS Perhaps | Annoying Bug | | |
| | 17:46:31 | viroment | | or ALWAYS! breaks TRACE | |
| 7011 | 6-Dec-86 | Programming En | Break Package | Superseded | debugger entry frame odd Absolutely |
| | 06:53:47 | Serious Design - UI | | | |
| | | viroment | | | |
| 7083 | 17-Dec-86 | Programming En | Break Package | Superseded | [SUPERSEDED BY 7121] ?= doesn't work for il:broken |
| | | Absolutely | Bug | | |
| | 18:01:11 | viroment | | fns | |
| 7247 | 15-Jan-87 | Programming En | Break Package | Superseded | Superseded by AR7236: Frame inspector window in De |
| | | Absolutely Moderate | Serious Bug | | |
| | 19:16:18 | viroment | | bugger always uses IL package, readtable | |
| 311 | 27-Mar-84 | Programming En | Break Package | Obsolete | HELPPFIX is calling EDITE with Type=FNS when there |
| | | Absolutely Easy | Annoying Design - UI | | |
| | 9:29:34 | viroment | | is no fn | |
| 2223 | 21-Sep-84 | Programming En | Break Package | Obsolete | BREAK frame window opens twice Perhaps |
| | | Annoying Design - Impl | | | |
| | 15:09:15 | viroment | | | |
| 2533 | 30-Oct-84 | Programming En | Break Package | Obsolete | Break window becomes Toplevel |
| | | Moderate Bug | | | |
| | 09:41:33 | viroment | | | |
| 2598 | 7-Nov-84 | Programming En | Break Package | Obsolete | Got UNBOUND ATOM "↑" and "?=" in break window |
| | | Perhaps | Annoying Bug | | |
| | 17:07:01 | viroment | | | |
| 2614 | 8-Nov-84 | Programming En | Break Package | Obsolete | MaxBkMenuHeight no longer effective Perhaps |
| | | Minor Bug | | | |
| | 14:55:36 | viroment | | | |
| 2928 | 19-Dec-84 | Programming En | Break Package | Obsolete | Break window frame inspect window opened twice whe |
| | | Perhaps | Minor Performance | | |
| | 08:09:37 | viroment | | n first created | |
| 2968 | 29-Dec-84 | Programming En | Break Package | Obsolete | REVERT to Nlambda-nospread with LOCALVAR arg |
| | | trans Absolutely | Moderate Bug | | |
| | 10:47:57 | viroment | | forms arg to (LIST arg) | |
| 6538 | 26-Sep-86 | Programming En | Break Package | Obsolete | TRACE keeps opening new windows, rather than reusi |
| | | Absolutely | Serious Bug | | |
| | 18:23:55 | viroment | | ng one trace window. | |
| 6829 | 6-Nov-86 | Programming En | Break Package | Obsolete | CLOSEW of a breakwindow sends ↑ instead of il:↑ |
| | | Minor Bug | | | |
| | 21:34:57 | viroment | | | |
| 539 | 6-Apr-84 | Programming En | Break Package | Incomplete | Break middle-button pop-up menu should switch the |
| | | Perhaps | Annoying Design - UI | | |
| | 10:20:11 | viroment | | TTY to the break window | |

DEdit test report

This report is for tests written and executed up to February 28, 1987 on the <Lyric>Basics>Full.Sysout generated 21-Jan-87.

The test plan for this report is {Erinyes}<Test>Lisp>Lyric>Plans>DEdit.tedit.

Groups of tests were written and executed on the 1186 for the following commands:

After, Before, Delete, Replace, SWitch, (), () out, Undo, Find, Swap, Reprint, Edit, EditCom, Break, Eval, Exit.

These require user interaction. They are stored at {Eris}<Lispcore>Test>DEDIT>high-level.u

New ARs generated

None

Scripts for testing the record package (assuming currently in **Common Lisp User** package)

Record type - Record

type:

```
(il:record record-test-name (alpha bravo gamma) (il:synonym alpha a) (il:type? (oddp (length il:datum))))  
(setq record-test-record (il:create record-test-name alpha il:_ '(a b c) bravo il:_ "some string"))
```

type:

```
(inspect record-test-record)
```

choose "inspect"

--should produce a window with three numbered elements 1 (a b c), 2 "some string", 3 nil

```
((a b c) some string
```

```
1 (a b c)  
2 "some string"  
3 nil
```

type:

```
(inspect record-test-record)
```

choose "as a record"

choose "record-test-name"

--should produce a window with four elements where a and alpha have the same value.

```
((a b c) some string nil)  
a (a b c)  
gamma nil  
bravo "some string"  
alpha (a b c)
```

type:

```
(inspect record-test-record 'record-test-name)
```

--should produce a window identical to the one in the previous step.

type:

```
(ed 'record-test-name 'records)
```

delete gamma from the element list.

type cntl-x to save edit.

type:

```
(inspect record-test-record 'record-test-name)
```

--should produce a window with three elements where a and alpha have the same value and there is no gamma.

return to the edit window and type gamma back in where it belongs.

type cntl-x to save again.

type:

```
(inspect record-test-record 'record-test-name)
```

--should produce a window with four elements where a and alpha have the same value and gamma is returned to the list

close the edit window

type:

```
(in-package 'interlisp)
```

type:

```
(inspect user::record-test-record 'user::record-test-name)
```

--should produce a window with four elements where user::a and user::alpha have the same value. All the values in the list (for a and alpha) are qualified as user.

```
((a b c) some string nil) Inspector
user::a      (user::a user::b user::c)
user::gamma nil
user::bravo  "some string"
user::alpha  (user::a user::b user::c)
```

type:

```
(cl:in-package 'user)
```

type:

```
(il:replace (record-test-name gamma) il:of record-test-record il:with '(a (b (c (d (e (f
(g)))))))
(inspect record-test-record)
```

select inspect

-should get:

```
((a b c) some string (
1  (a b c)
2  "some string"
3  (a (b #))
```

type:

```
(setq il:inspectprintlevel '(5 . 5))
(inspect record-test-record)
```

select inspect

-should get:

```
((a b c) some string (
1  (a b c)
2  "some string"
3  (a (b (c (d (
```

type:

```
(setq il:inspectprintlevel '(5 . 1))
(inspect record-test-record)
```

select inspect

-should get:

```
((a b c) some string (
1  (a ...)
2  "some string"
3  (a ...)
```

type:

```
(setq il:inspectprintlevel '(2 . 5))
```

Record type -Typerecord

type:

```
(il:typerecord record-test-name (alpha bravo gamma) (il:synonym alpha a) )
(setq record-test-record (il:create record-test-name alpha il:_ '(a b c) bravo il:_ "some
```

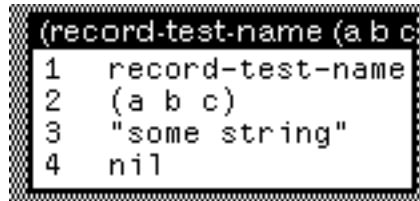
```
string"))
```

type:

```
(inspect record-test-record)
```

choose "inspect"

--should produce a window with four numbered elements 1 record-test-name 2 (a b c), 3 "some string", 4 nil



```
(record-test-name (a b c)
  1 record-test-name
  2 (a b c)
  3 "some string"
  4 nil)
```

type:

```
(setq il:maxinspectcdrlevel 2)
(inspect record-test-record)
```

choose "inspect"

--should produce a window with two numbered elements similar to the previous step and the elements number 3 and 4 in a list labeled il:&&

type:

```
(setq il:maxinspectcdrlevel 50)
```

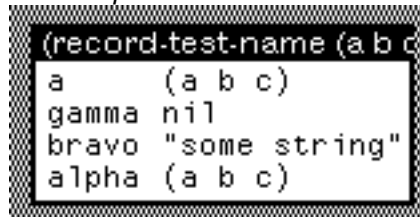
type:

```
(inspect record-test-record)
```

choose "as a record"

choose "record-test-name"

--should produce a window with four elements where a and alpha have the same value.



```
(record-test-name (a b c)
  a (a b c)
  gamma nil
  bravo "some string"
  alpha (a b c))
```

type:

```
(inspect record-test-record)
```

choose "as record-test-name"

--should produce a window identical to the one in the previous step.

type:

```
(inspect record-test-record 'record-test-name)
```

--should produce a window identical to the one in the previous step.

type:

```
(ed 'record-test-name 'records)
```

delete gamma from the element list.

type cntl-x to save edit.

type:

```
(inspect record-test-record 'record-test-name)
```

--should produce a window with three elements where a and alpha have the same value and there is no gamma.

return to the edit window and type gamma back in where it belongs.

type cntl-x to save again.

type:

```
(inspect record-test-record 'record-test-name)
```

--should produce a window with four elements where a and alpha have the same value and gamma is returned to the list

close the edit window

type:

```
(in-package 'interlisp)
```

type:

```
(inspect user::record-test-record 'user::record-test-name)
```

--should produce a window with four elements where user::a and user::alpha have the same value. All the values in the list (for a and alpha) are qualified as user.

```
(record-test-name (a b c) some string nil) Ins
user::a      (user::a user::b user::c)
user::gamma nil
user::bravo  "some string"
user::alpha  (user::a user::b user::c)
```

type:

```
(cl:in-package 'user)
```

Record type - Proprecord

type:

```
(il:proprecord record-test-name (alpha bravo gamma) (il:synonym alpha a) (il:type? (evenp (length
il:datum))))
(setq record-test-record (il:create record-test-name alpha il:_ '(a b c) bravo il:_ "some
string"))
```

type:

```
(inspect record-test-record)
```

choose "inspect"

--should produce a window with four numbered elements 1 alpha, 2 (a b c), 3 bravo, 4 "some string".

```
(alpha (a b c) bravo some string)
1  alpha
2  (a b c)
3  bravo
4  "some string"
```

type:

```
(inspect record-test-record)
```

choose "as a PLIST"

--should produce a window with two elements, alpha and bravo with their associated values.

```
(alpha (a b c) bravo some string)
alpha (a b c)
bravo "some string"
```

type:

```
(inspect record-test-record 'record-test-name)
```

--should produce a window with four elements where a and alpha have the same value.

```
(alpha (a b c) bravo some string)
a      (a b c)
alpha  (a b c)
bravo  "some string"
gamma  nil
```

type:

```
(ed 'record-test-name 'records)
```

delete gamma from the element list.

type cntl-x to save edit.

type:

```
(inspect record-test-record 'record-test-name)
```

--should produce a window with three elements where a and alpha have the same value and there is no gamma.

return to the edit window and type gamma back in where it belongs.

type cntl-x to save again.

type:

```
(inspect record-test-record 'record-test-name)
```

--should produce a window with four elements where a and alpha have the same value and gamma is returned to the list

close the edit window

type:

```
(in-package 'interlisp)
```

type:

```
(inspect user::record-test-record 'user::record-test-name)
```

--should produce a window with four elements where user::a and user::alpha have the same value. All the values in the list (for a and alpha) are qualified as user.

```
(alpha (a b c) bravo some string) Inspector
user::a      (user::a user::b user::c)
user::alpha  (user::a user::b user::c)
user::bravo  "some string"
user::gamma  nil
```

type:

```
(cl:in-package 'user)
```

Record type - Datatype

type:

```
(il:datatype record-test-name (alpha bravo gamma) (il:synonym alpha a))
(setq record-test-record (il:create record-test-name alpha il:_ '(a b c) bravo il:_ "some string"))
```

type:

```
(inspect record-test-record)
```

choose "inspect"

--should produce a window with four elements where a and alpha have the same value.

```
#<record-test-name @
a      (a b c)
alpha  (a b c)
bravo  "some string"
gamma  nil
```

type:

```
(inspect record-test-record 'record-test-name)
```

--should produce a window identical to the one in the previous step.

type:

```
(in-package 'interlisp)
```

type:

```
(inspect user::record-test-record 'user::record-test-name)
```

--should produce a window with four elements where user::a and user::alpha have the same value. All the values in the list (for a and alpha) are qualified as user.

```
#<record-test-name @ 375,74770> Inspect
user::a      (user::a user::b user::c)
user::alpha  (user::a user::b user::c)
user::bravo  "some string"
user::gamma  nil
```

type:

```
(cl:in-package 'user)
```

Record type - Arrayrecord

type:

```
(il:arrayrecord record-test-name (alpha bravo gamma) (il:synonym alpha a) (il:type? (cond
(il:datum t))))
(setq record-test-record (il:create record-test-name alpha il:_ '(a b c) bravo il:_ "some
string"))
```

type:

```
(inspect record-test-record)
```

--should produce a window with three numbered elements 1 (a b c), 2 "some string", 3 nil

```
#<arrayp @ 377,11
1  (a b c)
2  "some string"
3  nil
```

type:

```
(setq il:maxinspectarraylevel 2)
(inspect record-test-record)
```

--should produce a window similar to the previous one but with only two elements showing.

type:

```
(setq il:maxinspectarraylevel 300)
```

type:

```
(inspect record-test-record 'record-test-name)
```

--should produce a window with four elements where a and alpha have the same value.

```
#<arrayp @ 377,11514>
a      (a b c)
gamma nil
bravo "some string"
alpha (a b c)
```

type:

```
(ed 'record-test-name 'records)
```

delete gamma from the element list.

type cntl-x to save edit.

type:

```
(inspect record-test-record 'record-test-name)
```

--should produce a window with three elements where a and alpha have the same value and there is no gamma.

return to the edit window and type gamma back in where it belongs.

type cntl-x to save again.

type:

```
(inspect record-test-record 'record-test-name)
```

--should produce a window with four elements where a and alpha have the same value and gamma is returned to the list

close the edit window

type:

```
(in-package 'interlisp)
```

type:

```
(inspect user::record-test-record 'user::record-test-name)
```

--should produce a window with four elements where user::a and user::alpha have the same value. All the values in the list (for a and alpha) are qualified as user.

```
#<arrayp @ 377,11514> Inspector
user::a      (user::a user::b user::c)
user::gamma nil
user::bravo "some string"
user::alpha (user::a user::b user::c)
```

type:

```
(cl:in-package 'user)
```

Record type - Assocrecord

type:

```
(il:assocrecord record-test-name (alpha bravo gamma) (il:synonym alpha a) (il:type? (not (atom
(car il:datum))))))
(setq record-test-record (il:create record-test-name alpha il:_ '(a b c) bravo il:_ "some
string"))
```

type:

```
(inspect record-test-record)
```

choose "inspect"

--should produce a window with two numbered elements 1 (alpha a b c), 2 (bravo . "some string")

```
((alpha a b c) (bravo . some string))
1  (alpha a b c)
2  (bravo . "some string")
```

type:

```
(inspect record-test-record)
```

choose "as an ALIST"

--should produce a window with two elements, alpha and bravo and their associated values.

```
((alpha a b c) (bravo . some string))
alpha (a b c)
bravo "some string"
```

type:

```
(inspect record-test-record 'record-test-name)
```

--should produce a window with four elements and a has the same value as alpha.

```
((alpha a b c) (bravo . some string))
a      (a b c)
alpha  (a b c)
bravo  "some string"
gamma  nil
```

type:

```
(ed 'record-test-name 'records)
```

delete gamma from the element list.

type cntl-x to save edit.

type:

```
(inspect record-test-record 'record-test-name)
```

--should produce a window with three elements where a and alpha have the same value and there is no gamma.

return to the edit window and type gamma back in where it belongs.

type cntl-x to save again.

type:

```
(inspect record-test-record 'record-test-name)
```

--should produce a window with four elements where a and alpha have the same value and gamma is returned to the list

close the edit window

type:

```
(in-package 'interlisp)
```

type:

```
(inspect user::record-test-record 'user::record-test-name)
```

--should produce a window with four elements where user::a and user::alpha have the same value. All the values in the list (for a and alpha) are qualified as user.

```
((alpha a b c) (bravo . some string)) Inspecto
user::a      (user::a user::b user::c)
user::alpha  (user::a user::b user::c)
user::bravo  "some string"
user::gamma  nil
```


type:
 (cl:in-package 'user)

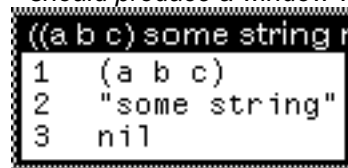
Record type - Accessfns

type:
 (IL:ACCESSFNS RECORD-TEST-NAME
 ((ALPHA (CAR IL:DATUM)
 (SETQ IL:DATUM (CONS IL:NEWVALUE
 (CDR IL:DATUM)))))
 (BRAVO (CADR IL:DATUM)
 (SETQ IL:DATUM (CONS (CAR IL:DATUM)
 (CONS IL:NEWVALUE
 (CDDR IL:DATUM)))))
 (GAMMA (CADDR IL:DATUM)
 (SETQ IL:DATUM (LIST (CAR IL:DATUM)
 (CADR IL:DATUM)
 IL:NEWVALUE))))
 (IL:CREATE (LIST '(a b c) "some string" NIL))
 (IL:TYPE? (ODDP (LENGTH IL:DATUM))))
 (setq record-test-record (il:create record-test-name)))

type:
 (inspect record-test-record)

choose "inspect"

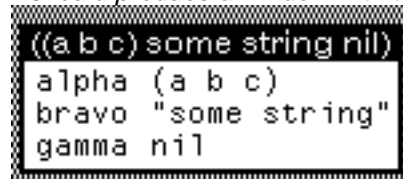
--should produce a window with three numbered elements 1 (a b c), 2 "some string". 3 ,nil



```
((a b c) some string nil)
1 (a b c)
2 "some string"
3 nil
```

type:
 (inspect record-test-record 'record-test-name)

--should produce a window with three elements: alpha, bravo and gamma with their associated values.



```
((a b c) some string nil)
alpha (a b c)
bravo "some string"
gamma nil
```

type:
 (ed 'record-test-name 'records)

delete the gamma entry from the element list.

type cntl-x to save edit.

type:
 (inspect record-test-record 'record-test-name)
 --should produce a window with two elements where there is no gamma.

return to the edit window and type the gamma entry back in where it belongs.

type cntl-x to save again.

type:
 (inspect record-test-record 'record-test-name)
 --should produce a window with three elements where gamma is returned to the list

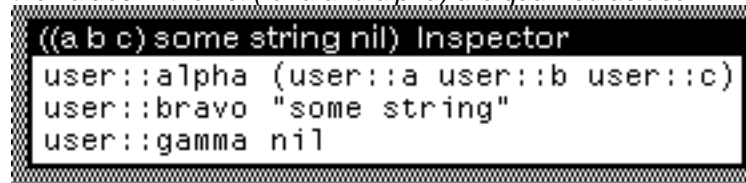
close the edit window

type:
 (in-package 'interlisp)

type:

```
(inspect user::record-test-record 'user::record-test-name)
```

--should produce a window with three elements where user::a and user::alpha have the same value. All the values in the list (for a and alpha) are qualified as user.



```
((a b c) some string nil) Inspector
user::alpha (user::a user::b user::c)
user::bravo "some string"
user::gamma nil
```

type:

```
(cl:in-package 'user)
```

Record type - Blockrecord


type:

```
(IL:DATATYPE RECORD-TEST-NAME
  ((ALPHA IL:POINTER)) alpha il:_ '(a b c))
(IL:BLOCKRECORD RECORD-TEST-NAME1
  ((BRAVO IL:WORD) (GAMMA IL:WORD)))
(SETQ RECORD-TEST-RECORD (IL:CREATE RECORD-TEST-NAME))
```

type:

```
(inspect record-test-record)
```

--should produce a window with alpha and the list (a b c)



```
#<record-test-r
alpha (a b c)
```

type:

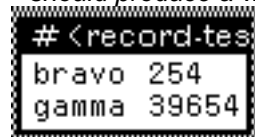
```
(inspect record-test-record 'record-test-name)
```

--should produce a window identical to the one in the previous step.

type:

```
(inspect record-test-record 'record-test-name1)
```

--should produce a window with bravo and gamma, each with two numbers



```
#<record-tes
bravo 254
gamma 39654
```

type:

```
(ed 'record-test-name1 'records)
```

delete the gamma entry from the element list.

type cntl-x to save edit.

type:

```
(inspect record-test-record 'record-test-name1)
```

--should produce a window with one element where there is no gamma.

return to the edit window and type the gamma entry back in where it belongs.

type cntl-x to save again.

type:

```
(inspect record-test-record 'record-test-name1)
```

--should produce a window with two elements where gamma is returned to the list

close the edit window

type:

```
(in-package 'interlisp)
```

type:

```
(inspect user::record-test-record 'user::record-test-name1)
```

--should produce a window with two elements where .All the values in the list are qualified as user.

```
#<record-test-name
user::bravo 254
user::gamma 39654
```

type:

```
(cl:in-package 'user)
```

Record type - Subrecords

type:

```
(il:record record-test-name (alpha bravo gamma) (il:synonym alpha a) (il:type? (oddp (length
il:datum))))
(il:record record-test-name1 (xray zebra record-test-name)
(il:subrecord record-test-name))
(setq record-test-record (il:create record-test-name1 alpha il:_ '(a b c) bravo il:_ "some
string" zebra il:_ "hi"))
```

type:

```
(inspect record-test-record)
```

choose "inspect"

--should produce a window with three numbered elements 1 nil 2 "hi" and 3 the list ((a b c) "some string" nil)

```
(nil hi ((a b c) some string nil)) Inspector
1 nil
2 "hi"
3 ((a b c) "some string" nil)
```

type:

```
(inspect record-test-record)
```

choose "as a record"

choose "record-test-name1"

--should produce a window with three elements the first one of which is a list of a list, a string and nil .

```
(nil hi ((a b c) some string nil)) Inspector
record-test-name ((a b c) "some string" nil)
zebra "hi"
xray nil
```

type:

```
(inspect record-test-record 'record-test-name1)
```

--should produce a window identical to the one in the previous step.

type:

```
(ed 'record-test-name1 'records)
```

delete zebra from the element list.

type cntl-x to save edit.

type:

```
(inspect record-test-record 'record-test-name1)
```

--should produce a window with no zebra and the value that used to be zebra is now record-test-name.

return to the edit window and type zebra back in where it belongs.

type cntl-x to save again.

type:

```
(inspect record-test-record 'record-test-name1)
```

--should produce a window with three elements the first one of which is a list of a list, a string and nil and zebra is returned to the list

close the edit window

type:

```
(in-package 'interlisp)
```

type:

```
(inspect user::record-test-record 'user::record-test-name1)
```

--should produce a window with three elements where all the values not in quotes are qualified as user.

```
(nil hi ((a b c) some string nil)) Inspector
user::record-test-name ((user::a user::b user::c)
user::zebra            "hi"
user::xray             nil)
```

type:

```
(cl:in-package 'user)
```

Script to check code inspectors on frames
(can be in any package)

type `cntl-b` to cause a break.

Place the mouse in the break window and press the middle button

select `BT!`

choose a frame that does not have a `\` and select it with the middle button.

When a menu appears choose `InspectCode`

Observe the code window -- it should scroll and allow selections but not allow any changing.

return to the window caused by the `BT!`

choose a frame that does have a `\` and select it with the middle button.

When a menu appears choose `InspectCode`

Observe the code window -- it should scroll and allow selections but not allow any changing.

return to the window caused by the `BT!`

select an entry with the left button.

a window will appear that has a title (whatever entry you chose) `Frame`.

In that window select a element in the left hand column with the left button.

Press the middle button over the same element.

select `"set"`

type: `5`

-- the value of the chosen element should become 5

Find a list in the right hand column of the frame window and select it with the left button.

Press the middle button and choose `"inspect"`

-- an inspect window with the elements of the list will appear

Script for testing defstruct and the inspector with defstruct
(assuming currently in **Common Lisp User** package)

type:

```
(cl:in-package "USER")
```

type:

```
(defstruct-rec (a "hello there" :type IL:pointer) (b 5 :type il:integer) (c 4 :type il:fixp) (d
2.3 :type il:floating) (e 4.5 :type il:floatp) (f -5 :type il:signedword :read-only t) (g T :type
il:flag) (h 7 :type il:bits3) (i 4 :type il:byte) (j 7 :type il:word) (k '(a b c d) :type
il:xpointer))
```

-- it should reply rec

type:

```
(setf rec1 (make-rec :e 2.3))
```

type:

```
(inspect rec1)
```

-- a window that looks like the following appears



```
IL:WITH S) S)
(EQ (IL:REPLACE BRAVO IL:OF RECORD-TEST-RECORD
IL:WITH ALFA) ALFA)))
```

(DO-TEST |refetch-arrayrecord|

```
(AND (EQ (IL:FETCH ALPHA IL:OF RECORD-TEST-RECORD) S)
(EQ (IL:FETCH BRAVO IL:OF RECORD-TEST-RECORD) ALFA)
(EQ (IL:FETCH ALPHA IL:OF RECORD-TEST-RECORD) S)))
```

(DO-TEST rereplace-arrayrecord

```
(AND (EQ (IL:REPLACE ALPHA IL:OF RECORD-TEST-RECORD
IL:WITH ALFA) ALFA)
(EQ (IL:REPLACE BRAVO IL:OF RECORD-TEST-RECORD
IL:WITH S) S)))
```

(DO-TEST |typeglobalvariable-GETFN: BMOBJ.GETFN3|

```
(EQ (IL:FETCH BRAVO IL:OF RECORD-TEST-RECORD) ALFA)
(EQ (IL:FETCH ALPHA IL:OF RECORD-TEST-RECORD) S)))
```

(DO-TEST rereplace-assocrecord

```
(AND (EQ (IL:REPLACE ALPHA IL:OF RECORD-TEST-RECORD
IL:WITH ALFA) ALFA)
(EQ (IL:REPLACE BRAVO IL:OF RECORD-TEST-RECORD
IL:WITH S) S)))
```

(DO-TEST |typeglobalvariable-assocrecord|

```
(EQ (SYMBOL-PACKAGE (IL:\TYPEGLOBALVARIABLE
(QUOTE RECORD-TEST-NAME)))
```

```
(FIND-PACKAGE "XCL-TEST"))))
```

```
(DO-TEST |using-assocrecord|
```

```
  (SETQ RECORD-TEST-RECORD3
```

```
    (IL:CREATE RECORD-TEST-NAME
```

```
      IL:USING RECORD-TEST-RECORD GAMMA IL:_ S))
```

```
  (AND (EQ (IL:FETCH ALPHA IL:OF RECORD-TEST-RECORD)
```

```
    (IL:FETCH ALPHA IL:OF RECORD-TEST-RECORD3))
```

```
    (EQ (IL:FETCH BRAVO IL:OF RECORD-TEST-RECORD)
```

```
      (IL:FETCH BRAVO IL:OF RECORD-TEST-RECORD3))
```

```
    (EQ (IL:FETCH GAMMA IL:OF RECORD-TEST-RECORD3) S)))
```

```
(DO-TEST |reusing-assocrecord|
```

```
  (SETQ RECORD-TEST-RECORD3
```

```
    (IL:CREATE RECORD-TEST-NAME
```

```
      ILuote (a b c d)) il:xpointer nil (rec 20 il:xpointer) rec-k)
```

Script for INSPECTALLFIELDSFLG
(assuming in the **Interlisp** package)

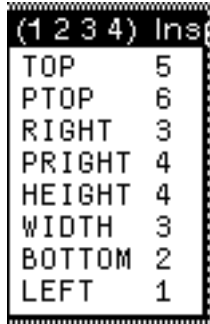
Type:

```
(CL:IN-PACKAGE "INTERLISP")
```

Type:

```
(SETQ INSPECTALLFIELDSFLG T)  
(INSPECT (CREATEREGION 1 2 3 4) 'REGION)
```

-The window should have the fields TOP, PTOP, RIGHT, PRIGHT, HEIGHT, WIDTH, BOTTOM, and LEFT. It should look like the following:

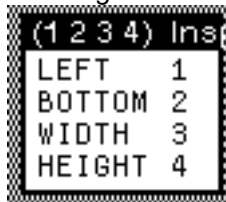


| | |
|--------|---|
| TOP | 5 |
| PTOP | 6 |
| RIGHT | 3 |
| PRIGHT | 4 |
| HEIGHT | 4 |
| WIDTH | 3 |
| BOTTOM | 2 |
| LEFT | 1 |

Type:

```
(SETQ INSPECTALLFIELDSFLG NIL)  
(INSPECT (CREATEREGION 1 2 3 4) 'REGION)
```

-The window should have the fields HEIGHT, WIDTH, BOTTOM, and LEFT. It should look like the following:



| | |
|--------|---|
| LEFT | 1 |
| BOTTOM | 2 |
| WIDTH | 3 |
| HEIGHT | 4 |

Type:

```
(SETQ INSPECTALLFIELDSFLG T)
```


Script for testing Inspect macro interface
Start in **interlisp** package

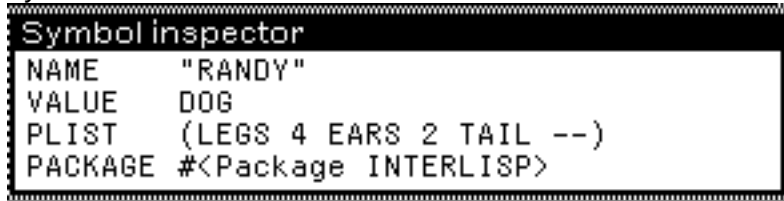
type:

```
(CL:IN-PACKAGE "INTERLISP")
```

type:

```
(DEFINEQ (DOGSP (ANIMAL) (OR (EQUAL (EVAL ANIMAL) 'DOG) (EQUAL (EVAL ANIMAL) 'dog))))  
(PUTPROPS RANDY LEGS 4 EARS 2 TAIL 1 HAIR LONG)  
(SETQ RANDY 'DOG)  
(INSPECT 'RANDY)
```

--you will see:



Symbol inspector

| | |
|---------|-------------------------|
| NAME | "RANDY" |
| VALUE | DOG |
| PLIST | (LEGS 4 EARS 2 TAIL --) |
| PACKAGE | #<Package INTERLISP> |

Type:

```
(SETQ INSPECTMACROS (CONS '((FUNCTION DOGSP) PROPNames GETPROP PUTPROP) INSPECTMACROS))  
(INSPECT 'RANDY)
```

--you will see:



RANDY Inspector

| | |
|------|------|
| LEGS | 4 |
| EARS | 2 |
| TAIL | 1 |
| HAIR | LONG |

Type:

```
(SETQ INSPECTMACROS (CDR INSPECTMACROS))
```

Testing the programmatic interface to INSPECTW
(assuming in **Interlisp** package)

type:

```
(CL:IN-PACKAGE "INTERLISP")
```

type:

```
(DEFINEQ (FETCHFROMLIST (AL PRP)
  (COND ((EQUAL PRP (QUOTE FIRST))
    (CAR (EVAL AL)))
    ((EQUAL PRP (QUOTE SECOND))
    (CADR (EVAL AL)))
    ((EQUAL PRP (QUOTE THIRD))
    (CADDR (EVAL AL)))
    ((EQUAL PRP (QUOTE FOURTH))
    (CADDR (EVAL AL))))))

(DEFINEQ (STOREINLIST (AL PRP NV)
  (SET AL
    (COND ((EQUAL PRP (QUOTE FIRST))
      (CONS NV (CDR (EVAL AL))))
      ((EQUAL PRP (QUOTE SECOND))
      (CONS (CAR (EVAL AL))
        (CONS NV (CDDR (EVAL AL))))))
      ((EQUAL PRP (QUOTE THIRD))
      (CONS (CAR (EVAL AL))
        (CONS (CADR (EVAL AL))
          (CONS NV
            (CDDR (EVAL AL))))))
      ((EQUAL PRP (QUOTE FOURTH))
      (CONS (CAR (EVAL AL))
        (CONS (CADR (EVAL AL))
          (CONS (CADDR (EVAL AL))
            (CONS NV NIL)))))))))

(DEFINEQ (PROPCOM (PRP AL INS)
  (SET AL (REVERSE (EVAL AL)))))

(DEFINEQ (VALCOM (VAL PRP AL INS)
  (SET AL (REVERSE (EVAL AL)))
  (INSPECTW.REDISPLAY INS NIL)))

(DEFINEQ (TITLECOM (INS AL)
  (SET AL (QUOTE (X Y Z A)))
  (INSPECTW.REDISPLAY INS NIL)))

(DEFINEQ (SELECTF (PRP VF INS)
  (PRINTOUT T T "YOU PICKED " PRP)
  (if VF then
    (PRINTOUT T " OR RATHER, THE VALUE OF " PRP T)
    else (PRINTOUT T " IT IS A PROPERTY" T))))

(DEFINEQ (PROPP (PRP AL)
  (LIST AL PRP)))

(SETQ ALIST ' (A B C D))

(SETQ WIND (INSPECTW.CREATE 'ALIST ' (FIRST SECOND THIRD FOURTH) 'FETCHFROMLIST 'STOREINLIST))
```

-- an inspect window appears



select a name (first,second, third or fourth) with the left button, then press the middle button - select set.

type in a new value for the selected element, confirm that the value changes.

Type ALIST on the exec window and notice that the list has changed.

type:

```
(INSPECTW.CREATE 'ALIST ' (FIRST SECOND THIRD FOURTH) 'FETCHFROMLIST 'STOREINLIST 'PROPCOM)
```

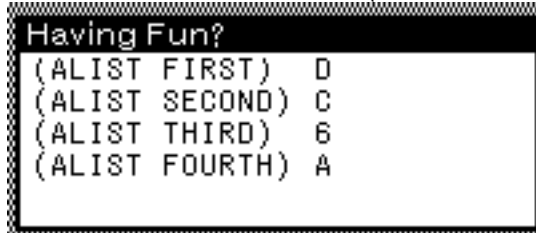
select a name (first,second, third or fourth) with the left button, then press the middle button. It should appear as if nothing has happened. Position the mouse in the title bar of the inspect window, press down the middle button - select refetch. *The values of Alist should be reversed.* Type ALIST on the exec window and notice that the list has changed.

Close all inspector windows

type:

```
(INSPECTW.CREATE 'ALIST '(FIRST SECOND THIRD FOURTH) 'FETCHFROMLIST 'STOREINLIST "HELLO THERE GUESS WHO?" 'VALCOM 'TITLECOM "Having Fun?" 'SELECTF WIND 'PROPP)
```

- the inspector window should appear where you placed the first one. Shape the window so you can view all of it. It should look like this (notice the new title):



select something from the left-hand column

-a window will appear telling you what you selected and that it is a property.

press the middle button on the same selection

- a message "HELLO THERE GUESS WHO" will appear in the prompt window

select something from the right hand column

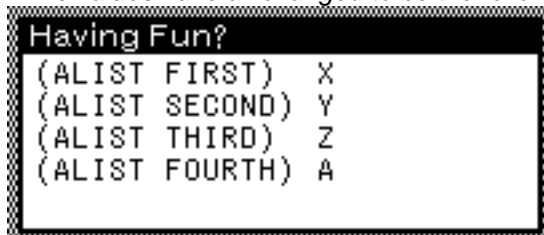
-a message will appear in the same window that appeared two steps back telling you what property the value you selected belongs to and that it is a value.

press the middle button on the same selection

-the values reverse themselves again

Put the mouse in the title bar and press the middle button

-The values have all changed to be the following.



Testing Report

Tested:

- different ways of creation: using, copying, reusing, smashing
- type?, typenamep, typename are tested for each type of record that maintains typename info: typerecord, arrayrecord, datatype
- for record types with and without typename info, provided a test form, and made sure it works via type?
- field name synonyms
- typenamep.- on an 1109
- records which turn into datatypes with type#>255 The sysout has more than 255 datatypes already, so this is well tested.
- defstruct's with types on the slots.
- checked that type number (as built by \\typeglobalvariable) are in the current package.
- Similarly, tested the interaction of packages with Interlisp records. (i.e. that the record name and the field names are in the same package.) Made sure that package qualified versions of things work with fetch and replace.
- recursive records with above.
- new record types: confirmed that the irm is correct.
- editing of records with SEdit (multiple completion).
- inspected any type of object the system can create
- specified type in ASTYPE arg to INSPECT
- user options INSPECTALLFIELDSFLG, MAXINSPECTCDRLEVEL, MAXINSPECTARRAYLEVEL, INSPECTPRINTLEVEL
- extensibility via INSPECTMACROS
- confirmed programmatic interface to INSPECTW's
- code inspectors on frames (both \realframep frames and others.),
- inspected values in frame variables.
- the setters on defstruct slots (including readonly-ness) for lists, vectors, :named and otherwise.
- examples of propprintfn

Left to be tested:

- typenamep should be tested on each machine. and was only tested on the 1109
- code inspectors on frames (both \realframep frames and others.), should be tested from both compilers
- all tests should be run compiled and interpreted on each machine.

Outstanding bugs and comments

7630 Smashing-arrayrecord fails

6950 WITH doesn't compile correctly. (new)

6606 re: record redeclaration.

4007 want to have - in record field names (closed)

1046 ensure that the editors can be successfully invoked from the inspector.

1820 make sure you can edit terminal tables

2019 problem with \applyinspectmacro (open)

2613 try inspecting processes (running and dead), other system types.

Note: there is no interaction between the inspector and *print-circle* and *print-pretty*

3. How to Create Sketch Elements

The elements a sketch consists of are text, boxed text, lines, boxes, polygons, curves, closed curves, circles, ellipses, arcs, arrowheads, and bit maps. This chapter explains how to create and change each kind of element, and how to change the way new elements will look.

To Use Text in a Sketch

In Sketch, text is provided by text elements. Each text element has some characters, a control point that positions it, and properties that determine the way it looks (e.g., boldness, font family) and how it is justified relative to its position (e.g., left, right, or center justification). A new text element is added by typing it in (see the section “To Type In Text,” below). You edit existing text elements by selecting within them and typing (see the sections “To Insert Characters Into a Piece of Existing Text” and “To Replace Characters in a Piece of Existing Text”). The size, font family, boldness, and italic properties, and the location of the text relative to its position, are changed using the Change command (see the section “To Change the Way Text Looks”).

In a sketch window, the three mouse buttons provide quick access to text editing and line drawing. Figure 8 summarizes these mouse button functions.

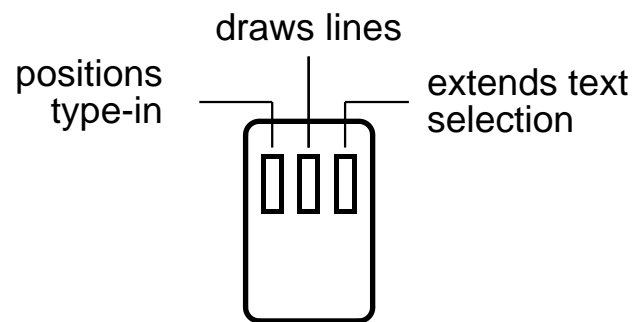


Figure 8. Mouse button text-editing and line-drawing functions

To Type In Text

Move the cursor to where you would like the text to be and press the left mouse button. The caret shape (I) will appear. Type the text. Typing a carriage return will start another line.

If a vertical bar (|) appears instead of a caret, the position you selected is in existing text (see the following sections). You cannot create a new text element in the middle of an existing text element. Any characters you type will be added to the existing text. To create a new text element, hold down the left button and move the cursor until the vertical bar (|) changes into the caret shape (^). At this point you can create a new text element by typing new text.



1. Put the cursor where the new text should be.



2. Type the text.
In this case, "a box."

Figure 9. Steps to insert new text

When characters are typed, a new piece of text is centered around the position of the caret. The alignment of the text relative to this position can be changed. For example, the text can be changed so that this position is its left edge. See the sections "To Change the Justification of Text" and "To Change the Properties of New Text."

To Insert Characters Into a Piece of Existing Text

Move the cursor to the place in the text where you want to insert the characters and click the left button. When you press the left button, the vertical bar (|) will appear between the characters. Type the characters. The typed characters will appear where the vertical bar is. If you hold the left button down while moving the cursor, the vertical bar will follow.

If you place the cursor over a piece of text and the vertical bar does not appear, the text may be contained in a group or it may be overflow from a text box. If it is in a group, you can edit this text only by ungrouping it first. See the section "To Use Groups." If it is overflow from a text box, you can reshape the text box to make it large enough to hold the text. See the section "To Change the Size of a Text Box."

You can insert characters at the beginning of a text element by selecting the left half of the first character. The vertical bar will appear in front of the first character when the cursor is positioned correctly. If you move out of the text, the vertical bar will change to a caret shape (^). If you type when the caret shape is visible, you will create a new text element rather than inserting characters into the existing one. You can insert characters at the end of a text element by selecting the right half of the last character.

To Replace Characters in a Piece of Existing Text

Move the cursor to in front of the first character to be replaced; press and release the left button. A vertical bar (|) will appear in front of that character. Move the cursor to the last character to be replaced; press and release the right button. After you press the right button, the text that will be replaced when you type is shown white-on-black. Type the new characters (see figure 10).

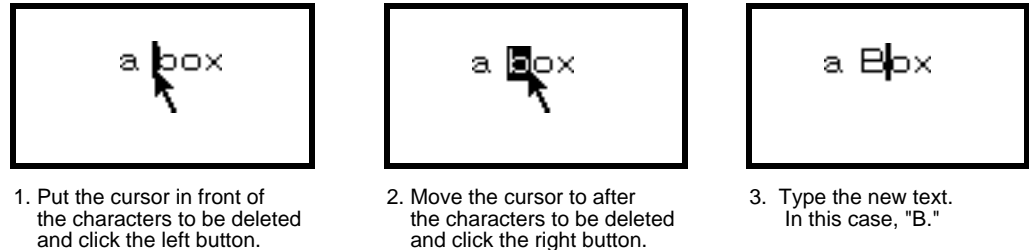


Figure 10. Steps to edit existing text

To Delete Characters in a Piece of Existing Text

Select the characters using the left and right buttons as described above in the section "To Replace Characters in a Piece of Existing Text" and press the delete key.


To Delete Text During Type-in

When you are typing text, you can use the backspace (BS) key to delete the previous character, and control-W to delete the previous word.

To Change the Way Text Looks

You can change the properties of text by using the Change command. After selecting the Change command, select a piece of text or a collection of pieces of text (see the section "To Select Sketch Elements") that you wish the change to apply to. You will then be presented with a menu (see figure 11) of possible ways of changing the text.

To Change the Bold and Italic Properties of Text


Select the Change command, move the cursor to the control point of the text (which will be marked with a ) and press and release the left button. The menu shown in figure 11 will appear. Select one of the items Bold, Unbold, Italic, or Unitalic.

Bold will make the selected text element appear in bold letters. Unbold will remove the bold property from the text if it was previously bold. Italic will make the selected text element appear in italics. Unitalic will change italic text to roman text.



Figure 11. Menu offered for changing text elements


To Change the Size of Text

Select the Change command, move the cursor to the control point of the text (which will be marked with a ) and press and release the left button. The menu shown in figure 11 will appear. Select one of the items Smaller Font, Larger Font, or Set Font Size.

Smaller Font will make the characters appear in the next smaller font. Larger Font will make the characters appear in the next larger font. Set Font Size will prompt you for a size to make the text.

Note: The font size is changed at the scale in which the text was originally entered. If you are viewing the text from a *zoomed view* (a window that has had its scale changed) the text size may not change. Or it may change more than a single size. If you have difficulty getting the size you want, enter a new piece of text, adjust its size, make the original piece of text have the same size (see the section "To Make Several Pieces of Text Look Alike"), and then delete the new text.

To Change the Font Family of Text


Select the Change command, move the cursor to the control point of the text (which will be marked with a ) and press and release the left button. The menu shown in figure 11 will appear. Select the item Different Font. A menu of the known font families and the item Other will appear. Select the name of the family you want or, if it is not there, select Other. If Other is selected, a small window with

the message "New family:" will appear above the sketch window. The caret will be blinking in it. Type the name of a font family, ending with a carriage return. If the selected or entered font family is not available in the size of the selected text element, a message is printed and nothing is changed.

When selecting the control point of the text, you can also select more than one text or text box element (see the section "To Select Sketch Elements"). If more than one text element is selected, any that have the same size as the first selected text element are changed to the new font family.

Note: the search for fonts encompasses any directories on DISPLAYFONTDIRECTORIES and may take a few minutes. (If font file servers are down or slow, it may take even longer.) If Sketch doesn't find a font that you believe exists, you can make that font the default font, both family and size (see the section "To Change the Properties of New Text") and retry the change.

To Change the Justification of Text

Select the Change command, move the cursor to the control point of the text (which will be marked with a ) and click the left button. The menu shown in figure 11 will appear. Select one of the items Left Justify, Center Justify, Right Justify, Top Justify, Bottom Justify, Middle Justify, or Baseline Justify. These commands will change where the text appears relative to its control point. Table 1 shows the effects of the different commands on horizontal and vertical justification.

| Vertical Horizontal | Baseline Justify | Middle Justify | Top Justify | Bottom Justify |
|------------------------|------------------|----------------|-------------|----------------|
| | | | | |
| Center Justify | Play | Play | Play | Play |
| Left Justify | Play | Play | Play | Play |
| Right Justify | Play | Play | Play | Play |

+ - control point of the text

Table 1. How the justification properties affect the position of text relative to its control point

To Make Several Pieces of Text Look Alike

Select the Change command, hold down one of the shift keys, move the cursor to the control point of the text that looks the way you want, and click the left button. Using the procedure described

in the section “To Use Menus and Submenus,” select the text elements you want to change to look like this one. When finished, release all the mouse buttons and the shift key. The menu shown in figure 11 will appear. Select the item Look Same. This will make all the selected text items be the same font size, face, and alignment as the one you selected first. The change is made to both boxed and unboxed text. This command is convenient for making text that was entered at different scales look the same.

To Use Boxed Text

Sketch has the capability of framing text and justifying text within the frame. The element that supports this is called a text box. Whenever the caret is inside a text box, the characters typed become part of the text within the box. The text in a box is broken into lines between words so that each line fits within the width of the box. If a single word in the text is wider than the box, it spills over. If there are more lines than fit in the height of the box, they spill over also. The characters outside the box cannot be selected for editing; the box must be enlarged to allow these to be edited.

A text box can be created by boxing a text element (see “To Put a Box Around Existing Text”), by typing a control-carriage return (see “To Create a Box to Put Text In”), or by using the **TEXT BOX** command (see “To Create a Sized Box to Put Text In”). The justification properties of the text specify the position of the text within the box (see “To Reposition the Text Within a Box”). The text inside a text box has the same looks properties as a text element (see the sections “To Use Text in a Sketch” and “To Change the Way Text in a Box Looks”).

The frame around the text is called its box. It has control points at its upper-right and lower-left corners. Moving one of these changes the size of the box (see “To Change the Size of a Text Box”). The box has thickness and dashing pattern properties that affect the frame (see “To Change the Border Thickness of a Text Box” and “To Make a Dashed Border Around a Text Box”) and a filling property that shades the part of the box not occupied by text (see “To Change the Filling of a Text Box”). Figure 12 shows some examples of text boxes.

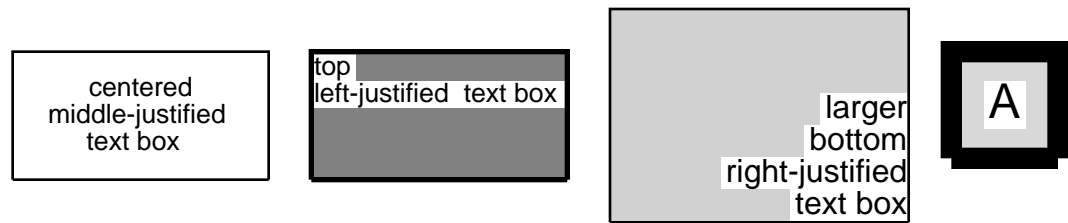


Figure 12. Examples of text boxes

To Put a Box Around Existing Text

Select the Change command from the command menu, move the cursor to the control point of the text you wish to box, and press and release the left button. A large menu titled Change Text How? will appear (see figure 11). Select the Box the Text item. The text will become part of a text box.

To Create a Box to Put Text In

Hold down the control key and press the return key. If the caret is inside a text box, a new text box of the same size will appear below it. Any characters typed will now go into the new box. If the caret is not in a text box, a new box will appear at the current cursor position. Note: you should hold down the control key until the new box appears.


To Create a Sized Box to Put Text In

Select the **Text Box** command from the command menu, move the cursor to one corner of the desired box location, press and hold the left button, move the cursor to the diagonally opposite corner, and release the button. While you are moving to the opposite corner, a gray outline of the box will be shown. When you release the button, the gray outline will be replaced by a solid one. You can stop this command by releasing the button when the cursor is outside the window.

To Change the Size of a Text Box

Move either control point of the text box using one of the point-moving methods described in the section "To Move Elements."

To Change the Border Thickness of a Text Box

Select the Change command, move the cursor to one of the corner points of the text box (which will be marked with a ) and press and release the left button. The menu shown in figure 14 will appear. Select the item Box Thickness. This will bring up a menu titled Change Size How? that contains the items Smaller Line, Larger Line, and Set Line Size. Selecting Smaller Line will

make the box outline be one size smaller than it is. Selecting Larger Line will make it one size larger than it is. Selecting Set Line Size will bring up a number pad menu (see figure 13). In this case, you should enter the size in screen points (1/72 of an inch) that you want the box thickness to be. The box thickness can be set to zero with the Set Line Size command, but the Smaller Line command will not make it less than one. If the thickness is zero, the box around the text won't appear, but any filling will remain and the text will still justify itself within the box's boundary.

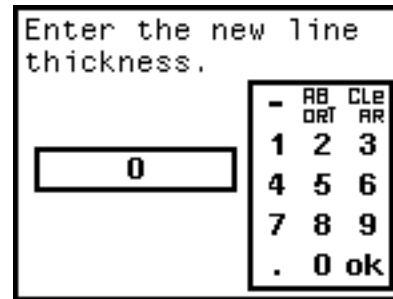



Figure 13. The number pad menu for entering line thickness

To Make a Dashed Border Around a Text Box

You can make the border of a text box dashed by specifying a *dashing pattern* (see the section "To Make a Dashed Line"). A dashing pattern is a sequence of numbers that indicates how many brush marks should be on and off. To specify a dashing pattern, first select the Change command, then move the cursor to one of the corner points of the text box (which will be marked with a ) and press and release the left button. The menu shown in figure 14 will appear. Select the Dashing item. This will bring up a menu titled New Dashing Pattern?, similar to the one shown in figure 34, that contains several dashing patterns and the items Other and No Dashing. Selecting one of the dashing patterns will make the border have that pattern. Selecting the Other item will bring up a series of number pad menus in which you enter, alternatively, the size of the black portion of the pattern and the size of the white portion of the pattern. The pattern can have as many alternations as you like. Number pad menus will continue to appear until you enter zero. When you enter zero, a menu is brought up showing what your new dashing pattern looks like and requesting confirmation. If you like the pattern, select Yes. If you don't like it, select No. If you select No, you will be given a chance to enter another series of sizes. If you select Yes, the box outline will be dashed according to your pattern. The pattern you entered will be remembered and will appear in the dashing menu next time you change the dashing.

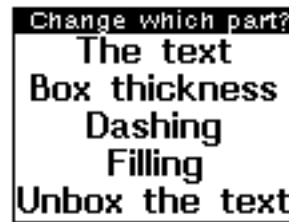




Figure 14. Change menu for text boxes

To Change the Filling of a Text Box

Select the Change command, move the cursor to one of the control points of the text box (which will be marked with a ) and press and release the left button. The menu shown in figure 14 will appear. Select the Filling item. From this point, follow the procedure given in the section "To Change the Filling of a Box."

To Change the Way Text in a Box Looks

Select the Change command, move the cursor to one of the corner points of the text box (which will be marked with a ) and press and release the left button. The menu shown in figure 14 will appear. Select the item The Text. The menu shown in figure 15 will appear. Select one of the items Different Font, Smaller Font, Larger Font, Set Font Size, Bold, Unbold, Italic, or Unitalic. The action of these items is described in the section "To Change the Way Text Looks."

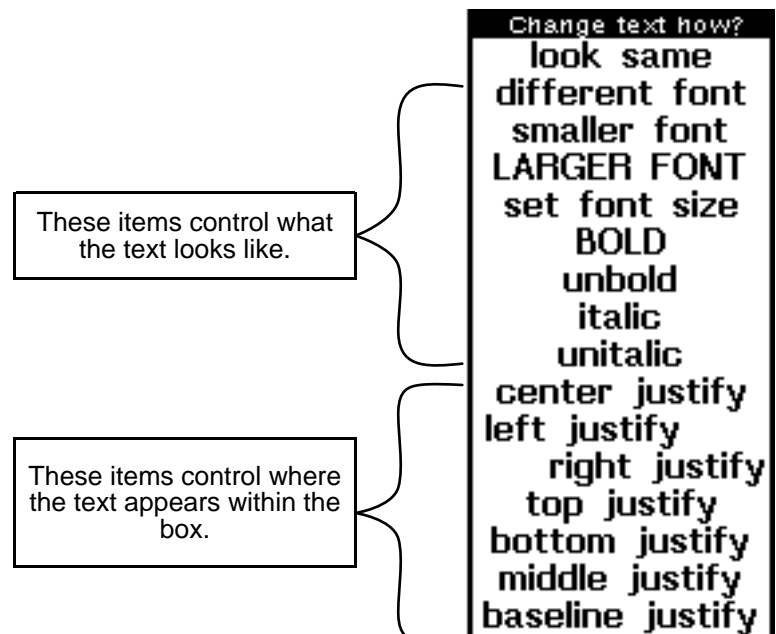




Figure 15. Menu offered for changing boxed text elements

To Remove a Box From Around Text

Select the Change command, move the cursor to one of the corner points of the text box (which will be marked with a ) and press and release the left button. The menu shown in figure 14 will appear. Select the item Unbox the Text. The box and all of its properties (such as filling) will be removed. If you want to keep the filling but not display the box, set the border thickness to zero (see the section “To Change the Border Thickness of a Text Box”).

To Reposition the Text Within a Box

Select the Change command, move the cursor to one of the corner points of the text box (which will be marked with a ) and press and release the left button. The menu shown in figure 14 will appear. Select the item The Text. The menu shown in figure 15 will appear. Select one of the items Left Justify, Center Justify, Right Justify, Top Justify, Bottom Justify, Middle Justify, or Baseline Justify. Left Justify will move each line of the text to the left edge of the box. Right Justify will move lines to the right edge. Center Justify will cause each line to be centered between the left and right edges. Top Justify will move the text so that the first line is at the top edge of the box. Bottom Justify will move it so that the last line of text is at the bottom edge. Middle Justify will move it so that the middle of the middle line of text is halfway between the top and bottom edges. Baseline Justify will move it so that the baseline position of the middle line of text is halfway between the top and bottom edges.


To Make the Text in Several Text Boxes Look Alike

The procedure is the same as the one described in the section “To Make Several Pieces of Text Look Alike.”

To Add Lines to a Sketch

Most sketch elements other than text, such as boxes, circles, and curves, are made up of lines. This section describes all the methods for adding lines, but does not tell you how to change their properties. The line properties are thickness, brush shape, and dashing, and you can change them using the information in the section “To Change the Way Lines in Elements Look.”

To Add a Line

To add a single line to a sketch, move the cursor into the sketch window and press the middle mouse button. While the middle button is held down, the cursor changes to . A mark will

follow the cursor on grid points, described in the section “To Use the Grid Display”). Move the cursor to where you want one end of the new line and release the middle button. Press and hold the middle button again and move the cursor to where you want the other end. A line will be stretched from the first position to the cursor position and will follow it. When the line is in the right place, release the middle button. If you move outside the window, the stretched line will disappear. If you release the button while the cursor is outside the window, no line is added. This provides a way of aborting if you change your mind while placing the second end of a line. If you move back into the window with the button still down, the line will reappear. If you want to change the position of the first point, click the left button while inside the sketch window, then start this procedure over.

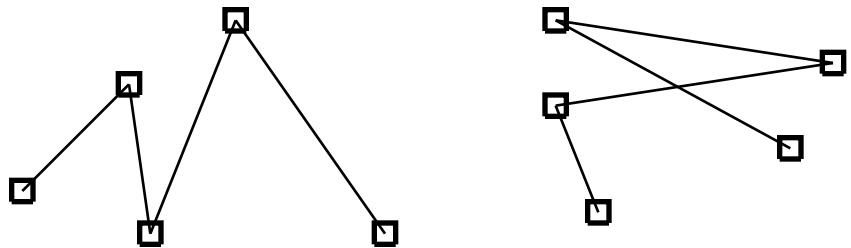




Figure 16. Two examples of connected lines with control points highlighted

To Add a Series of Connected Lines

You can add a series of connected lines using either the mouse or the  command. Using the mouse enables you to see and reposition the lines as you draw them, while using the  command may sometimes be faster.






With the Mouse

Move the cursor to over the Defaults menu command in the Sketch command menu, press and hold the left button, and slide the cursor out the right side through the triangle. A menu that includes the item Line will appear. Move the cursor over the Line item and slide out its right side. A menu that includes the item Mouse Line Specs will appear. Move the cursor over the Mouse Line Specs item and release the left button. A fourth (and final) menu with the title Connect Middle Button Lines? will appear. Select the Yes item from this menu. This procedure (which you only have to do once) has changed the mode of adding lines with the middle button to construct a series of lines. If you now draw lines as before, a new line will be stretched to the cursor position from the end of the previous line. To start a new series of lines, press either the left or the right mouse button in the sketch window. (A new series of lines is also started when you select a menu command.) You can stop the drawing of an individual line segment by moving the mouse cursor outside the window.



This does not start a new series of lines; moving back into the window and pressing the middle button picks up where the last line ended.

If you want to change the mode from entering connected lines back to unconnected lines, follow the above procedure but select the No item from the Connect Middle Button Lines? menu.

With the Command

Select the  from the Sketch command menu. The cursor will change to  and an  will follow the cursor. Move the cursor to each point that the lines go through and click the left mouse button. Each time you specify an endpoint, it is marked with a . After all the points have been specified, move the cursor outside the window and press the left mouse button. After you click outside the window, the line will be drawn connecting the . Figure 16 shows two examples of connected lines.

To Add Boxes to a Sketch

Sketch boxes can be either filled or unfilled; in addition, their borders may vary in thickness and brush shape, or be dashed. To add a box to a sketch, select the  command from the Sketch command menu. The cursor will change to . Move the cursor to one corner of the rectangle where you want the box. Press the left button. Holding the left button down, move the cursor to the opposite corner and release the button. While the left button is down, the rectangle will be highlighted in gray. When you release the button, a box will be added. To abort, move the cursor outside the window and click the left button. Note: the specified box must be entirely within the window. Figure 17 shows some examples of boxes.



*Figure 17. Three examples of boxes.
The left one has its control points highlighted*

To Change the Filling of a Box

Select the Change command from the command menu, then select the box or boxes whose filling you wish to change using the procedure described in the section "To Select Sketch Elements." You can include elements other than boxes as long as the first element you select is a box. The menu shown in figure 18 will appear.



Figure 18. Menu of properties you can change for boxes

Select the Filling item. A menu titled New Filling? similar to the one shown in figure 19 will appear.

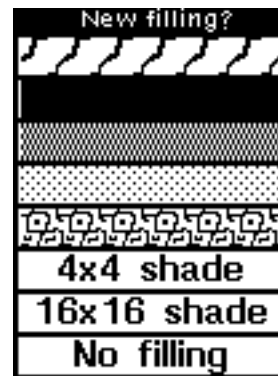


Figure 19. Menu of textures that can fill boxes

If the filling you want is shown on the menu, select it. If you want to remove the filling from a box, select the No Filling item. To create a shade that is not on the menu, select either the 4x4 Shade or the 16x16 Shade item. Selecting either item will cause the shade editor to appear (see figure 20). The 4x4 Shade item will allow you to create a 4-bit-by-4-bit shade. The 16x16 Shade item will allow you to create a 16-bit-by-16-bit shade. Construct the shade you want in the lower part of the window by turning points black with the left button and white with the middle button. Selecting Quit will cause the box to be filled with the newly constructed shade. The shade you created will be added to the menu for future selection. Advanced user note: the function (SK.CACHE.FILLING SHADE) can be called to add a filling to the menu.

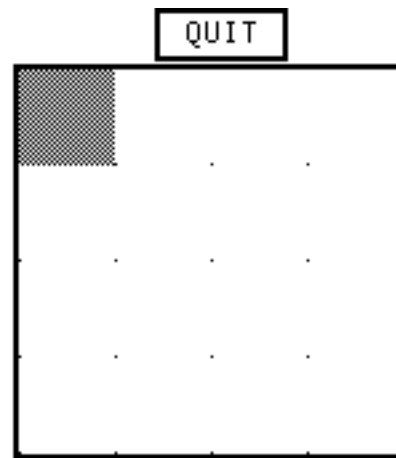






Figure 20. The shade editor

To change the thickness, dashing, or brush shape of a box's border, see the section "To Change the Way Lines in Elements Look."

To Add Polygons to a Sketch

Select the  command from the Sketch command menu. The cursor will change to  and an  will follow the cursor. Move the cursor to each vertex of the polygon and click the left button. When you select a point, it is marked with a . When you have selected all the points, move the cursor outside the window and click the left button. The points you selected will become the control points of the polygon. To abort, move the cursor outside the window and click the left button before selecting any points. Figure 21 shows some examples of polygons.

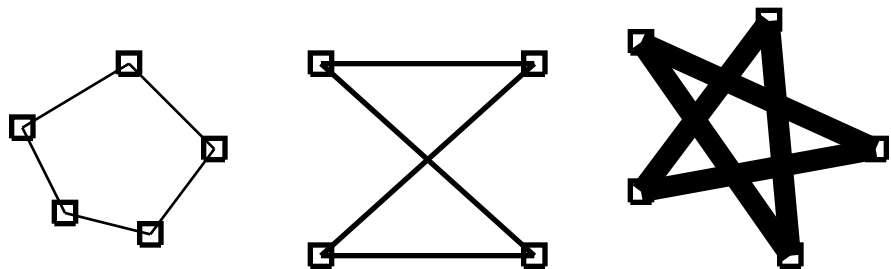






Figure 21. Three examples of polygons with control points highlighted

You can move any of the control points (using one of the point-moving commands described in the section "To Move Elements," below) to change the polygon.

To Add Curves to a Sketch

There are two kinds of curves you can add to a sketch: open curves and closed curves. This section tells you how to add each kind of curve.

To Add an Open Curve

Select the  command from the Sketch command menu. The cursor will change to  and an  will follow the cursor. Move the cursor to the points you want the curve to go through and click the left button. When you select a point, it is marked with a . When you have selected all the points, move the cursor outside the window and click the left button. The points where you clicked will become the control points of the curve. To abort, move the cursor outside the window and click the left button before selecting any points. Figure 22 shows some examples of curves.

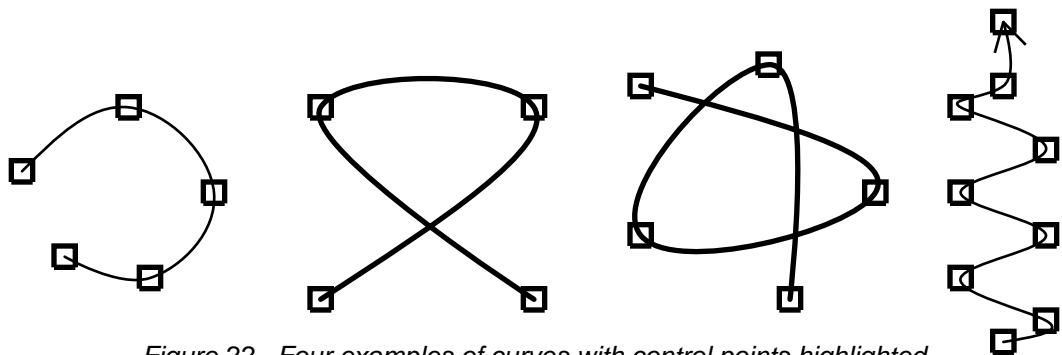






Figure 22. Four examples of curves with control points highlighted

You can move any of the control points (using one of the point-moving commands described in the section “To Move Elements”) to change the shape of the curve. In general, when you construct a curve, the closer together the points are the sharper the curve is; the farther apart they are, the smoother it is. The best way to learn how the control points affect the shape is to enter lots of different curves and move their points around.

To Add a Closed Curve

Select the  command from the Sketch command menu. The cursor will change to  and an  will follow the cursor. Move the cursor to the points you want the curve to go through and click the left button. When you select a point, it is marked with a . When you have selected all the points, move the cursor outside the window and click the left button. The points where you clicked will become the control points of the closed curve. To abort, move the cursor outside the window and click the left button before selecting any points. Figure 23 gives some examples of closed curves.

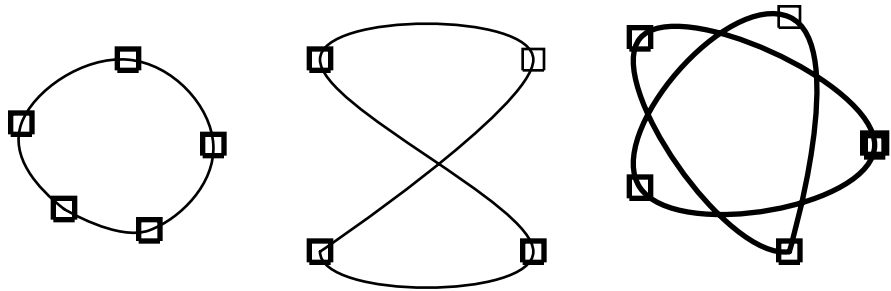






Figure 23. Three examples of closed curves with control points highlighted

You can move any of the control points (using one of the point-moving commands) to change the shape of the closed curve.

To Add Circles to a Sketch

Select the  command from the Sketch command menu. The cursor will change to . Move the cursor to the point you want to be the center of the circle and click the left button. The selected point is marked with  and the cursor changes to . Move the cursor to a point you want to be on the radius of the circle and click the left button. The circle will be added. The two points you selected are the control points of the circle. Either can be moved using one of the point-moving commands to change the radius and location of the circle. To abort this command, move the cursor outside the window and click the left button. Figure 24 shows two examples of circles.

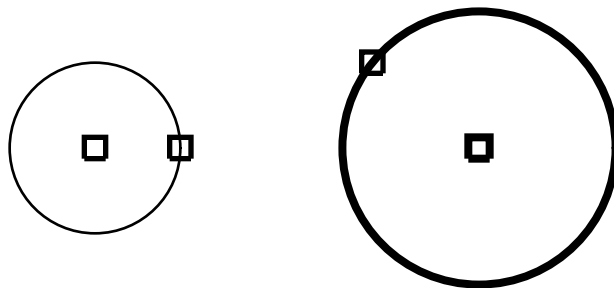








Figure 24. Two examples of circles with control points highlighted

To Add Ellipses to a Sketch

Select the  command from the Sketch command menu. The cursor will change to . Move the cursor to the point you want to be the center of the ellipse and click the left button. The selected point is marked with  and the cursor changes to . Move the cursor to the point you want to determine one radius and the orientation of the major axis of the ellipse (see figure 25). Click the left button. The selected point is marked with  and the cursor changes to . Move the cursor to any point that is the same distance from the center as you want the second radius to be and click the left button. The ellipse will be added.

The control points of the ellipse are the center and first radius point you selected and the point on the ellipse at the minor radius. Any of them can be moved using one of the point-moving commands to change the size, orientation, and location of the ellipse. To abort this command, click the left button with the cursor outside the window.

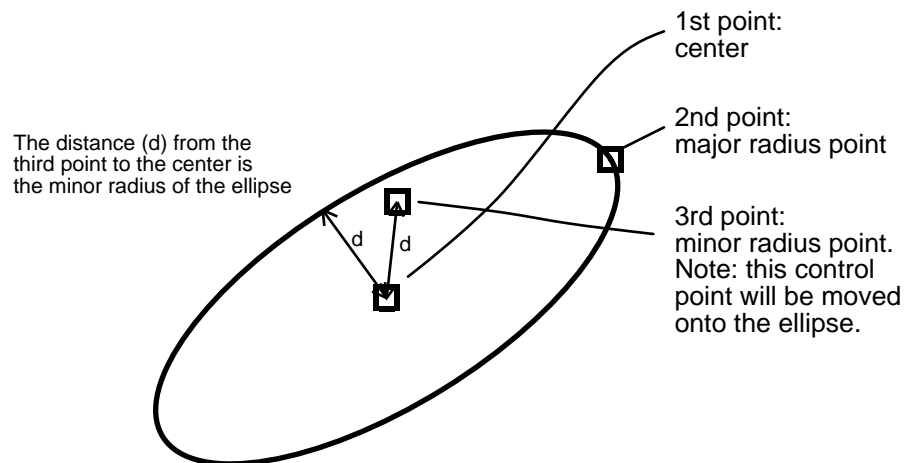


Figure 25. The control points for an ellipse

o Use Arcs in a Sketch

In Sketch, partial circles are provided by the arc command. An arc is characterized by three control points (see figure 26) and a direction. The shape and angle of the arc can be changed by moving any of the control points (using one of the point-moving commands). The angle of the arc can also be changed by setting the number of degrees the arc spans. (See the section "To Set the Number of Degrees an Arc Spans," below.) The direction determines whether the arc is traversed in a clockwise or a counterclockwise direction from the starting point. An arc has

thickness (size), brush shape, and dashing properties; it can also have arrowheads.

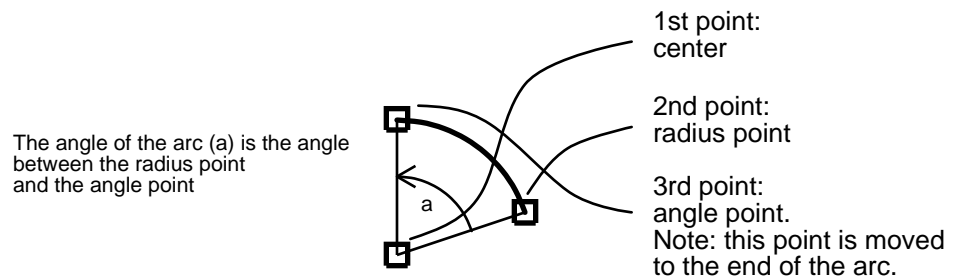









Figure 26. The control points of an arc

To Add an Arc

Select the  command from the Sketch command menu. The cursor will change to . Move the cursor to the center of the arc and click the left button. The selected point is marked with  and the cursor changes to . Move the cursor to one end of the arc and click the left button. This point determines the radius of the arc and its first end (see figure 26). The arc will begin at this point.

The selected point is marked with  and the cursor changes to . Move the cursor to the other end of the arc and click the left button. The arc will be added. To abort this command, click the left button with the cursor outside the window.

To Set the Number of Degrees an Arc Spans

Select the Change command, move the cursor to one of the control points of the arc (which will be marked with a ) and click the left button. The menu shown in figure 27 will appear.

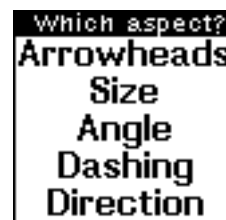


Figure 27. Menu of ways to change an arc

Select the Angle item. A number pad menu titled Enter Arc Angle in Degrees will appear (see figure 28). Enter the number of degrees that the arc should span by selecting digits from the number pad. When you are done, select the OK item. The selected arc (or arcs) will be changed to span the indicated number of degrees. The third control point (the angle point) is moved to accomplish this. To abort the command, select the Abort item from the number pad.

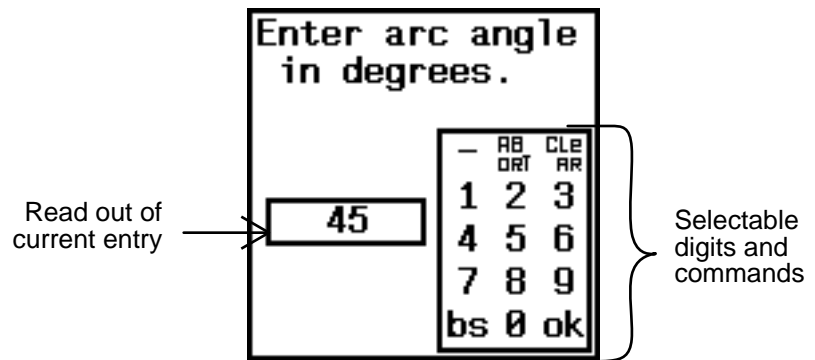



Figure 28. Number pad menu for changing the angle of an arc

To Reverse the Direction of an Arc

Select the Change command, move the cursor to one of the control points of the arc (which will be marked with a ) and click the left button. The menu shown in figure 27 will appear. Select the Direction item. A menu titled Which Way Should the Arc Go? with the items Clockwise and Counterclockwise will appear. Select Clockwise if you want the arc to go in a clockwise direction. Select Counterclockwise if you want the arc to go in a counterclockwise direction. Figure 29 shows the same arc with different directions.

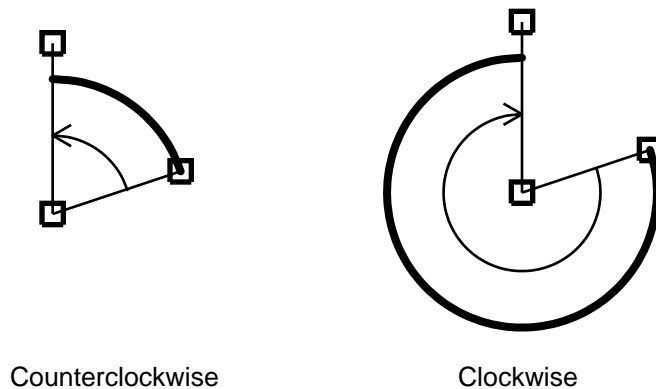


Figure 29. The direction of an arc

To Use Arrowheads in a Sketch

Arrowheads can be added to lines, open curves, and arcs. An arrowhead can be added to either or both ends of an element. It is then a property of the element and moves with it if it moves.

An arrowhead has a shape, a size, and an angle (see figure 30). The default arrowhead shape, size, and angle specify what a newly added arrowhead looks like. If you want to add several arrowheads that look alike, it is easiest to change the default arrowhead properties (see the section "To Change the Properties of New Arrowheads") before you add the arrowheads. The shape, size,

and angle of arrowheads on existing elements can be changed using the Change command.

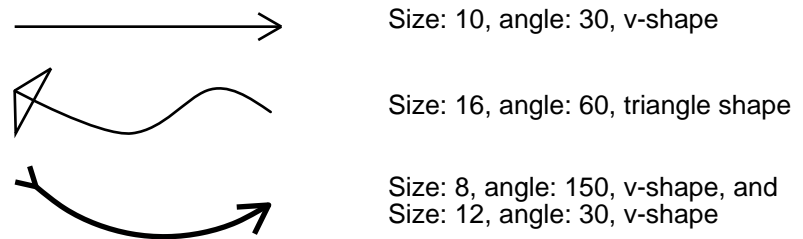


Figure 30. Sample arrowheads

To Add an Arrowhead to an Existing Element

Select the Change command from the command menu, then select the element or elements you wish to add an arrowhead (or arrowheads) to using the procedure described in the section "To Select Sketch Elements." You can include elements that do not allow arrowheads (such as text or circles) as long as the element first selected does. A menu will appear that contains all of the properties that can be changed in the first element you selected. It will include the item Arrowheads. Select it. The menu shown on the left in figure 31 will appear.

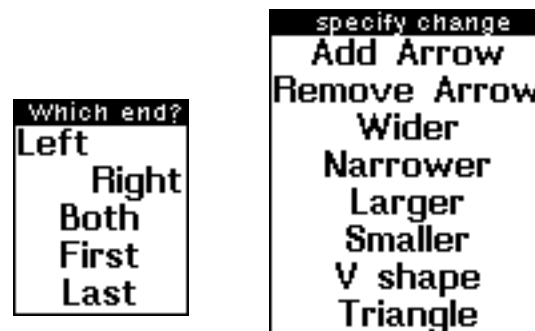


Figure 31. The menus for changing arrowheads.
The left one specifies which end of the line or curve will be changed.
The right one specifies how it changes

This menu allows you to specify the end or ends of the selected element(s) to which the arrowhead(s) will be added. Selecting Left will add the arrowhead to the leftmost end of the element (or the topmost if both ends have the same x position). Selecting Right will add the arrowhead to the rightmost end of the element (or the bottommost if both ends have the same y position). Selecting Both will add the arrowhead to both ends of the element. Selecting First will add the arrowhead to the end of the element that was entered first when this element was added. Selecting Last will add the arrowhead to the end of the element that was entered last.

After you select one of these items, the menu shown on the right in figure 31 will appear. Select the item Add Arrow. Arrowheads will be added to the specified ends of the selected elements that do not

already have arrowheads. To abort this operation, click the left button outside any menu instead of selecting an item.

To Remove an Arrowhead

Select the Change command from the command menu, then select the element or elements you wish to remove an arrowhead (or arrowheads) from using the procedure described in the section “To Select Sketch Elements.” You can include elements that do not have arrowheads as long as the element you select first does allow them. A menu will appear that contains all the properties that can be changed in the first element you selected. It will include the item Arrowheads. Select it. The Which End? menu shown in figure 31 will appear. Select the item that describes the end or ends (Left, Right, Both, First, or Last) from which the arrowhead(s) should be removed. (These items are described in the section “To Add an Arrowhead to an Existing Element.”) After you select one of these items, the Specify Change menu shown in figure 31 will appear. Select the item Remove Arrow. Arrowheads will be removed from the specified ends of the selected elements that have arrowheads. To abort this operation, click the left button outside any of the menus instead of selecting an item.

To Change the Way an Arrowhead Looks

Select the Change command from the command menu, then select the element or elements whose arrowhead(s) you wish to change using the procedure described in the section “To Select Sketch Elements.” You can include elements that do not have arrowheads as long as the first element you selected allows them. A menu will appear that contains all the properties that can be changed in the first element you selected. It will include the item Arrowheads. Select it. The menu shown on the left in figure 31 will appear. Select the item that describes the end or ends on which the arrowheads to be changed occur. (These items are described in the section “To Add an Arrowhead to an Existing Element.”) After you select an item, the menu shown on the right in figure 31 will appear. See figure 30 for some examples of arrowheads that may help you understand the following instructions.

Selecting the item Wider will increase the angle of the specified arrowhead by 10 degrees. Selecting the item Narrower will decrease the angle of the specified arrowhead by 10 degrees. Selecting the item Larger will increase the length of the specified arrowhead by two screen points. Selecting the item Smaller will decrease the length of the specified arrowhead by two screen points. Selecting the item Triangle will add a base to the arrowhead. Selecting the item V-Shape will remove the base from the arrowhead. To abort this operation, click the left button outside any menu instead of selecting an item.

If you want to change several properties of some arrowheads, it may be easier to set the default arrowhead specifications to be like the arrowhead you want (see the section “To Change the Properties of New Arrowheads”), remove the existing arrowheads, and then add the arrowheads again. The newly added arrowheads will have the default properties.

To Indicate That New Elements Should Have Arrowheads

Move the cursor over the Defaults command, press and hold the left button, and slide the cursor out the right side of the menu. Another menu will appear. In this new menu, with the left button still held down, position the cursor over the Line item and roll out its right side. A third menu will appear. In this menu, position the cursor over the Add Arrowhead item and release the left button. Two of the menus will disappear, and the menu titled Which End? shown on the left in figure 31 will appear. From this menu select the end of the elements you want to have arrowheads. Usually you will want to select Last. After selecting an item, whenever you add a line, curve, or arc, it will have an arrowhead on the specified end or ends.

To indicate that arrowheads should no longer be added to newly created elements, follow the above procedure but select the Neither item from the Which End? menu.

To Change the Way Lines in Elements Look

Lines, polygons, curves, closed curves, boxes, ellipses, circles, and arcs have thickness (size), brush shape, and dashing properties. You change these properties using the Change command. Figure 32 shows some examples of different-sized lines.

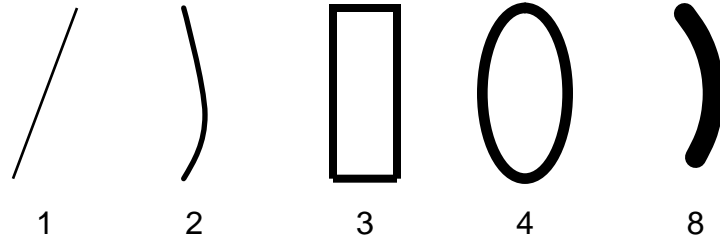


Figure 32. Examples of different sizes of lines

To Change the Size of a Line

Select the Change command from the command menu, then select the element(s) you wish to change using the procedure described in the section "To Select Sketch Elements." You can include elements that do not contain lines as long as the element first selected does; the change will not affect the others. A menu will appear that contains all the properties that can be changed in the first element you selected. Select the item Size. The menu shown in figure 33 will appear. Selecting the item Smaller Line will make all the lines one point thinner. Selecting the item Larger Line will make all the lines one point thicker. Selecting the item Set Line Size will bring up a number pad menu with the title Enter the New Line Thickness. Enter the size you want the lines to be by selecting the digits as you would on a calculator. When you have entered the number, select OK; all the lines will be made that

thickness. To abort the command, select the item Abort from the number pad.

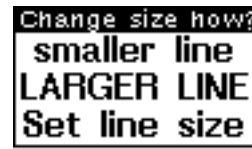


Figure 33. The menu of ways to change the line size of an element

To Make a Dashed Line

Select the Change command from the command menu, then select the element(s) you wish to change using the procedure described in the section "To Select Sketch Elements." You can include elements that do not contain lines as long as the element first selected does; selected elements without lines will be unaffected by the command. A menu will appear that contains all the properties that can be changed in the first element you selected. Select the item Dashing. A menu similar to the one shown in figure 34 will appear.



Figure 34. The menu of dashing patterns

Selecting one of the dashing patterns will make all the lines in the selected element(s) dashed with that pattern. Selecting the item No Dashing will make all the lines be solid; that is, it will remove the dashing from them. Selecting the item Other will prompt you for a new dashing pattern.

A dashing pattern is a sequence of numbers that indicates how many brush marks should be on and off. For example, the pattern (1 4 3 8) is , that is one on, four off, three on, eight off, repeated. When you select Other, a number pad menu with the title Number of Points On will appear. Enter the number of points you want to have on. When you have finished, select OK. A number pad menu with the title Number of Points Off will appear. Enter the number of points you want to have off. When you have finished, select OK. Number pads will continue to appear, giving you a chance to specify as long a dashing pattern as you like. After you have entered the last number in your dashing pattern, select OK when the number pad display has zero in it. A menu with the title Is This Pattern OK?, similar to the one shown in figure 35, will appear.



Figure 35. The menu presented to confirm a new dashing pattern

The first item will show the pattern you entered. If this is what you wanted, select the item Yes or the pattern; all the lines in the selected elements will change to that pattern. This pattern will also appear in the menu of dashing patterns the next time you change the dashing of an element. If you want to enter a different pattern, select the item No and you will be prompted for another dashing pattern. If you want to abort, select No and then select Abort from the number pad that appears. Selecting Abort from any number pad will abort the command.

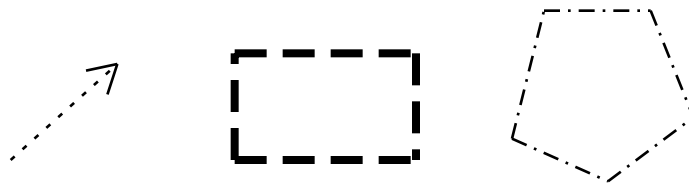


Figure 36. Examples of different dashing patterns for lines

Note: most printers do not support the dashing of splined curves, so curves, circles, ellipses, and arcs will not have dashed lines on hard copy. The effect will only be visible on the display.

To Change the Brush Shape of a Line

Select the Change command from the command menu, then select the element(s) you wish to change using the procedure described in the section "To Select Sketch Elements." You can include elements that do not contain lines as long as the first element you selected does. A menu will appear that contains all the properties that can be changed in the first element you selected. Select the item Shape. The menu titled Pick a Shape shown in figure 37 will appear. Select the brush shape you want. When you have selected one, all of the lines will be painted with that brush. To abort the command, select outside any of the menus.

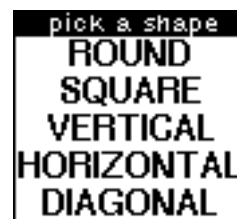


Figure 37. The menu of brush shapes

Note: because most printers do not support all of the available brush shapes, the effect may be visible on the display only.

To Change the Way New Elements Look

The properties an element has when it is first added to a sketch are called the *default* properties. For example, there is a default size that determines how wide newly added lines, curves, boxes, etc., will be. All these defaults can be changed using the Defaults command and subcommands (see figure 38). If you want to add a collection of elements that are different from the standard, changing the default is often the easiest way to do it. For example, if you want a bunch of extra-thick arrows, you could change the default properties of new lines to have a thickness of two and an arrowhead on the last point specified before you enter the lines for the arrows. Any defaults that you change are saved when the sketch is saved.



Figure 38. Submenu for the Defaults command

To

Change the Properties of New Text

1 Move the cursor over the Defaults command, press the left button, and slide out the right side through the triangle. The middle menu shown in figure 39 will appear. Move the cursor over the Text item and slide out the right through the triangle. The right menu shown in figure 39 will appear.

2 The Font Size will prompt you

for the size that new text should have; supply it using the same procedure described in the section "To Change the Size of Text." Selecting the item Font Family will prompt you for the family that new text should have, which you specify using the procedure described in the section "To Change the Font Family of Text." If the specified font cannot be found in the current default size, an error message is printed and the default is not changed. Selecting the item Horizontal Justification will prompt you for whether the new text should be left, right, or center justified. Selecting the item Vertical Justification will prompt you for whether the new text should be top, bottom, middle, or baseline justified. For more information about text justification, see the section "To Change the Justification of Text." Selecting the item Bold and/or Italic will prompt you for the bold and italic properties that new text should have.

To Change the Properties of New Text Boxes

Move the cursor over the Defaults command, press the left button, and slide out the right side through the triangle. The middle menu shown in figure 40 will appear. Move the cursor over the Text Box item and slide out the right through the triangle. The right menu shown in figure 40 will appear.

Selecting the item Horizontal Justification will prompt you for whether the text in new text boxes should be left, right, or center justified. Selecting the item Vertical Justification will prompt you for whether the text should be top, bottom, middle, or baseline justified. For more information about text justification, see the section "To Reposition the Text Within a Box." The font size, family, and bold and italic properties for new text boxes are the same as for text. See the section "To Change the Properties of New Text." The line thickness of the box is the same as the thickness of lines, so to change it see the section "To Change the Properties of New Lines."

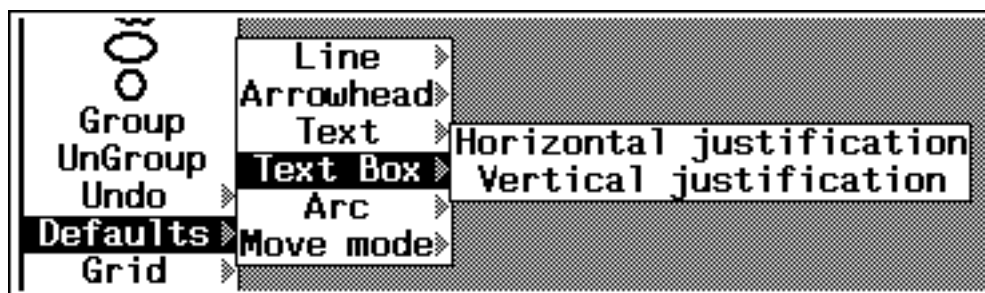


Figure 40. Submenu to set the default position of text inside a text box

To Change the Properties of New Lines

Move the cursor over the Defaults command, press the left button, and slide out the right side through the triangle. The menu shown in figure 38 will appear. Move the cursor over the Line item and slide out the right through the triangle. A menu containing the items Size, Shape, Add Arrowhead, and Mouse Line Specs will appear. Selecting the item Size will prompt you for a number that will become the thickness of any new lines, curves, circles, etc. Selecting the item Shape will prompt you for a brush shape that will become the shape of any new lines, curves, circles, etc. Selecting the item Add Arrowhead will prompt you for which end or ends, if any, of new lines, curves, and arcs should automatically get arrowheads. To change the characteristics new arrowheads have, see below. Selecting the item Mouse Line Specs enables you to choose whether lines created by middle buttoning in the window should be connected.

To Change the Properties of New Arcs

Move the cursor over the Defaults command, press the left button, and slide out the right side through the triangle. A menu that contains (among others) the item Arc will appear (see figure 41). Still holding down the left button, move to over Arc and slide out the right side through the triangle. A menu with the items Clockwise and Counterclockwise will appear (see figure 41). Selecting Clockwise will make new arcs go from their radius point to their angle point in a clockwise direction. Selecting Counterclockwise will make new arcs go from their radius point to their angle point in a counterclockwise direction.

The line thickness of arcs is the same as the thickness of lines. See "To Change the Properties of New Lines" if you want to change arc thickness.

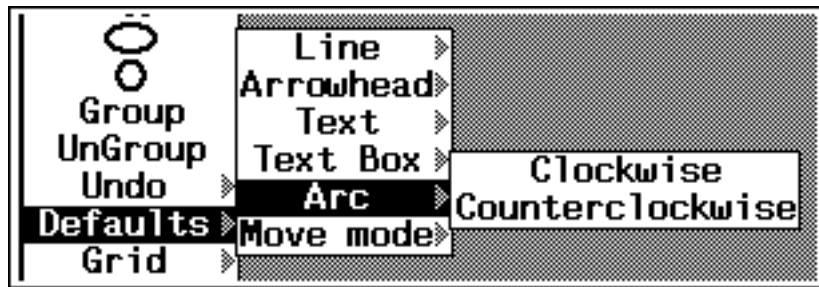


Figure 41. Submenu to change the default direction of arcs

To Change the Properties of New Arrowheads

Move the cursor over the Defaults command, press and hold the left button, and slide the cursor to the right through the triangle. The menu shown in figure 42 will appear. In this new menu, with the left button still held down, position the cursor over the Arrowhead item and slide it to the right again. A menu containing the commands Size, Angle, and Type will appear (see figure 42).

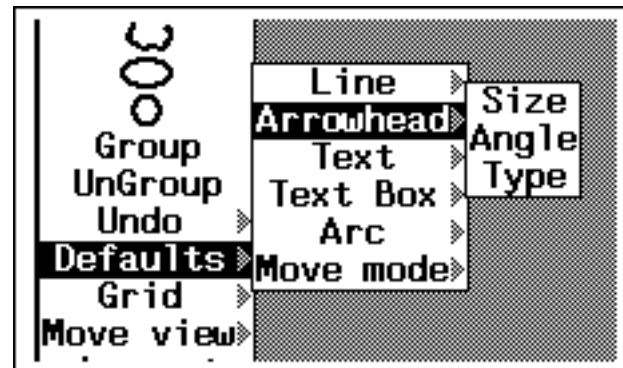


Figure 42. Menu of ways to change the default properties of arrowheads

From this new menu, select the arrowhead property you want to change. Selecting Size will bring up a number pad menu titled New Arrowhead Size in Screen Pts and giving the current default arrowhead size (length of the edges). Enter the number you want to become the size of new arrowheads, then select OK. You can abort the Size command by selecting Abort from the number pad menu.

Selecting Angle will bring up a number pad menu titled New Head Angle in Degrees, which gives the current number of degrees of the angle between the edges of new arrowheads. Enter the angle you want, then select OK. To abort, select the Abort item from the number pad menu.

Selecting the item Type will bring up a menu with the items V-Shape and Triangle, from which you can select the type of end you want new arrowheads to have. If you select V-Shape, your arrowheads will consist of two lines from the head. If you select


Triangle, new arrowheads will be triangles (two lines from the head and a line connecting their endpoints).


After this, whenever you add an arrowhead it will have the specified property.

To Use Bit Maps in a Sketch

You can include bit maps and other types of image objects in Sketch drawings. This section describes the procedures for dealing with bit maps because they are a particularly useful kind of image object, but many of the procedures described here apply to other image objects as well. You can place a bit map in a sketch with the standard copy-select mechanism. A bit map has a single control point at its lower-left corner and can be moved, copied, and deleted like other elements. Advanced user note: applying the Change command to an image object or pressing a button down while over its image in the sketch window calls that object's `BUTTONEVENTINFN`, which is often an editor for the object.

To Insert a Bit Map From the Screen

Move the cursor into the sketch window and click the left button. A caret () will appear. (If it doesn't appear, click again.) Move the cursor into the background (i.e., so it is not in a window). Hold down the copy key. On 1108 and 1186 keyboards, this can be either shift key or the Copy key; on Alto-style keyboards, it is either shift key. A menu with the single item Snap will appear. Select it.

The cursor will change to . Move the cursor to one corner of the region of the screen you want to include in the sketch. Press the left button. Holding the left button down, move the cursor to the opposite corner and release the button. While the left button is down, the region will be highlighted in gray. When you release the button, the message "Move the figure into place and press the left button" will appear in a small window above the sketch window. If this doesn't happen, you probably forgot to click in the sketch window first, and the bit map was inserted into whatever window was active. When the message appears, move the cursor into the sketch window and place the bit map image where you want it. When it is positioned, click the left button to insert the bit map.

To Insert a Bit Map of a Pop-Up Menu


Images of pop-up menus are often useful in illustrating documentation (see, for example, figures 38 and 42). To insert a pop-up menu in a sketch, you must break the process that is popping it up using the following procedure. While the pop-up menu is visible, type the help interrupt character. Initially this is control-G (that is, hold the control key down and type G); however some systems move it onto control-H. When you have typed the help interrupt character, a menu containing the names of all the current processes will appear. Select the process that has popped up the menu, which is usually marked with an asterisk. It is often

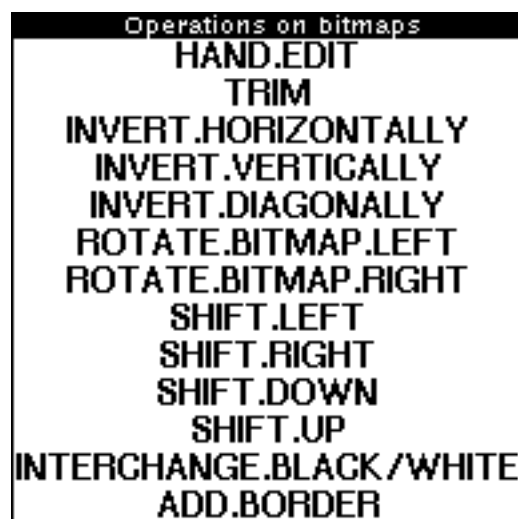
the process Mouse. If you selected the correct process, a break window will appear and the menu will still be on the screen. If the menu disappears, you broke the wrong process; move the cursor into the break window that appeared, press the middle button, and select OK from the menu that appears. This will continue the process you did break. Bring up the pop-up menu again, type the help interrupt character, and select a different process from the menu. When you get the pop-up menu image to stay up, follow the procedure described above in "To Insert a Bit Map From the Screen." Important: when you have finished getting the image, move the cursor into the break window that appeared, press the middle button, and select OK from the menu that appears. This will continue the process that popped up the menu. If you forget this, strange things will happen when you next use the pop-up menu.

It is often helpful to consider the background onto which the pop-up menu will appear. This is because you often get parts of the background when you copy a menu image. And the bit map editor Trim command (see below and figure 43) only trims away white space. So if there are black background bits, you will have to edit them out using the bit map editor (see the next section). This step can sometimes be avoided by changing the background to white (by typing (CHANGEBACKGROUND WHITESHAE) into the executive window). It is also helpful to move other windows away from the area where the pop-up menu will appear.

To Touch Up a Bit Map

Move the cursor over the bit map's image in the sketch window and press the left button. The menu shown in figure 43 will appear.

Select the Hand Edit item. The cursor will change into  and a large box outline will appear. This outline is the region the bit map editor will occupy. Move the box to the place on the screen where you want the bit map editor window to reside and click the left button. The bit map editor window will appear at that location (see figure 44).



*Figure 43. Menu of commands to edit a bit map.
You obtain this menu by pressing the left button
when the cursor is over a bit map image in a sketch window*

Edit the image by pressing the left or middle button in the large area at the bottom of the window. The image can be scrolled using the normal scroll bars if not all of it appears in the editing area. To quit, press the middle button while in the gray area at the upper-right part of the window. A menu will appear. Select OK to have the changes you made put back into the sketch. Select Stop if you want your changes disregarded. After you exit the bit map editor, the image in the sketch window is often incorrect. See the section "How to Clean Up the Display" for instructions on making it pretty again.

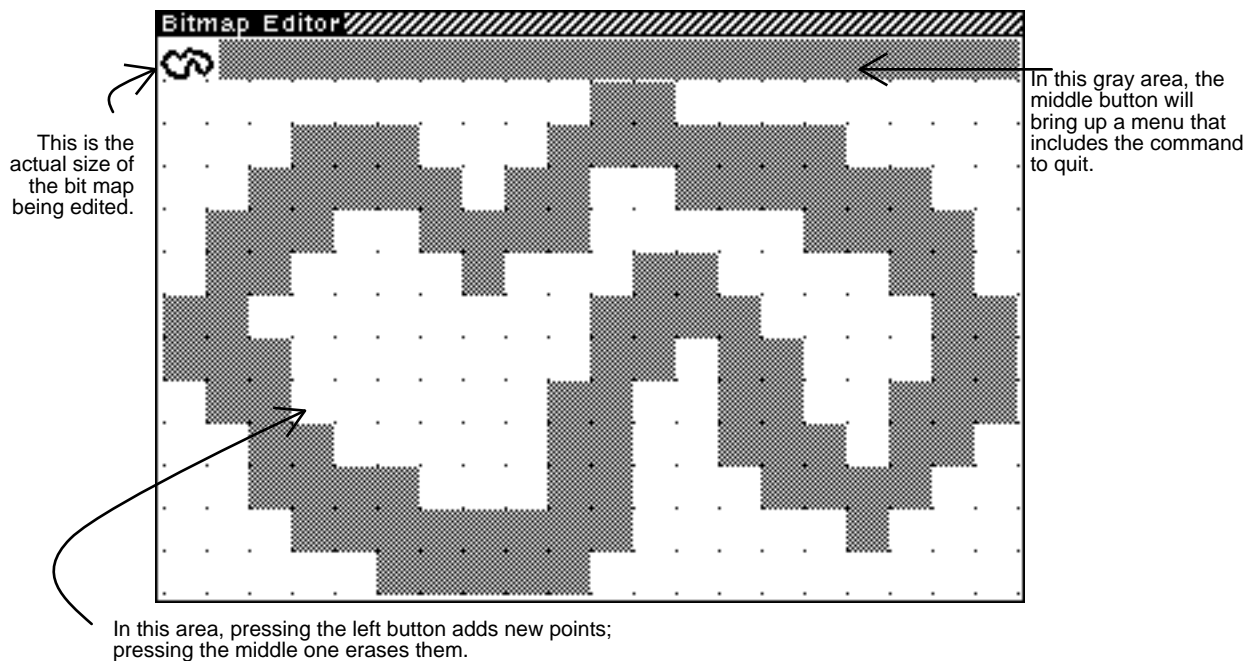


Figure 44. A bit map editor window

Another useful bit map editor command is Trim, available from the Operations on Bitmaps menu shown in figure 43. Trim will remove all the edge rows and columns that contain only white bits, making it easier to place lines and text around the bit map and saving storage space.

To Put a Border Around a Bit Map

Move the cursor over the bit map's image in the sketch window and press the left button. The menu shown in figure 43 will appear. Select the Add Border item. This will prompt you for the number of bits you want in the border, then allow you to edit a four-by-four shade that will be put in the border. You can add multiple borders. For example, many of the bit maps in this document have two points of white surrounded by one point of black.

For a complete description of the bit map editor, see the EditBitMap documentation in the *Lisp Library Packages Manual*.

as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse. Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

01UR.TEDIT

Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button

On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse. Local copy:

{DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse. Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC
 What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button

On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse. Local copy:
 {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILES>REQT PUBLIC:
 {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC
 What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse. Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match. Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match. Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match. Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse. Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match. Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

contains a capital X, a lower-case x will not qualify as a match.Middle button

On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse. Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse. Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? **In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions** Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match. Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILES>REQT PUBLIC:

{ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols.

E.g. if a search string contains acapital X, a lower-case x will not qualify as a match. Middle button On a 2-button mouse, pressing both buttons x once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPF|LES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC
 What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match. Middle button

On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse. Local copy: {DSK}<LISFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILS>REQ PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC

What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match. Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC

What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEDIT PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match. Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILES>REQ PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a **capital X**, a lower-case x will not qualify as a match. Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILIES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? **In a normal LOGOUT and boot without TEdit PUT and QUIT?**
Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match. Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILES>REQT PUBLIC:

{ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains acapital X, a lower-case x will not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

**Local copy: {DSK}<LISPFILes>REQT PUBLIC:
{ROSE}<REIDY>SPEC>TEDIT.DOC** What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button

On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse. Local copy:
{DSK}<LISPFILes>REQT PUBLIC:
{ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy:
{DSK}<LISPFILes>REQT PUBLIC:
{ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy:
{DSK}<LISPFILes>REQT PUBLIC:
{ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILes>REQT PUBLIC:
{ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button On a 2-button mouse,

pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILS>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? **In a normal LOGOUT and boot without Tedit PUT and QUIT?**

Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match. **Middle button** On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILS>REQT PUBLIC:

{ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols.

E.g. if a search string contains a capital X, a lower-case x will not qualify as a match. Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPF|LES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC
What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without
TEDIT PUT and QUIT? Definitions Case-sensitive Treating the upper and lower
 cases of the same letter as distinct symbols. E.g. if a search string
 contains a capital X, a lower-case x will not qualify as a match. Middle button

On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse. Local copy: {DSK}<LISPFFILES>REQT PUBLIC:
<ROSE><REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match. Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: (DSK)<LISPPFILES>REQ PUBLIC: (ROSE)<REIDY>SPEC>TEDIT.Doc
 What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without
 Tedit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same
 letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will
 not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will
 have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC
What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match. Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILES>REQ PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a **capital X**, a lower-case x will not qualify as a match. Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? **In a normal LOGOUT and boot without TEdit PUT and QUIT?**
Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. ~~E.g. if a search string contains a capital X, a lower case x will not qualify as a match.~~ Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK} <LISPFILES>REQT PUBLIC:

{ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a

capital X, a lower-case x will not qualify as a match. Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

**Local copy: {DSK}<LISPFILES>REQT
PUBLIC:**

{ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match. Middle button

On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse. Local copy:
~~{DSK}<LISPFILES>REQT PUBLIC:
{ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match. Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.~~

~~Local copy:
{DSK}<LISPFILES>REQT PUBLIC:
{ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match. Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.~~

Local copy:
{DSK}<LISPFILES>REQT PUBLIC:
{ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match. Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILES>REQT PUBLIC:
{ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match. Middle button On a 2-button mouse,

pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILes>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT?

Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match. Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILes>REQT PUBLIC:

{ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct

symbols.EUROPEAN

CLASSIC

| ^ ~ - ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
 \1234567890, ^

ÁÉÍÓÚÀÈÌÒÙ Ø

áéíóúàèìòù ø

ÂÊÎÔÛÄËÏÖ:“”

âêîôûäëïö;»«

ŽĄĆÃÕÑMÆı

ßăçăõñ æœ/ E.g. if a search string contains a capital X, a lower-case x will not qualify as a match. Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILes>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC

What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without

TEDIT PUT LOGIC

CLASSIC

|!@#%~&*()±+
 \1234567890-=

<[]>←∩∪≥[]
 >≤∩∪⇒∪∩≤[]

∇⇔[]- []≈Σ[]
 ∇⇔∇[]- []∏⊕[]

∃[]∞[]↔∇[]≤∅→
 ∃[]=⊗⇔∇[]∪∅←

TIMESROMAN

<[]>←∩∪≥[]
 >≤∩∪⇒∪∩≤[]

∇⇔[]- []≈Σ[]
 ∇⇔∇[]- []∏⊕[]

∃[]∞[]↔∇[]≤∅→
 ∃[]=⊗⇔∇[]∪∅←

and QUIT? Definitions Case-sensitive Treating the upper and lower cases of

ts for saving a file in a boot without Tedit PUT and QUIT?
UI? Definitions Case sensitive Treating the upper and
as distinct symbols the same letter as distinct
symbols. E.g. if a search string contains a capital
A, a lowercase a will not qualify as a match. Middle
button On a 2-button mouse, pressing both buttons
at once will have the same effect as pressing the
middle button on a 3-button mouse.

ts for saving a file in a normal floppy and boot without TEdit PUT and QUIT?

UI: Definitions Case

as distinct symbols, the same letter as distinct symbols. E.g. if a search string contains a capital 'A', a lower case 'a' will not qualify as a match.

button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

for saving a file in a normal mode and boot without TEdit PUT and QUIT?

Prerequisites Case Sensitive Treating the upper and distinct symbols the same letter as distinct symbols. E.g. if a search string contains a capital Xonder will case the will not qualify as a match. Middle button - On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy:
(DSK)<LISPPFILES>REQT PUBLIC:
(ROSE)<REIDY>SPEC>TEDIT.DOC What are the
requirements for saving a file in a crash? In a
normal LOGOUT and boot without TEDIT PUT and QUIT?
Definitions Case-sensitive Treating the upper and
lower cases of the same letter as distinct
symbols. E.g. if a search string contains a
capital X, a lower-case x will not qualify as a
match. Middle button On a 2-button mouse, pressing
both buttons at once will have the same effect as
pressing the middle button on a 3-button mouse.

```
{DSK}<LISPFILES>REQT PUBLIC:
{ROSE}<REIDY>SPEC>TEDIT.DOC What are the
requirements for saving a file in a crash? In a
normal LOGOUT and boot without Tedit PUT and QUIT?
Definitions Case-sensitive Treating the upper and
lower cases of the same letter as distinct
symbols. E.g. if a search string contains a
capital X, a lower-case x will not qualify as a
match.Middle button On a 2-button mouse, pressing
both buttons at once will have the same effect as
pressing the middle button on a 3-button mouse.
```

Local copy: {DSK}<LISPFILES>REQT PUBLIC:
{ROSE}<REIDY>SPEC>TEDIT.DOC What are the
requirements for saving a file in a crash? **In a normal
LOGOUT and boot without Tedit PUT and
QUIT? Definitions** Case-sensitive

Local copy: {DSK}<LISPFILS>REQT PUBLIC
{ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements
crash? In a normal LOGOUT and boot without TEdit PUT and Q
sensitive Treating the upper and lower cases of the same letter
E.g. if a search string contains acapital X, a lower-case x will not
match.Middle button On a 2-button mouse, pressing both button
same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILS>REQT PUBLIC
{ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements
crash? In a normal LOGOUT and boot without TEdit PUT and Q
sensitive Treating the upper and lower cases of the same letter
E.g. if a search string contains acapital X, a lower-case x will not
match.Middle button On a 2-button mouse, pressing both button
same effect as pressing the middle button on a 3-button mouse.

On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse. Local copy:
{DSK}<LISFILES>REQT PUBLIC:
{ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEDIT PUT and QUIT?
Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower case x will not qualify as a match.
Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse. Local copy:
{DSK}<LISFILES>REQT PUBLIC:
{ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEDIT PUT and QUIT?
Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower case x will not qualify as a match.
Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy:
{DSK}<LISPFILES>REQT PUBLIC:
{ROSE}<REIDY>SPEC>TEDIT.DOC What are the
requirements for saving a file in a crash? In a
normal LOGOUT and boot without Tedit PUT and QUIT?

Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT?

Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

08IMOB.TEDIT

Local copy: {DSK}<LISPPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT?

Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match. Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPPFILES>REQT PUBLIC:

{ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols.

EUROPEAN

CLASSIC

| ^~ - ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
\1234567890.~

ÁÊÍÓÚÀÈÌÒ Æ

áêíóúàèìò æ

| | | |
|------------------|--|--|
| RTP | | |
| DANTE#LEAF | | |
| POSEBOML#LEAF | | |
| LAFITEMAILWATCH | | |
| \LONBWATCHER | | |
| EXEC#Z | | |
| EXEC | | |
| \MSGATELISTENER | | |
| \RUPGATELISTENER | | |
| \TIMER.PROCESS | | |
| MOUSE | | |
| BACKGROUND | | |

| | | |
|------|-------|---------|
| BT | WHO? | KILL |
| BTV | KBD# | RESTART |
| BTV* | INFO | WAKE |
| BTV | BREAK | SUSPEND |

AEIOUAEIO:“”

âëïôûâëïö;»«

ZÅÇÃÕÑMÆŒ;

Bacão ñ æ œ / E.g. if a search string contains a capital X, a lower-case x will not be a **local copy**.

as a match. Middle button On a 2-button mouse, pressing both buttons at once will have

the same effect as pressing the middle button on a 3-button mouse.

requirements for saving a

Local copy: {DSK}<LISPFILES>REQT

PUBLIC:

{ROSE}<REIDY>SPEC>TEDIT.DOC What are

the requirements for saving a file in a crash? In a

normal LOGOUT and boot without TEdit

*PUT*LOGIC

CLASSIC

|!@#%~&*()±+

\1234567890-=

$$\langle \mathbf{u} \otimes \mathbf{v} \mid \mathbf{u} \otimes \mathbf{v} \rangle = \langle \mathbf{u} \mid \mathbf{u} \rangle \langle \mathbf{v} \mid \mathbf{v} \rangle$$
$$\langle C \in \mathcal{C} \Rightarrow \cup C \subseteq \mathcal{C} \rangle$$
$$\Delta \Leftrightarrow \square \dashv \square \approx \Sigma \square$$
 $\Delta \rightleftharpoons \nabla \rightarrow \blacksquare \dashv \square \perp \oplus \blacksquare$
$$\exists \mathbf{A} \in \mathcal{A} \leftrightarrow \nexists \mathbf{A} \in \emptyset \rightarrow$$
$$\mathbb{E}[\mathbf{X}] \equiv \mathbf{X} \Leftrightarrow \mathbf{X} \rightarrow \mathbf{X} \text{ and } \mathbf{X} \leftarrow \mathbf{X}$$

TIMESROMAN

$\langle \square \ominus \square \Leftarrow \cap \sqsubset \square \{ \} \rangle$

$$\gamma \in \mathcal{C} \Rightarrow \gamma \rightarrow \gamma \in \mathcal{C}$$
$$\exists x \in A \leftrightarrow \neg \forall x (x \in A \rightarrow \text{false})$$
 $\vdash \square(A \rightarrow B) \rightarrow (\Box A \rightarrow \Box B)$

and QUIT? Definitions *Case-sensitive*
Treating the upper and lower cases of
the same letter as distinct symbols.
E.g. if a search string contains a
capital X, a lower-case x will not
qualify as a match. Middle button

On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse. Local copy:

What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without Tedit PUT and QUIT?

Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct

symbols. E.g. if a search string contains a capital X, a lower case x will not qualify as a match.

Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

lower-case x will not equal y:

DISKDISPLES*RECT

~~(ROSE)<REIDY>SPEC>TI~~

requirements for saving a

normal LOGOUT and boot without FEdit PUT and QUIT?
Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy:

{DSK}<LISPFILES>REQT PUBLIC:

{ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match. Middle button On a 2-button mouse, pressing

both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct

symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT?

~~Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.~~Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

~~Local copy: {DSK}<LISPFILES>REQT PUBLIC:~~

~~{ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols.~~

~~E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.~~Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC
What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button

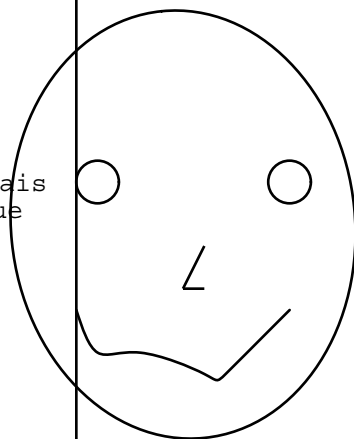
On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse. Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

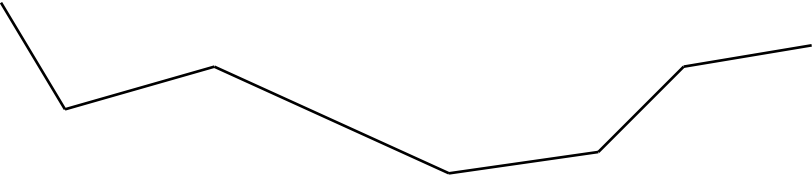
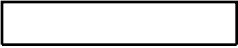
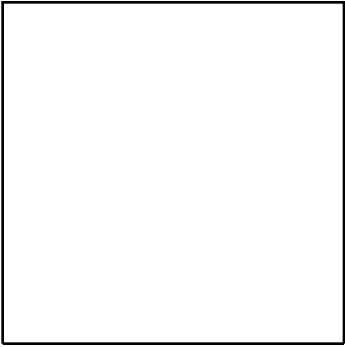
~~Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC~~
~~What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.~~Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

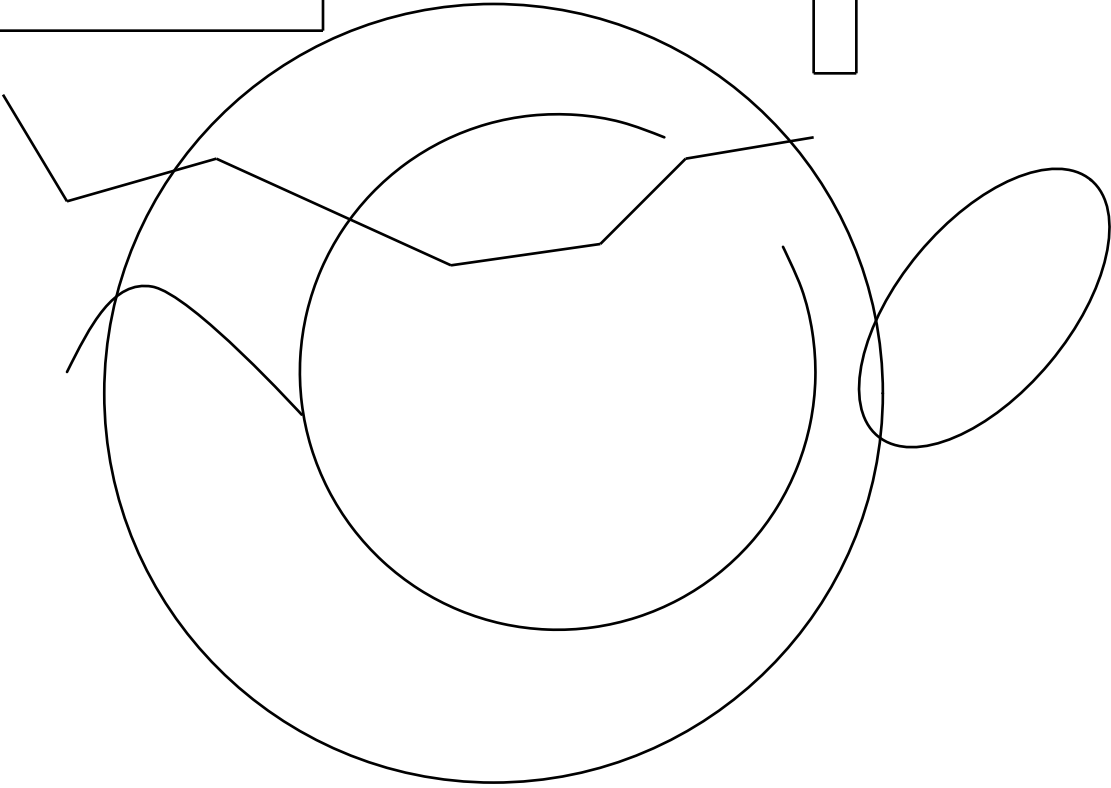
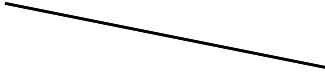
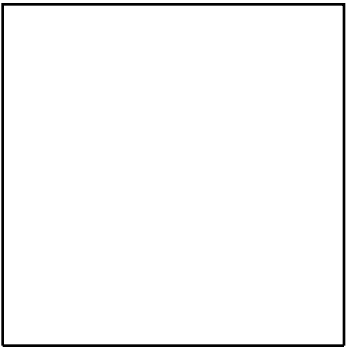
Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC
What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

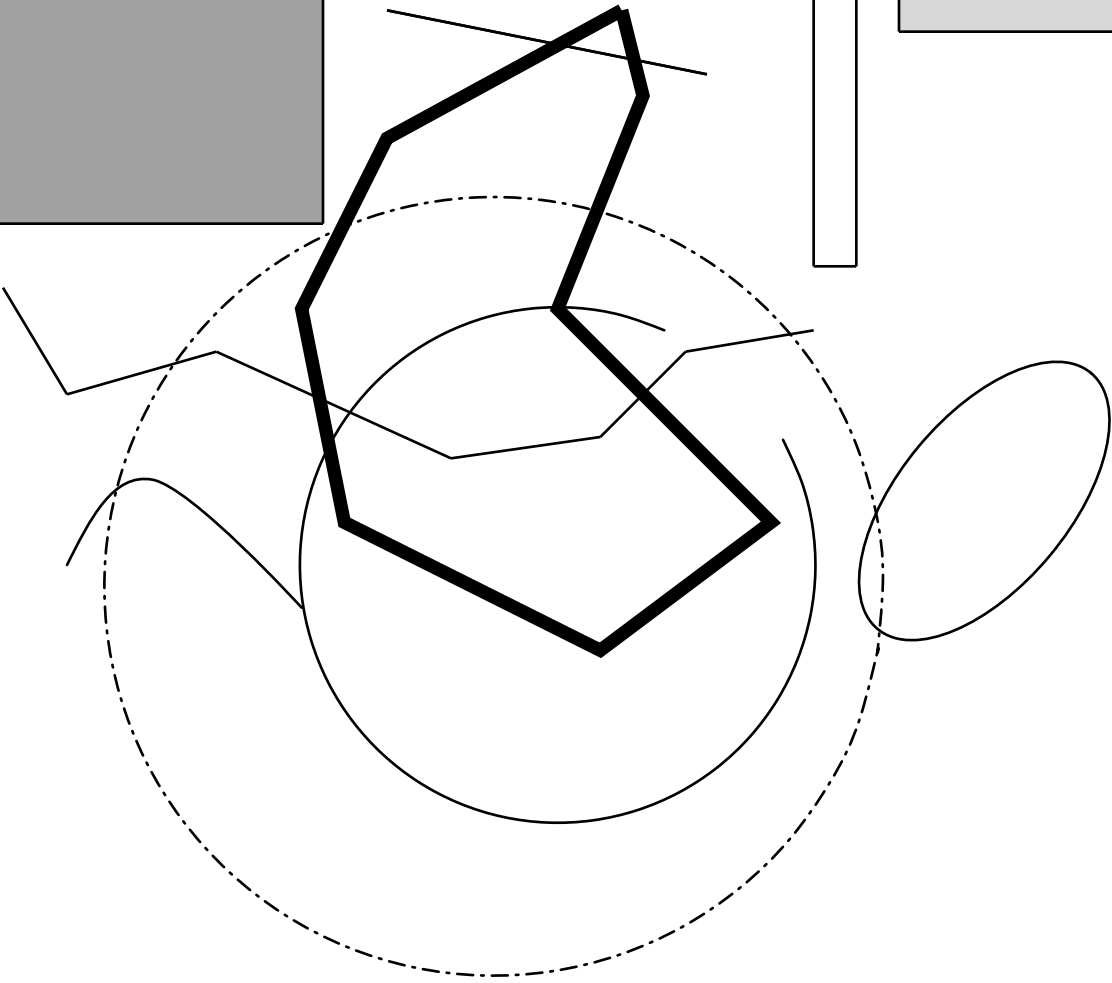
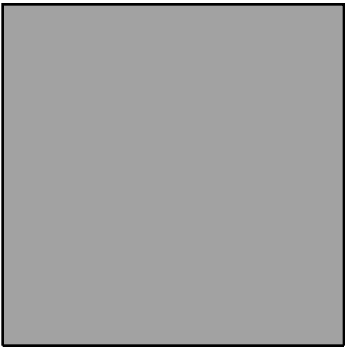
Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

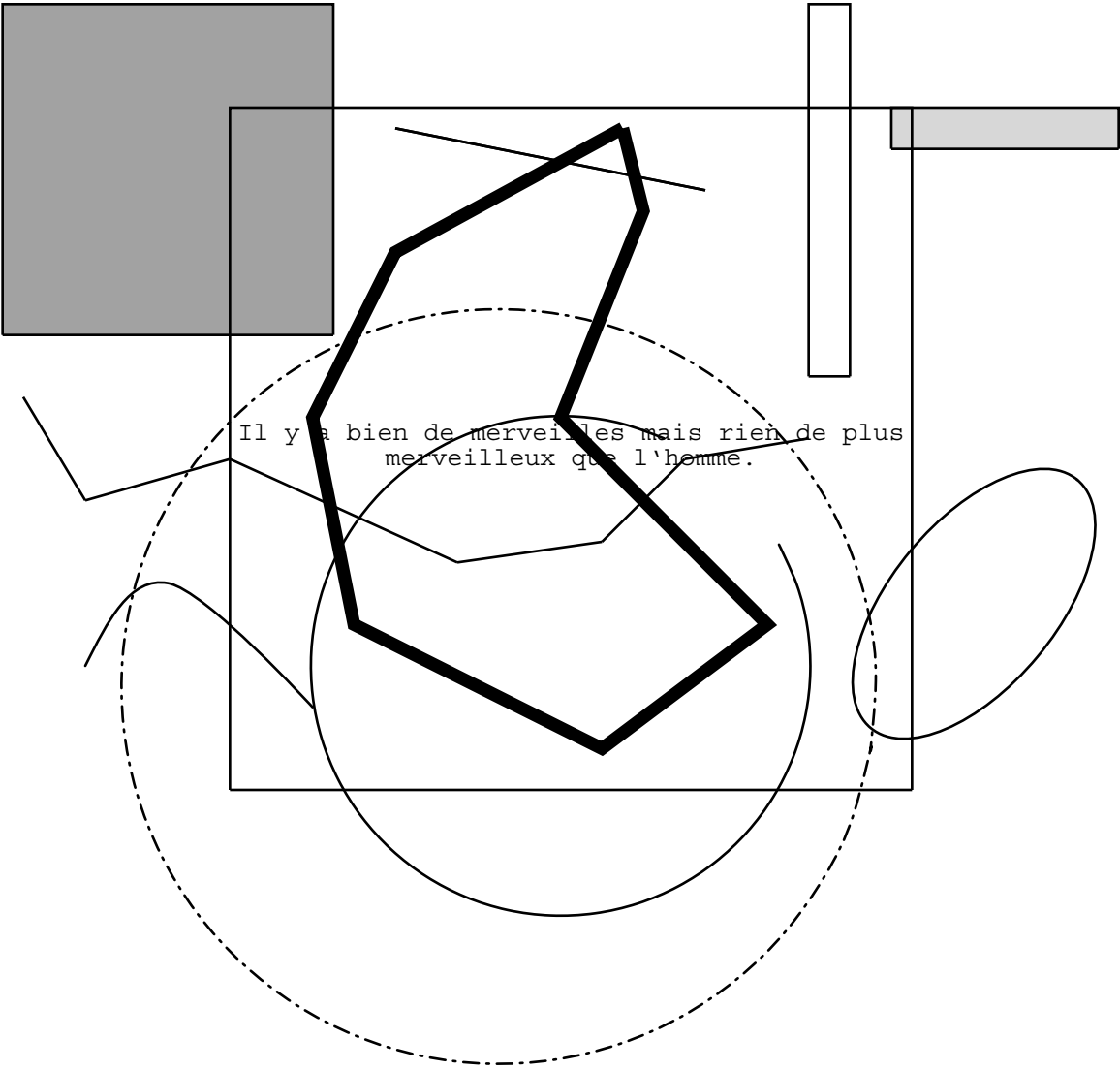
Il y a bien de merveilles, mais
rien de plus merveilleux que
l'homme.



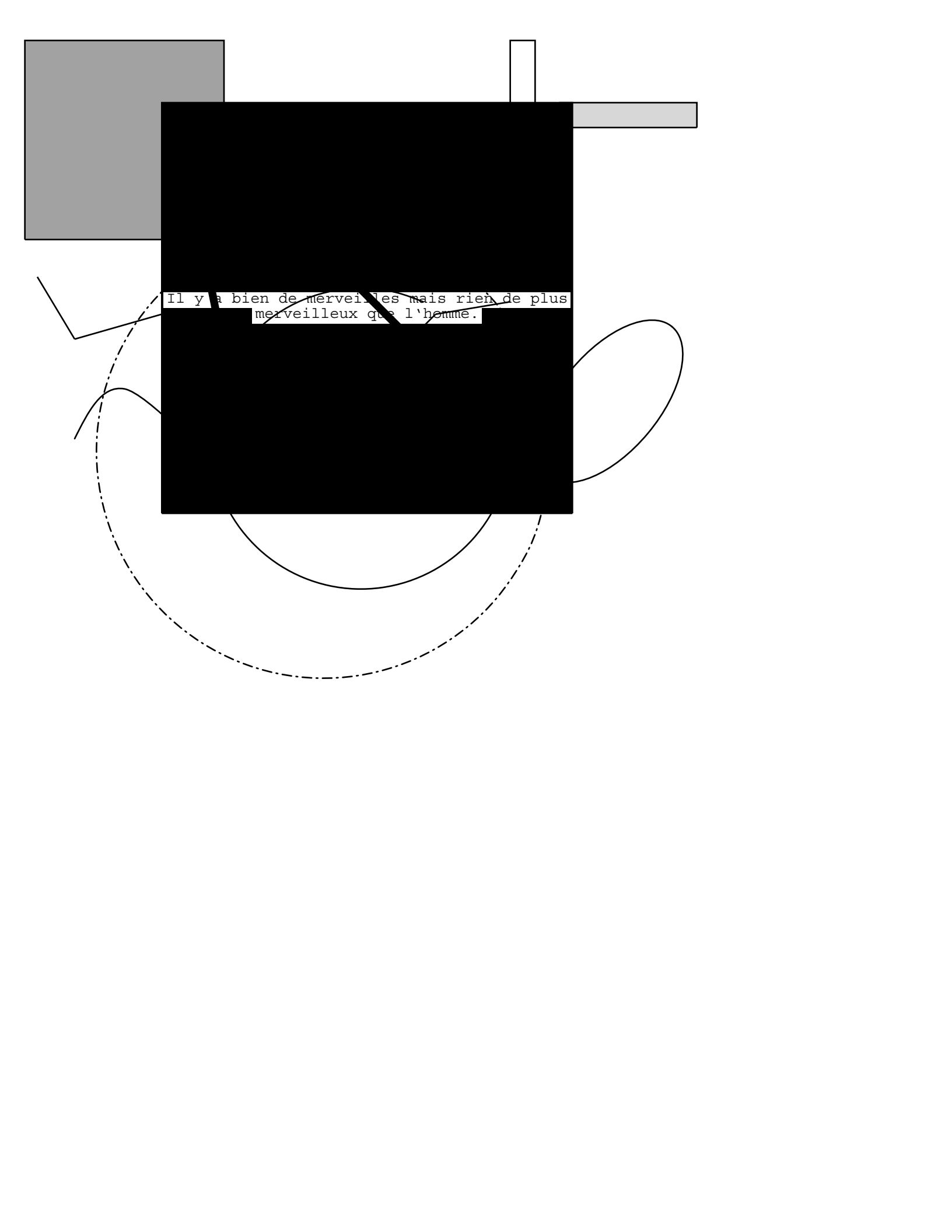








Il y a bien de merveilles mais rien de plus
merveilleux que l'homme.



Il y a bien de merveilles mais rien de plus
merveilleux que l'homme.

XAIS Testing Directory Structure

Language: The Interlisp & Common Lisp languages, interpreters, compilers

- Plans
- Results
- Auto
 - Test-Files {Eris}<Test>Language>Auto>
 - Aux-Files {Eris}<Test>Language>Auto-Aux>
- Hand
 - Scripts
 - Aux-Files

I/O: Streams, Windows

- Plans
- Results
- Auto
 - Test Files
 - Aux Files
- Hand
 - Test Scripts
 - Aux Files

Prog Environment: Exec, Code Editing, Debugging, Application Toolkits

- [subsection]
 - Plans
 - Results
 - Auto
 - Test Files
 - Aux Files
 - Hand
 - Test Scripts
 - Aux Files

Library: Individual Modules

- [Module name]
 - Plans
 - Results
 - Auto
 - Test Files
 - Aux Files
 - Hand
 - Test Scripts
 - Aux Files

LispUsers: Repository for obsoleted library tests when modules are demoted

- [Module name]
 - Plans
 - Results
 - Auto
 - Test Files
 - Aux Files
 - Hand
 - Test Scripts
 - Aux Files

Applications: "Customer Applications" for stress-testing system or subsystems

- [Module name]
 - Plans
 - Results
 - Auto

- Test Files
- Aux Files
- Hand
 - Test Scripts
 - Aux Files

[Subsystem, e.g. PCE]: One subdirectory per distinct product (PCE, Rooms, DEI, ...)

- Plans
- Results
- Auto
 - Test Files
 - Aux Files
- Hand
 - Test Scripts
 - Aux Files

File Naming

—**.test**

A file of DO-TEST forms, suitable for running with DO-TEST-FILE or DO-ALL-TESTS. Within a directory, files are named x-y-z-w....test, with more general descriptions to the left, and more specific naming to the right. For example, there might be several tests of Lists: LIST-CREATION.TEST and LIST-MODIFICATION.TEST. It is poor form to include the words REGRESSION, TEST, or such like in the file name: Regression tests should be folded into the main test file, and all of these files are tests, so that's redundant.

—**.u**

A script to be followed during hand testing. Within a directory, files are named x-y-z-w....u, with more general descriptions to the left, and more specific naming to the right. For example, the Exec might have several test files, e.g., Exec-PL.u, Exec-DIR.u, etc.

—**.plan**

A plan for testing a section of the system, a new product like PCE, etc.

—**m-d-y.log**

The results of running a test or series of tests.

This msg is stored on {eris}<lispcore>internal>library>do-test.tedit
The tester is on {eris}<lispcore>internal>library>do-test.dcom.

The main entries are the following:

(DO-TEST *name forms*)

A test succeeds if the final *form* returns a non-nil result. *Name* is just the name which can be an atom or string; strings are preferred. 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*)

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.

(EXPECT-ERRORS *error-types forms*)

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. Normal use of this form is:

```
(DO-TEST "a test"
  (EXPECT-ERRORS (type-of-error)
    (THIS-FORM 'SHOULD 'ERROR)))
```

(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

(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.

```
(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-SETQ, TEST-DEFUN, TEST-DEFMACRO

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

Relevant variables:

TEST-MODE

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

Defaults to "{eris}<lispcore>cml>test>test-results"

TEST-FILE-PATTERN

Defaults to ("{eris}<lispcore>cml>test>*.test;" "{eris}<lispcore>cml>test>*.x") which runs all the internal tests.

TEST-COMPILE

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

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"]).

Options to do-test-group.

:before allows for a setup form for a group of tests.
:after allows a form to be run after the tests without affecting results.

The normal form 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" ....)
)
```

Add new abbreviation characters below, with a tab between. To test, expand the right-hand instance of each, and print the result.

| | |
|----------|---------------------|
| b | b |
| n | n |
| m | m |
| T | fTf (this space) |
| d | d |
| D | D |
| s | s |
| , | , |
| ' | ' |
| " | " |
| ~ | ~ |
| 1/4 | 1/4 |
| 1/2 | 1/2 |
| 3/4 | 3/4 |
| 1/3 | 1/3 |
| 2/3 | 2/3 |
| c | c |
| c/o | c/o |
| % | % |
| -> | -> |
| | |
| ^ | ^ |
| <- | <- |
| DATE | DATE |
| >>DATE<< | >>DATE<< |
| L | L (pound sterling) |
| o | o (degrees) |
| Y | Y (yen) |
| + | + (+/-) |
| x | x |
| / | / |
| = | = (goes both ways) |
| p | p (paragraph) |
| r | r (registered mark) |
| tm | tm (trademark) |

SADF ASDF ASD•F ASDF ASDF ASDF ASDF ASDF ASDF ASDF ASDF ASDF ASDF ASDF ASDF ASDF ASDF

Introduction

AGAST is an attempt to produce a program that can write intelligent stories. With an eclectic combination of ideas from the work of both computer scientists and writers, we have produced the flexible core of what could be a very intelligent story teller.

Work being done in cognitive science, natural language processing, and other areas that are closely based on human actions fits neatly into the project. Story-telling is not one isolated behavior, but a combination of many; work from text generation, decision-making, story-planning, character development and other areas is needed.

AGAST uses a formula similar to those developed by various professional writers to teach beginning authors how to write stories*. This formula (described in detail in Part II) divides stories into five sections. AGAST attempts to model this formula by creating stories in the form of five inter-related sections.

AGAST is flexible because it "writes" stories in two stages. The first stage creates a story tree, where every action that happens is stored. As the tree is generated, the internal representation of the physical world (including locations, objects, and characters) is affected, which in turn affects the progress of the story.

A straight-forward depth-first traversal of the story tree produces a "chronological" account of the story. The second part, the text generator, walks the tree this way and thus tells the story. This part is extremely simple now; it just writes a sentence for each and every action, telling the story in excruciating detail. But since the story structure is unaffected by the telling, a different text

* i.e. to capture what is essential in a story as opposed to a random collection of sentences, or some other form of prosaic writing such as a newspaper article or master's thesis

generator could easily be used before, after, or instead of the one used now. Sequences of events could be summarized to different levels as needed, events could be told in varying orders, or two stories could be meshed together.

The fact that the story exists as a tree after its generation means also that actions can be undone and the story can take a new direction in the retelling. AGAST uses this feature to handle stories that end in story-telling failure. Like a human writer, AGAST can "change its mind" and rewrite the story to end successfully. This also means that AGAST doesn't need to plan every detail of the plot ahead of time. It can randomly generate complications for the plot, handle them using any sort of decision making process, and know that if it paints itself into a corner, it can either undo the actions that got it into trouble, or change the situation so that the characters can successfully handle the problem.

Related Work

A. James Meehan's "TALE-SPIN"

"Tale-Spin" [Meehan 76] seems to be the grandparent of computer story writing--at least, everyone who does work in the field must refer to James Meehan's ground-breaking work.

Like Tale-Spin, AGAST has a physical world, with objects and locations. Both have actions, although AGAST's actions are arranged in a writerly (GIVE, PICK-UP) rather than a formal (PTRANS, MTRANS) fashion.

Tale-Spin's stories are generated by goal stacks--the original goal puts other goals on the stack, the achievement of all of which completes the original goal. AGAST's stories have this feature as well. However, like AGAST actions, AGAST goals are arranged in a writerly fashion, each broken down into story parts. In addition there is the possibility of having multiple goals, all being solved simultaneously.

One example of this is when a character is looking for two different objects. First one object is sought, and when it is found, the other is sought. However, if the second object is run across in the search for the first, it is picked up and the search for it is never initiated. Although it isn't yet implemented in AGAST, characters can have other character's goals as subgoals, thus helping friends achieve their major goals.

Also, in storing actions and their side-effects as they occur, AGAST allows story revision and backpatching, which aren't conceivable in Tale-Spin.

Tale-Spin does have some level of social interaction, which AGAST is at present totally missing.

B. Natalie Dehn's thesis

Natalie Dehn [Dehn 81] makes the point that in writing a story, authors have a goal: to write an interesting story. Her project concentrates on author intentionality. AGAST attempts to emulate this goal with the plot formula that drives the story, and with backpatching that "saves" the story when it plots its way into a dead end.

However, there is also the point that characters must have goals. If they start without a specific goal, they are quickly given one, from simply staying alive to saving the universe. Dehn points this out (but not in terms of character goals) when she mentions justifying the situation a character finds him or herself in. A story goes wherever the author intends, but it won't be a very good story if the characters seem to be acting only on the author's whim. They should be following their own goals; their actions should make sense to them, not just to "The Story," of which characters generally aren't aware, anyway. AGAST attempts to combine the internal logic of goal-driven behavior of Tale-Spin with the author-intention-driven stories that Dehn promotes.

C. Michael Dyer's "BORIS"

Michael Dyer's work [Dyer 81] is more on story understanding than on story generation. Dyer's BORIS attempts to understand stories not only by general semantic, grammatical and lexical knowledge but by discerning the context that the story creates. AGAST creates and stores its context, but so far makes only a limited use of it. One example in which AGAST uses the context of an event is when an accident occurs (a character is injured--they trip, or some

such accident). If the character is just travelling or exploring, they can cure themselves (but only if they're carrying a medikit). However, if they are fighting or escaping, they can't take the time to do anything about the injury.

While it would surely be interesting to have BORIS read in AGAST stories and answer questions about them, it would be more interesting to have a BORIS-like program enhance the context that AGAST builds. A memory of past events would allow characters to "learn" and would make social interaction easier to simulate. For instance, suppose Frank killed Libby's cat. When Libby next meets Frank, the past event might make her want to get revenge on Frank, and thus would influence what she did during the meeting. As in Tale-Spin, she would know Frank was not to be trusted--but she would conclude it rather than knowing it from the start. From the examples in Dyer's paper, it would seem possible to use such a system to determine characters' attitudes toward other characters and their current emotional states.

D. Eduard H. Hovy's "PAULINE"

The actual text of AGAST stories is generated very simply--every object in a story tree knows how to print a description of the action it represents. This produces very lengthy, boring text (see sample stories).

Eduard H. Hovy [Hovy 87] discusses a much better text generation model. His program, PAULINE, groups related actions together and summarizes them, specifically mentioning only the "high points" of the event. PAULINE interprets the actions, draws conclusions, and adds them to its knowledge of the event. PAULINE can also "shade" what it tells, adding evocative words that can slant the meaning of the text, although the event is still accurately portrayed.

These abilities would greatly enhance the "story-ness" of AGAST's stories.

Instead of:

Libby swung her sword at the giant centipede, injuring its leg.

The giant centipede bit Libby, injuring her arm.

...and so on, each exchanging many blows and ending with:

Libby swung her sword at the giant centipede, injuring its head. Its head was severed and dropped to the floor. The giant centipede was killed.

A program such as PAULINE might be able to produce more writerly text:

Libby drew her sword as the giant centipede attacked. She slashed at the slaver creature as it bit at her. Howling with rage, the giant centipede sank its mandibles into Libby's left arm. Libby raised her sword and with a cry of desperation cut off the centipede's head.

Since AGAST's actions are already grouped and catalogued (a series of "injure" actions that constitute a "fight" are stored in a slot of a "fight" event), summarizing events and choosing weighted words that fit the situation (desperate, rage) and the characters in it ("slaver", since giant centipedes are defined as non-intelligent animals) should be relatively easy to do.

E. Michael Lebowitz's "UNIVERSE"

Michael Lebowitz's UNIVERSE program generates plots for soap-opera-like stories. This is different from most other work in the field in that stories in UNIVERSE are deliberately constructed not to end, but to have continuing characters moving from mishap to mishap. Below we examine two versions of UNIVERSE which have appeared in the literature.

[Lebowitz 83]

While AGAST is action-heavy, Lebowitz is primarily concerned with character consistency and development. Past events affect the personality (and thus actions taken) of characters. Like UNIVERSE, AGAST creates important

characters--the protagonist, the antagonist, anyone directly involved in the main goal--before the story starts. Unimportant characters--attacking monsters, for instance--are created on the fly.

AGAST reaches for this ideal with the Background section of the story providing the motivation for the story. However, UNIVERSE goes much further, with each character carrying around a changing history. Like Dyer's BORIS, a UNIVERSE-like program could significantly help the character development of AGAST's stories.

UNIVERSE also keeps track of character relationships in a more consistent way than AGAST. AGAST's characters can be related to one another (e.g. Libby is Frank's mother, and Frank is Libby's son), but if Frank is Stella's brother, all Libby knows is that she is somehow related to Stella. Relationships that change with time, such as marriage, exist, but are assumed permanent, with no history of past divorces or whatever.

[Lebowitz 87]

Using goal precedence and mutual-achievement criteria for goal selection, UNIVERSE nicely manages the interweaving of plots that is important for a complex, interesting story. AGAST currently has only one form of subplot implemented, the substory. Here the main goal is temporarily suspended while a subgoal (with a "subprotagonist") is "written" in a "meanwhile, back at the ranch..." type of story. The conclusion of the subgoal, usually with the subprotagonist and the protagonist joined as companions, then allows the completion of the major goal. The nesting of substories, however, can produce quite complex stories (Libby rescues Fred; they both rescue John; all three join with Natasha to continue the search for the Lost Ark of the Covenant...).

The "churning" of plots that UNIVERSE uses as one of its author goals is somewhat emulated by the introduction of obstacles and problems into the path of the protagonist. Plans have particular problems associated with them, and goals (which determine the type of story, such as the "quest") can also have special problems associated with them, especially at the climax of the story (which is a concept UNIVERSE doesn't have, since it writes "slice of life" narratives).

F. Schank and Abelson's scripts

Schank and Abelson [Schank and Abelson 77] discuss many of the methods used in story writing programs. One important idea is that of scripts--an outline of how to behave in particular situations. They allow both understanding and generation of simple stories involving frequently done events, such as eating in restaurants or taking the bus.

Actions in the AGAST story tree are grouped and stored in PLAN objects. Many plans are similar to scripts in that they generate a restricted series of actions that constitute a type of event. For instance, if the event is a FIGHT between two parties: first, a character (randomly chosen from the first party) injures one (randomly chosen from the second party), then a return injury is done. These actions are repeated until one of the parties has no one left who can continue fighting.

ASDF ASDF ASDF ASDF ASDF ASDF
QWER QWER QWER QWER QWER QWER

OKZXCV ZXCV ZXCV ZXCV ZXCV ZXCV.

LESSON A7: FETCHING A NEW SYSOUT ON A DANDELION

B. Burwell December 7, 1987

Filed on : {Phylum}<ISLDOC>A07■DaybreakSysoutFetch

Objective: This document provides a recipe for the installation of a new copy of an Xerox Lisp "Sysout" (i.e., the main Xerox Lisp software), on a Daybreak which resides on a local network. You will be retrieving a new copy of the Xerox Lisp software from a network-based file. This document does not tell you how to load a new sysout from floppy disks.

Discussion:

The Xerox Lisp software machine uses various resources, primary memory in particular, during its life cycle. Not all of these resources are relinquished and deallocated when no longer needed, and the machine tends to "fill up" after prolonged use. To avoid problems when running low on the available memory that Xerox Lisp will want to use, the software on the machine (i.e. sysout) should be periodically restored to its prior initial state. This is accomplished by loading into the computer a fresh copy of the software, which reflects a full complement of the resources that are supposed to be available. The frequency of such need varies according to use: busy programmers may require a new copy daily, while casual users using only the mail and editing facilities may survive several weeks or more. The need to load a new copy of the sysout may be heralded by seeing such error messages as "ARRAYS FULL". Also, when the system starts to get sluggish and it takes longer than usual for processes to complete this is generally an indication that a new sysout is needed.

Sometimes your machine may crash with an error that is unrecoverable until you get a new sysout. If you get a 0217, 1185 or some other error in the cursor, please refer to Lesson L5. If you are really stuck just get a new sysout.

The following assumes you have a working Dandelion that has an Xerox Lisp system installed and running. Additionally, this machine should be connected to an Ethernet and able to access the appropriate file server designated for your use. When you have completed the sequence of steps listed you should have a fresh copy of the Xerox Lisp sysout running and ready for service. All of your local disk files and permanently stored files on file servers will remain unchanged. Your personal additions may not change, unless you have previously modified your "user init file", which directs Xerox Lisp to customize your machine according to your individual specifications. (See Lesson H for more on Init files.)

In the following, comments are in *Italicized print*; you type the things that are underlined, and normal print represents statements issued by the computer.

Procedure:

1. Saving your work and running the Lisp Installer

IF THE MACHINE IS ALREADY ON and running Xerox Lisp, first save your work by doing the following: put any TEdit or Sketch files that you have made changes to, save any functions that you have defined, close any mail folders that you have open. Now logout by doing the following:

(LOGOUT)

IF THE MACHINE IS OFF, turn it on by moving the rocker switch on the main unit front panel toward the 1 position.

After a few seconds a number of icons will appear which will correspond with the function keys. Press the F1 key immediately followed by the 0 (zero) key. Do not hold both keys down at once.

This will look for and boot from a specific Mesa boot file on your local hard disk.

The Daybreak will come alive as a Mesa machine (i.e. running under the Pilot operating system).

2. Installer Version 1.0

Copyright (C) 1984, 1985 by Xerox Corporation. All rights reserved.

Processor = 0AA007B74H = 25200075564B = 2■852■158■324

Memory size = 1536K bytes

Shall I try to find remote scripts? (Y/N): N <CR>

>Online

Drive Name: RD0 <CR>

Pressing the carriage return at this point will bring the rigid disk (RD0) online.

3. Installing a new sysout

There are two choices for installing a new sysout. Since retrieving a new sysout from a file server on the Ethernet can take up to ten minutes, some people stash a fresh copy on

their LispFiles volume to save time. The sysout does take up space, though. If you already have a sysout on your LispFiles volume that you want to use, skip to step 5.

In this step you will actually fetch a new sysout onto the local disk. You will need to know ahead of time exactly which sysout you want and where it is stored. Refer to lesson M1 for a discussion about possible sysouts to load. The sysout is either on an NS file server or an IFS.

First you will open a connection to the file server and then fetch the sysout from that file server.

For a sysout on an NS file server

If the sysout is on an NS file server (e.g. IE:PARC:Xerox) then you need to make sure you are logged in with your NS id (you were previously when connected to Starfile Public).

> login
User: your NS id (e.g. Joe Public:PARC:Xerox) <CR>
Password: your NS password <CR>

> open
connection to: your NS file server (e.g. IE:) <CR>

For a sysout on an IFS:

If the sysout is on an IFS (e.g. Phylum or qv) you will first need to login with your Grapevine name.

> pup login<CR>
User: your Grapevine id (e.g. Jones.pa) <CR>
Password: your Grapevine password <CR>

> pup open
connection to: your IFS (e.g. Phylum) <CR>

Note: even if you give the name of an IFS that is running, it is possible to get an error returned. If you will get the message "No Route to host" simply execute the pup open command again until you get a connection. Alternatively, if you get the message "connection rejected by host" this means that the file server is very busy. The pup open command will keep trying to make a connection.

4. *You now have a connection open to a file server. You want to ask the Installer to fetch the sysout onto a partition on the disk.*

> Lisp Sysout Fetch

Logical volume: volume name where you want sysout stored (e.g. Lisp) <CR>

Source: directory on file server where sysout is located (e.g. <Lisp>Lyric>Basics>Full.sysout) <CR>

Expand volume: (answer Y unless you are fetching the sysout to be a fresh Lisp sysout)
Y <CR>

expanding. . . . (unless you said N above)

Shall I make this the physical volume: N <CR>

Fetching a sysout usually takes 5 to 10 minutes.

5. **Copying a fresh lisp sysout:**

If you didn't fetch the sysout into a fresh lisp sysout then go onto step 6. You will now copy the fresh lisp sysout from the partition you fetched it into to the partition that you want to run Lisp in. After that you will expand the destination partition.

> copy vmem

Source volume: your fresh lisp sysout volume (e.g. LispFiles) <CR>

Destination volume: your working lisp partition (e.g. Lisp) <CR>

copying.....done.

> expand volume

Logical volume: your working Lisp partition (e.g. Lisp) <CR>

expanding..... done.

6. **N.B. Unlike a Dandelion, the Daybreak requires that there be Lisp microcode in each volume that you want to run Lisp in.** You must have the right version of Lisp microcode which corresponds with the release of Lisp. For example, to run Lyric, you must use Lyric microcode. The Lisp microcode is available on Phylum, IE:PARC:Xerox and Starfile Public. In the following example, Lisp Lyric microcode will be retrieved from Phylum. Refer to Lesson A6 is Phylum isn't available. **If you are just getting a new sysout and are staying with the same release then continue on to step 7.**

> pup login<CR>

User: your Grapevine id (e.g. Jones.pa) <CR>

Password: your Grapevine password <CR>

> pup open
connection to: Phylum<CR>

> Lisp Microcode Fetch
Logical Volume Name: (the name of the volume e.g. Lisp) <CR>
Source: <Lisp>Lyric>Basics>LispDove.db
Shall I make this the physical volume? N <CR>

7. **Starting Lisp:** *You are now ready to go into Lisp.*

> Start Lisp
Logical volume: your lisp volume (e.g. Lisp) <CR>
Are you sure: Y <CR>

8. The screen will go dark and possibly display an odd pattern on the screen. The cursor will stop on 199 for a number of seconds and finally go to 1186. If the maintenance panel hangs on some other number refer to lesson L5.

As Xerox Lisp looks for and activates different tools, the screen may change slightly. This process will be complete when the Xerox Lisp prompt "1>" returns to the 'TTY' window.

At this point you should now be in possession of a fully functional Xerox Lisp machine, and it is ready for you to resume your work.

Notes on {Medley}<internal>test>

<test> subdirs:

admin, ARs, env, GC, IO , LANGUAGE, Library, loops, lyric, Maiko, tools

<test>Top level files

README.TEDIT (this file)

TEST-RESULTS (contains log from running AUTO tests from 1988)

DOT.read-me-first (originally .read-me-first)

<test>4045> Deleted

<test>Maiko>

Subdir AUTO, OBSOLETE

Top level files moved to OBSOLETE

STACKHAX (has CHECKSTACKSPACE, seems to get tangled up in it's own stack)

STACKTAKESI (seems to cause a stack overflow on opurpose, which leaves stack clean enough that URAID hard-reset recovers from. Suspect timeouts aren't correct)

BAD-XREF (no compiled file)

display.cl (says "from Texas Instruments")

subdirs moved <ARs **optests** & .dfasl (**AuxHAND**
ENDLESSPUSHES **AR-TEST-CASES.Auto-log**

<test>Maiko>AUTO>

OPCODES.TEST

OPCODES.DFASL

may need EXPORTS.ALL to compile

most tests succeed

test BITBLT-DIAGONALS and BITBLT-SLOPED-LINES fail

A little hard to debug because the inspector for 2D

Another cop y of **bbtests** and **optests.lisp**

<test>Maiko>OBSOLETE>

Probably incorporated into AUTO>

AREF-TESTER

ARRAY-TESTER.TEST

FLOAT-TESTER

MAIKO-UNWIND-TESTS

TESTER (compiled OK)

unwindtest
xclopcodetests

<test>Library>

4045xlpstream> RS232 deleted
 CASH-FILE> only hand
 GCHAX>Auto>
 HASH-FILE> only HAND
 MatMult>Auto>
 TEdit>Hand-Aux> samples to TEdit
 WHERE-IS> only

<test>loops>

LOOPS-SETUP.TEDIT
LOOPS-TESTER-.... files

test>Lyric>

(old-versions of **DO-TEST .dfasl** and **.tedit**)

<test>Tools>

AUTOTEST.TEDIT (originql AUTOTEST.TEDIT-orig)
AUTOTEST & .DFASL framework for running tests

DO-TEST & .DFASL (copied newer version from Medley
 internal/library)

DO-TEST.TEDIT on Writing Software Tests
 (many other files not reviewed yet)

XAIS Testing Directory Structure

Special Files: For saving overall-testing information

| | |
|-------------|--|
| Information | {Eris}<Test>, files named <i>.name</i> , e.g. “.read-me-first” , this file. |
| Procedures | {Eris}<Test>Admin> for overall procedure files, rather than specific test plans or scripts |
| Results | {Eris}<Test>ARs> for the results of AR test-case regression runs. |
| Tools | {Eris}<Test>Tools> for general-purpose files like do-test or do-test-menu. |

Language: The Interlisp & Common Lisp languages, interpreters, compilers

| | |
|------------|--------------------------------|
| Plans | {Eris}<Test>Language>Plans> |
| Results | {Eris}<Test>Language>Logs> |
| Auto | |
| Test-Files | {Eris}<Test>Language>Auto> |
| Aux-Files | {Eris}<Test>Language>Auto-Aux> |
| Hand | |
| Scripts | {Eris}<Test>Language>Hand> |
| Aux-Files | {Eris}<Test>Language>Hand-Aux> |

I/O: Streams, Windows

| | |
|------------|--------------------------|
| Plans | {Eris}<Test>IO>Plans> |
| Results | {Eris}<Test>IO>Logs> |
| Auto | |
| Test-Files | {Eris}<Test>IO>Auto> |
| Aux-Files | {Eris}<Test>IO>Auto-Aux> |
| Hand | |
| Scripts | {Eris}<Test>IO>Hand> |
| Aux-Files | {Eris}<Test>IO>Hand-Aux> |

Prog Environment: Exec, Code Editing, Debugging, Application Toolkits

| | |
|---------------------------------------|---|
| [subsection: Exec, Debugger, Editors] | |
| Plans | {Eris}<Test>Env> <i>subsection</i> >Plans> |
| Results | {Eris}<Test>Env> <i>subsection</i> >Logs> |
| Auto | |
| Test-Files | {Eris}<Test>Env> <i>subsection</i> >Auto> |
| Aux-Files | {Eris}<Test>Env> <i>subsection</i> >Auto-Aux> |
| Hand | |
| Scripts | {Eris}<Test>Env> <i>subsection</i> >Hand> |
| Aux-Files | {Eris}<Test>Env> <i>subsection</i> >Hand-Aux> |

Library: Individual Modules

| | |
|---------------|---|
| [Module name] | |
| Plans | {Eris}<Test>Library> <i>module</i> >Plans> |
| Results | {Eris}<Test>Library> <i>module</i> >Logs> |
| Auto | |
| Test-Files | {Eris}<Test>Library> <i>module</i> >Auto> |
| Aux-Files | {Eris}<Test>Library> <i>module</i> >Auto-Aux> |
| Hand | |
| Scripts | {Eris}<Test>Library> <i>module</i> >Hand> |
| Aux-Files | {Eris}<Test>Library> <i>module</i> >Hand-Aux> |

LispUsers: Repository for obsoleted library tests when modules are demoted

| | |
|---------------|---|
| [Module name] | |
| Plans | {Eris}<Test>LispUsers> <i>module</i> >Plans> |
| Results | {Eris}<Test>LispUsers> <i>module</i> >Logs> |
| Auto | |
| Test-Files | {Eris}<Test>LispUsers> <i>module</i> >Auto> |
| Aux-Files | {Eris}<Test>LispUsers> <i>module</i> >Auto-Aux> |
| Hand | |
| Scripts | {Eris}<Test>LispUsers> <i>module</i> >Hand> |
| Aux-Files | {Eris}<Test>LispUsers> <i>module</i> >Hand-Aux> |

Applications: "Customer Applications" for stress-testing system or subsystems

```
[Application name]
Plans      {Eris}<Test>Customer-Applications>application-name>Plans>
Results    {Eris}<Test>Customer-Applications>application-name>Logs>
Auto
  Test-Files {Eris}<Test>Customer-Applications>application-name>Auto>
  Aux-Files  {Eris}<Test>Customer-Applications>application-name>Auto-Aux>
Hand
  Scripts    {Eris}<Test>Customer-Applications>application-name>Hand>
  Aux-Files  {Eris}<Test>Customer-Applications>application-name>Hand-Aux>
```

[Subsystem, e.g. PCE]: One subdirectory per distinct product (PCE, Rooms, DEI, ...)

```
Plans      {Eris}<Test>subsystem>Plans>
Results    {Eris}<Test>subsystem>Logs>
Auto
  Test-Files {Eris}<Test>subsystem>Auto>
  Aux-Files  {Eris}<Test>subsystem>Auto-Aux>
Hand
  Scripts    {Eris}<Test>subsystem>Hand>
  Aux-Files  {Eris}<Test>subsystem>Hand-Aux>
```

File Naming

—.test

A file of DO-TEST forms, suitable for running with DO-TEST-FILE or DO-ALL-TESTS. Within a directory, files are named x-y-z-w....test, with more general descriptions to the left, and more specific naming to the right. For example, there might be several tests of Lists: LIST-CREATION.TEST and LIST-MODIFICATION.TEST. It is poor form to include the words REGRESSION, TEST, or such like in the file name: Regression tests should be folded into the main test file, and all of these files are tests, so that's redundant.

—.u

A script to be followed during hand testing. Within a directory, files are named x-y-z-w....u, with more general descriptions to the left, and more specific naming to the right. For example, the Exec might have several test files, e.g., Exec-PL.u, Exec-DIR.u, etc.

—.plan

A plan for testing a section of the system, a new product like PCE, etc.

—m-d-y.log

The results of running a test or series of tests.

AUTOMATED TEST HARNESS INTERFACES

This document specifies the interfaces to the automated tester harness. The harness is composed of two parts: the **top-level tester** and the **individual test handlers**. The name of the file to load for this is `AUTOTEST.LCOM` in the top level of the `{MEDLEY}internal/test` directory.

The top-level tester is set up similarly to the package `FileBrowser`. Items are selected in the same manner as `FileBrowser`, and are displayed similarly. The portions of the display are as follows (from top to bottom):

1. A prompt window for displaying messages and getting new input.
2. A command menu with the following commands:

TEST Tests sequentially each of the items selected in the test files window. Testing consists of loading the file containing the test suite, calling a function which has the same name as the `NAME` field of the filename (this function must return `NIL` iff the test suite is not successful), then undoing (as best as possible) the side-effects of loading and running the test suite. The function which is called is passed one argument: the name of the directory that the test suite came from (including the host name). If this item is selected with the middle button, then first it asks for the name of the file to direct output to (selecting this item with the left button will direct output to `T`, the process TTY display stream), before running the test suites. All output directed to `NIL`, the default output stream, will go to this file, including all error messages generated by the automated test harness and by `TEST-MESSAGE` (see below). It is assumed that no other activity is being performed while testing is in progress.

ABORT Aborts any tests in progress. Confirmation (via clicking the left mouse button) is required. New tests can be selected, tests can be re-run, etc. after an abort.

PAUSE Temporarily pauses any tests in progress. Any pause time does not count in the computation of timeouts (see below).

RESUME Resumes `PAUSED` testing.

DIRECTORY Does a directory of files (the directory pattern is prompted for in the prompt window) and puts them in the test files window in order to have a new set of test suites to select from.

PRINT Prints the results of testing the selected files. Selecting this item with the left button will print on the default printer. Selecting this item with the middle button will put up a menu asking whether to print to a printer or a file. If a printer is selected, then a menu asking for the printer to print to (gotten from `DEFAULTPRINTINGHOST` plus the selection "Other"; the latter will ask for the name of a new printer to print to) is up. Otherwise, if a file is selected, then the user will be prompted for the name of a file to print to (also, if the type of output is not obvious, i.e. `PRESS` or `INTERPRESS`, then the user will be prompted for the type of output). When the `Hardcopy` item of the right button menu is selected for this window, then this command is performed (except that selecting the main item does the default, while selecting either the printer or the file sub-item starts the sequence of questions at the intuitive place).

SUMMARIZE Similar to `PRINT`, except that it prints only those tests (out of the selected tests) which failed.

c. **QUIT** Quits testing, closing the window and throwing away all test results, test names, et stored in the window. If any tests are currently in progress, then confirmation (via clicking the left mouse button) is required in order to quit (in this case an ABORT is performed before quitting). When the tester window is closed, this command is performed.

3. A status window, which has the following fields:

Suite The name of the test suite currently running.
ID The ID of the current test being performed by SINGLE-TEST.
Start The time that the current test was started.
End The time that the current test will time out at, or blank if none.

4. A summary window, which has the following fields:

Files The number of files in the test files window.
Selected The number of files (test suites) selected in the test files window.
Completed The number of test suites completed.
Successful The number of test suites which were successful.

5. The directory pattern used to select the test suite files. Unless otherwise overridden, the directory pattern by default only selects the latest version of each test suite file. Also, unless otherwise overridden, the directory pattern by default only selects .LCOM files (if a source file is more recent than the corresponding compiled file, then an error message is displayed).

6. A heading line which identifies each column in the test files window.

7. The test files window which has a line for each test suite file which matches the directory pattern. The left button on an entry selects only that entry. The middle button on an unselected entry adds that entry to the selected entries. The middle button on a selected entry removes that entry from the selected entries. The right button in the left portion of an entry will extend the current entries to include this entry and all the entries inbetween (the mouse cursor will change to a right pointing arrow when this action is enabled). This window is also scrollable (both vertically and horizontally). When each test is completed, a line is drawn through the entry. This window has the following columns:

Result: The result of testing using the corresponding test file. The following can appear in this column:

? The test suite has not been completed or possibly even initiated, so no results are known.

pass The test suite completed successfully.

FAIL The test suite did not complete successfully. This could be because a test in the test suite returned bad results, a test in the test suite aborted, a test in the test suite ti out, etc..

med

Name: The NAME portion of the test suite file name.

File: The full name of the test suite file (except for the host name).

When the tester is loaded, a new entry is added to the background menu, labelled AutomatedTester. When this is selected, an automated tester process is started, which will prompt (in the system prompt window) for a directory pattern which is used to initialize the test files window.

The individual test handler is a function which is called by the top-level function of each test suite (the function which was called by the top-level tester). This function has the following interface (all arguments must be supplied):

Name: SINGLE-TEST (LAMBDA function).

Arguments:

IDENTIFIER The integer identifier of this test. Identifiers are assigned manually and are unique across all tests in all test suites. [We need to set up an index file for this purpose, in the standard test directory.]

ht EXPRESSION The expression to evaluate (e.g. (PLUS 2 3)). Note that in order to get the right results, this argument would normally be quoted with QUOTE (or ') or be an expression such as (QUOTE (fn)), where fn is a separately defined function (and is therefore compiled code, instead of interpreted code).

5)) PREDICATE The (one argument) predicate to check the result (e.g. (LAMBDA (X) (EQP X or NULL)). This must be NIL iff the result was not correct (non-NIL indicates that the result was correct). If more than one error can occur, then output identifying the specific error should be printed (to NIL). Note that this argument would normally be quoted with QUOTE (or ') or FUNCTION in order to get the right results.

he TIMEOUT The maximum elapsed (wall) time (in milliseconds) that the expression EXPRESSION should take to complete (NIL implies that no timeout is to be used). With the current Interlisp-D process mechanism, this will only work if the expression (or anything it calls) does a BLOCK, so that another process can check to see whether a timeout has occurred. Also, the timing is not exact, so the actual timeout used will be no less than the value supplied. Time elapsed while the test was PAUSED is not counted in checking for a timeout.

Result: NIL iff the test was not successful (due to PREDICATE returning NIL, a NOBIND being returned, a timeout occurring, or a deep exit (such as an abort) occurring). Non-NIL indicates success.

Description: This function evaluates the expression EXPRESSION and checks the result with the predicate PREDICATE, returning the result from calling PREDICATE. If NOBIND is returned from either EXPRESSION or PREDICATE, then an error message is printed (to NIL) and a NIL is returned from SINGLE-TEST. If the timeout is exceeded (and timeouts can be checked) then the evaluation of the expression is aborted and an error message is printed (to NIL) and a NIL is returned from SINGLE-TEST. If a deep exit occurred in either EXPRESSION or PREDICATE (e.g. from aborting of the expression), then an error message is printed (to NIL) and a NIL is returned from SINGLE-TEST.

Side Effects: A message can be printed (to NIL).

Assumptions: Deep exits completely out of EXPRESSION or PREDICATE are not part of the successful behaviour of either EXPRESSION or PREDICATE (any such exits must be caught internally within EXPRESSION or PREDICATE). Note that deep exits are caught via ERRORSET, so RETFROM, RETTO, RETEVAL, RESUME, etc. are not caught.

There is a function available which prints out an easily identifiable error message in a standard format to the standard output. This function has the following interface (all arguments must be supplied):

Name: TEST-MESSAGE (LAMBDA function).

Arguments:

IDENTIFIER The integer identifier of this test (as given to SINGLE-TEST).

TEXT The text of the error message.

INFO Information specific to this instance of this error.

Result: Not useful.

Description: The error message along with the test identifier and the specific information is printed to NIL in a standard, easy to notice format.

Side Effects: A message is printed (to NIL).

Assumptions: None.

Some side-effects of the automated test harness are:

1. The History List for the Programmer's Assistant is used, therefore old items are lost and a REDO, etc. immediately after testing will redo the last command that the automated test harness performed, not the last item printed in the top level typescript window.
2. The top level value and the value in the Programmer's Assistant of `HELPFLAG` are change for the duration of running a test suite.
3. Extra processes are run to perform the testing.

Known deficiencies with the implementation are:

1. ABORTing and PAUSEing can only be done between individual tests.
2. If a test is aborted between individual tests, but not between tests suites, then the effects LOADing and running that test suite are not UNDOne.
3. Some errors are not caught, and some side effects are not undone if errors occur.

Some possible extensions to this package are:

1. Utilities to help with testing for deliberate errors.
2. Utilities to help with automating input which would normally be manual.

Running DSKTEST
The Disk-file-system test utility

1. Load the file DSKTEST.DCOM from whichever directory & server it is stored on.
2. Type
(DSKTEST ' {DSK} <LISPFILES>

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.

THE INTERLISP-D TESTING SYSTEM

The Interlisp-D testing system is an integrated system built for creating, managing and using a large set of programmed tests for testing the correctness and the performance of the Interlisp-D programming environment.

The system is consisted of three parts : The test driver, the data base management system, and a graphic control tool. In addition, there are various tools for helping the test builders in the process of creating new tests.

All parts of the system assumes the structure of a TEST which is a data type consists of several fields, of which the most important are the expression which has to be evaluated, and a predicate which takes the results of this evaluation and determines whether the test was a success or a failure (i.e. whether the actual result is the same as the expected result).

The test driver is in principal a function which gets an object of type TEST, performs the test, and return either success or failure plus some additional information. It includes facilities for monitoring the test execution, tracing and recording the testing process to enable reproducing tests, Remote Eval protocols to enable performing tests with two machines and more.

The data base management system works in two levels. In the low level, the "test cluster" level, the system manages and organizes the tests in the file system, enable retrieving tests through a caching system, and allows concurrent access to test files using a simple locking scheme.

In the high level, the system enables each user to manipulate the database using its own VIEW of the system. This view is implemented through the CONCEPT SPACE which is a directed acyclic graph that will usually reflect the logic structure of the system as seen by the user.

The graphic control tool displays a concept space as a graph and allows the user to perform most of the Test system operations by selecting nodes from the graph.

The tools for building and manipulating tests include test inspector (and editor), a random generator which can generate random specified Lisp objects, Indirect reference to other tests in TEST fields for shrinking the space of the tests themselves and avoiding redundant work when creating tests which share some of their fields, and more .

In the next sections the different parts of the system will be described as well as the interaction between them.

The TEST data type

The TEST is the data type of test objects. Its structure reflects the various properties that tests have. It includes the following fields:

TestID : The tests are identified by an integer.

Input : This field contains an expression that, when evaluated, will generate the list of arguments on which the tested expression will be applied. There are several tools which help in creating this entry. The random generator helps in generating random objects with specified restrictions. The SYSTEMATIC operation helps in generating systematically all the combinations over finite ranges.

Expression: It can contain a function name, a lambda expression or arbitrary sexpression. In the first two cases it will be applied on the input.

Success Predicate: This field contains a lambda expression with two arguments - the ACTUAL input for the test, and the result of the evaluation. It returns one of the two atoms: Success or Failure. When performing tests with random input, some tricks may have to be used as demonstrated in the examples in the end.

Timeout :This is a lambda expression which gets the ACTUAL input as an argument, and produces an upper limit to the estimated time of evaluation.

EvalBefore and EvalAfter: expressions to be evaluated before and after the test execution. usually, before the test we may want to set the appropriate environment for the test (like loading certain files), and after the test we may want to clean up the environment (like deleting files which the test created).

Pretests: Contains a list of links to other tests. These links may influence the order of an execution of a set of tests. Currently there are two types of links. A STRONG link to other test means that whenever the current test is going to be executed, the pretest must be executed first. An example for such pretest may be tests for the tests themselves. If a test generates a few thousands combinations of some arguments, it may be useful to test first if the test itself works correctly by executing a simplified test which works only on one set of arguments, and check that the test outcome is reasonable. A WEAK link to other test means that whenever a SET of tests is being executed, and both the test and pretest are in this set, the pretest will be executed before the test (thus it defines a partial order on any set of tests). This link may be used in cases where there is logical order on the execution of tests - for example, it is reasonable to test opening a file before testing writing to a file.

The Test Driver

The test driver accept a test as its input and returns either success or failure. It will evaluate the input and the tested expression itself on a remote machine if requested, or on the local machine otherwise. All the process of the testing is recorded on a trace file, such that as much information as possible will be available if needed.

The driver evaluates the EVALBEFORE form, evaluates the input expression to generate the input for the tested expression, applies the tested expression on the generated input, applies the success predicate on the result and the generated input, and evaluates the EVALAFTER form. After some of the above stages the appropriate information is written on the trace file. The most important one is the input generated, especially in cases of random input.

All the evaluation done by the driver uses the Interlisp-D ERRORSET command, thus allowing evaluation that will not break under error condition. The error type may be used by the success predicate to determine if the result is a success or a failure. Thus one test for many arithmetic functions can be to supply them with non numeric arguments and to check that the error reported will be the right one.

The evaluation of the tested expression is done as a separated process, such that the driver will be able to try to interrupt it in case where the time of execution is larger than the value of TIMEOUT field of the test. This interrupt will work only if the test execution process will release voluntarily the cpu (when waiting on I/O for example) since Interlisp-D uses non preemptive scheme for process scheduling.

Remote evaluation will not benefit us much in this type of problems. If the remote machine is in infinite loop for example, it will not listen to interrupt attempts as well. The advantages of using remote machine are two: If a long sequence of tests are executed, and the machine "freezes", a remote test will freeze the remote machine and the local machine will be able to call for help and resume operation (as soon as the remote machine does not respond for more than some estimated limit of time, the local machine sends messages to a preset distribution list and asking for human help). A second benefit of remote evaluation is when we need to evaluate the tested expression in a different environment than the Testing system resides. We will want the testing system to work in considerably different environment (software release), while we are testing an experimental different environment.

The data base management system: the "test cluster" level.

The user can retrieve a test by calling the GetTest function. The low level of the dbms is responsible for performing the appropriate operations to retrieve the requested test. If the test is not already loaded it will be loaded from its file. There is a limit on the number of the tests that are loaded, and if this number is exceeded a replacement will take place and a test will be removed. The replacement policy is LRU (least recently used) and is implemented by moving each test being referenced to the front of the list of the loaded tests. Thus the last test in the list will be the one to be removed. The limit on the number of loaded tests is dynamically modified according to the amount of the available memory.

The Interlisp-D testing system is designed to work with several users who use it concurrently. There are no problems if the users were only retrieving tests from the data base. Problems may occur if two users modify the same part of the data base in the same time.

For such cases a locking scheme was integrated into the system. There is a special designated file which is the "gate" for the data base. Users can obtain write LOCKS on tests. The file contains the list of users with their locked tests. The basic locking function is ObtainDatabaseWriteLock(testnumber) which checks the LOCK file, and registers the tests that are not already locked. The user has the option of

automatically generated messages that will be sent to the locking users, inform them that somebody is waiting for their locked tests and request them to release them as soon as they are not needed.

Thus, either automatically or manually, whenever a user edits a test, he will first obtain LOCK on tests. The locking scheme will work only if the users will follow the rules and will not try to access tests not through the testing system.

The basic operation - ObtainDatabaseWriteLock is "atomic" in the sense that the LOCK file is opened for read and write throughout the execution of this procedure, and thus no other user will be able to open it. The time interval in which the file is opened is very short.

Another problem that may arise from concurrent access to the data base is test numbering. As mentioned above, each test has a unique integer as an ID. Thus there is a file which contains the last ID issued, and the procedures for creating new tests will access and update this file.

The "Concept Space" level.

What is the "thing" which is being tested by the test? it may be a specific low level system function, a library package, or a new representation scheme for integers. It is hard to find a common class to which all these entities belong. Thus the testing system assumes that it is some CONCEPT of the Interlisp-D system that is being tested.

While it is true that when a test is CREATED, its creator intent to test a specific concept, the test itself is not necessarily a test only for this concept. A test that was built for testing the READ function, may actually test also the NS communication protocols, the OPENFILE function etc.

For this reason the tested concept is not considered to be a part of the test itself. There is a separated knowledge space which is the way that the user views the test cluster. A concept space is an acyclic directed graph of CONCEPTS. Each node of the graph is of type CONCEPT which has four fields: The concept name, the tests that tests this concept, the subconcepts and the superconcepts.

The main purpose of the concept space is to enable the user to group tests in a logical way and to perform operations on these sets of tests. The semantic of a concept node is : "the tests which tests this concept are the tests of the concept itself plus the tests that tests its subconcepts (recursively)".

Such a definition allows us to build concept spaces which view the tests from different points of view. We may have a concept named "Arithmetic system" with subconcepts "Integer arithmetic", "Float arithmetic" and "Arithmetic functions". The "Arithmetic functions" will have as subconcepts, the concepts "IPLUS", "FPLUS", "PLUS" etc. "IPLUS" is also a subconcept of "Integer arithmetic", and "FPLUS" of "Float arithmetic". Thus, if a new representation for the integers was introduced to the system, we will test the "Integer Arithmetic" concept, while in other cases we may want to test the "Arithmetic Functions".

From the above example and from the more detailed example at the end, we can conclude that the organization of the test system should be very flexible, since there can be many parallel views into the same part of the system. We can also see why a tree structure would not be sufficient as a representation scheme. The Testing system supports the co-existence of several concept spaces, and thus each user can build and use his own concept space(s) to reflect his view of the system.

THE CONCEPT SPACE BROWSER

Most of the operations of the Interlisp-D testing system are done through the "Concept Space Browser". The browser is a graphic tool which is applied on a concept space.

It has a few types of operations. Any operation that require a concept as an argument will get it by a selection from the displayed graph.

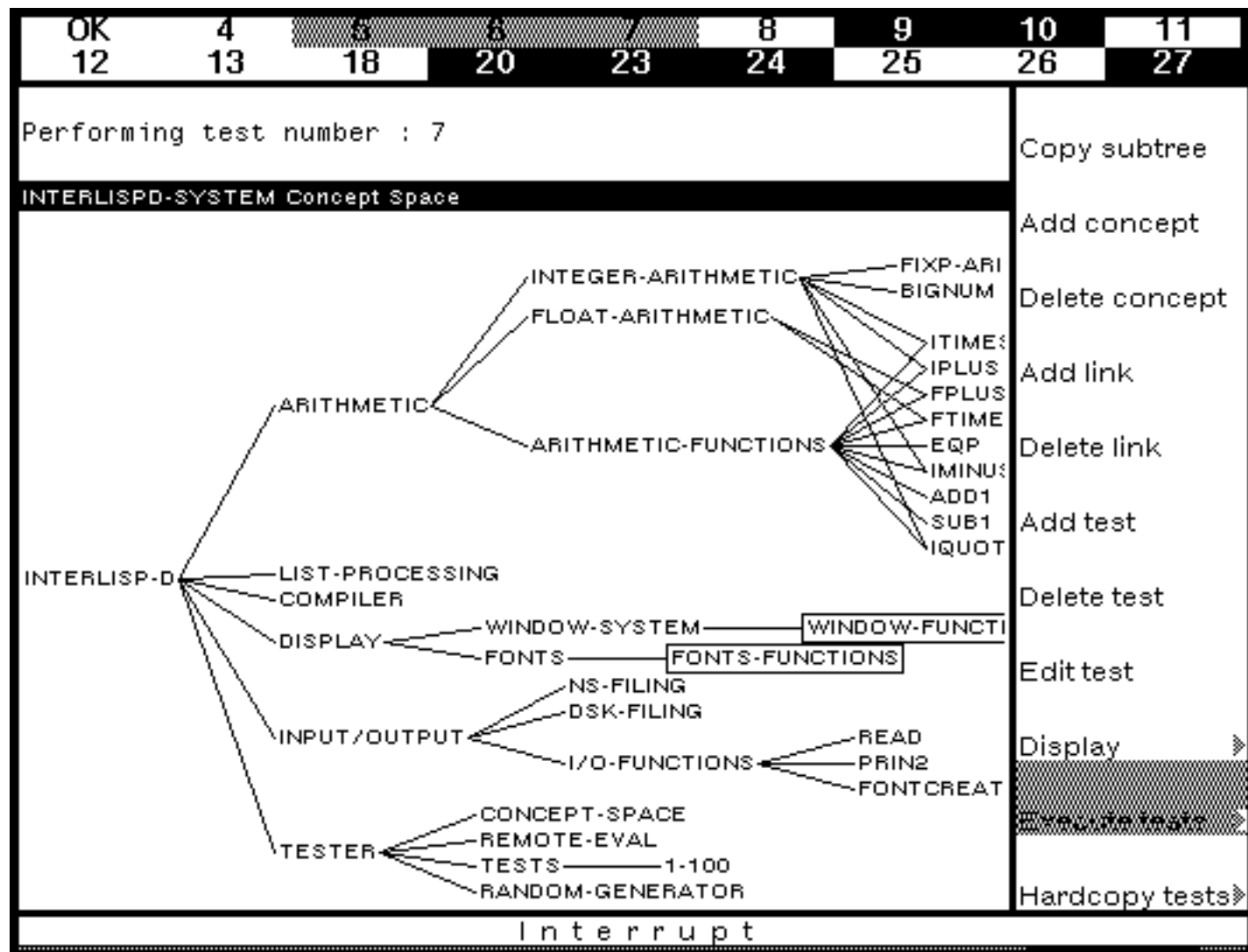
The first type of operations are operations for modifications of the concept space itself. There are commands to add new concept, to delete a concept, to add and delete a link and to add and delete a test to a concept.

Second type of operations are data base operations. The user can edit a test selected from specific concept, can hardcopy all (or part of) the tests of a selected concept, and can request to lock all (or part of) the tests of a concept.

A third type are display op can be specified, a browser of a subgraph can be created, and the tests can be dre are commands to execute all (or part of) the tests of a certain concept, with different modes of execution .

The browser also allows to copy subgraphs between two displayed concept spaces, and to get all the tests of a node by a copy selection so that functions that are not available in the browser can use the concept space as well.

.



Test creation tools and misc tools

There are several tools that were created to help the tests builders in their task.

In each field of a test the user can write (& n) when n is a test number. This tells the system that to retrieve the value for this field it should refer to the same field in test n. This was done since many tests share values for some of their fields.

The test inspector is built on the top of the Interlisp-D inspector. The user can inspect and modify the various fields of a test. In addition he can call the inspector on indirect referenced tests.

There are functions which keep tracks of changes done to tests, and functions which stores modified tests.

The Random Generator is a very important tool for creating a test. It has many entries for different Lisp objects which he can generate randomly. These set of possible entries will grow constantly as test builders will need more types of random objects.

The random generator function gets as an argument an object type and a list of modifiers. The test builder can specify in the input field of a test a call like `(GenerateRandom 'LARGE-INTEGER)`, or `(GenerateRandom 'WINDOW)` or `(GenerateRandom '(LIST-OF-ITEMS WINDOW 50 100))` to get a random list of length between 50 and 100 of windows.

From the experiments done with the Testing system it was clear that random tests are an important part of any testing and can discover bugs that would be hard to find otherwise.

Another tool is the SYSTEMATIC input generator. Many times we want to test a function if it works right with all the possible combinations of values of its arguments, or to find out whether an library package works with all possible settings for its flags. For such cases the test builder can specify in the input field that he wants a systematic test, and supply the expressions that produces the ranges of values to combine.

TEST EXAMPLES:

EXAMPLE 1

```

Test number 18
PRETESTS :NIL
TESTCOMMENT :(* * Generates systematically all the pairs of all
               the "special" bignums
               (stored in TEST.BIGNUM-SPECIAL-NUMBERS in file
                TEST-ARITHMETIC-UTILS)
               apply IPLUS on them and the IDIFFERENCE
               and compare the results)
EVALAFTER :[LAMBDA (RES ARGS)
EVALBEFORE :(& 4)
TIMEOUT :NIL
TIMES :1
SUCCESSPREDICATE :[LAMBDA (RES ARGS)
                    (if (EQP (IDIFFERENCE RES (CADR ARGS))
                        (CAR ARGS))
                        then (QUOTE SUCCESS)
                        else (QUOTE FAILURE)]
INPUT : (SYSTEMATIC TEST.BIGNUM-SPECIAL-NUMBERS
        TEST.BIGNUM-SPECIAL-NUMBERS)
EVALEXPR :IPLUS
TESTID :18

```

EXAMPLE 2

Test number 14**PRETESTS** :((WEAK 10))**TESTCOMMENT** :(* * Creates a random window, reshapes it to a sequence of random regions, **and** backwards to the original shape ; Checks if the screen bitmaps at the begining **and** end of operations are equal)**EVALAFTER** :[LAMBDA (RES ARGS)
 (CLOSEW (CAR ARGS))]**EVALBEFORE** :NIL**TIMEOUT** :[LAMBDA (ARGS)
 1000]**TIMES** :1**SUCCESSPREDICATE** :[LAMBDA (RES ARGS)
 (if (TEST.COMPARE-BITMAPS RES (CADR ARGS))
 then (QUOTE SUCCESS)
 else (QUOTE FAILURE))]**INPUT** :[LIST (TEST.GENERATE-RANDOM (QUOTE WINDOW))
 (BITMAPCOPY (SCREENBITMAP))
 (TEST.GENERATE-RANDOM (QUOTE (LIST-OF-ITEMS REGION
 10 50)))]**EVALEXPR** :[LAMBDA (WINDOW OLDScreen REGION-LIST)
 (PROG (OLD-WINDOW-REGION)
 (SETQ OLD-WINDOW-REGION (WINDOWPROP
 WINDOW
 (QUOTE REGION)))
 (**for** R **in** REGION-LIST
 do (SHAPEW WINDOW R))
 (**for** R **in** (APPEND (REVERSE REGION-LIST)
 (LIST OLD-WINDOW-REGION))
 do (SHAPEW WINDOW R))
 (RETURN (BITMAPCOPY (SCREENBITMAP)))]**TESTID** :14

NTI

EXAMPLE 3

```

Test number 4
PRETESTS :((STRONG 11 23)
              (WEAK 7))
TESTCOMMENT :(* * Generates 1-30 BIGNUMS of the form 100...0,
                  applies ITIMES on them,
                  and checks that the result is of the form
                  100....00 with the right number of ZER0s)
EVALAFTER :NIL
EVALBEFORE : (LOAD? (QUOTE TEST-ARITHMETIC-UTILS))
TIMEOUT : [LAMBDA (ARGS)
             (IPLUS 10000 (ITIMES (LENGTH ARGS)
                                   10000))]
TIMES :10
SUCCESSPREDICATE : [LAMBDA (RES ARGS)
                       (PROG (SUM-OF-LENGTH UNPACKED-RESULT)
                             [SETQ SUM-OF-LENGTH
                               (for ARG in ARGS
                                sum (SUB1 (LENGTH (UNPACK ARG))
                                           (SETQ UNPACKED-RESULT (UNPACK RES)))
                                (if
                                 [AND (EQP (ADD1 SUM-OF-LENGTH)
                                             (LENGTH UNPACKED-RESULT))
                                      (EQ (CAR UNPACKED-RESULT)
                                           (QUOTE 1))]
                                 (for DIGIT
                                  in (CDR UNPACKED-RESULT)
                                  always (EQ DIGIT
                                             (QUOTE 0))
                                 then (RETURN (QUOTE SUCCESS))
                                 else (RETURN (QUOTE FAILURE))
                                (TEST.GENERATE-RANDOM (QUOTE (LIST-OF-ITEMS
                                                             POSITIVE-POWEROF10-BIGNUM 1
                                                             30))))
                               (QUOTE 0))]
                             (QUOTE 0))]
INPUT : (TEST.GENERATE-RANDOM (QUOTE (LIST-OF-ITEMS
                                          POSITIVE-POWEROF10-BIGNUM 1
                                          30)))
EVALEXPR :ITIMES
TESTID :4

```


EXAMPLE 4

```

Test number 16
PRETESTS :NIL
TESTCOMMENT :(* * This test create systematically all
               combinations of arguments to FONTCREATE
               function,
               and tests whether the only error is
               "file not found")
EVALAFTER :[LAMBDA (RES ARGS)
            (if (NOT (TEST.ERRORP RES))
                then (APPLY (QUOTE SETFONTDESCRIPTOR)
                           ARGS)]
EVALBEFORE :NIL
TIMEOUT :NIL
TIMES :1
SUCCESSPREDICATE :[LAMBDA (RES ARGS)
                   (if (OR (NOT (TEST.ERRORP RES))
                           (EQP (CADR RES)
                                17))
                       then (QUOTE SUCCESS)
                       else (QUOTE FAILURE)]
INPUT : (SYSTEMATIC (QUOTE (GACHA NIL))
                (QUOTE (-1 12 NIL))
                [TEST.ALL-COMBINATIONS
                 (QUOTE ((BOLD MEDIUM LIGHT)
                         (ITALIC REGULAR)
                         (REGULAR COMPRESSED EXPANDED))
                 (QUOTE (0 90 NIL))
                 (QUOTE (DISPLAY PRESS NIL))))
EVALEXPR :FONTCREATE
TESTID :16

```

TEST APPRENTICE

This is the preliminary documentation for the first experimental version of the Test Apprentice. The purpose of this tool is to help with testing. It is eventually intended to generate and execute tests (it would be an AI application). In its current state it is just learning by watching what other testers do. But it is useful in this state because it can repeat exactly what other testers have done before.

The Test Apprentice is easy to use, you just type in the commands that you would use for doing the test the first time, with only an occasional extra command. To repeat a test, you only need to specify a special Test Apprentice function giving it the name of the test to re-perform.

The Test Apprentice groups test steps (commands) into tests, and groups tests into test suites. A test is a group of highly related test steps which are used to test a (small) portion of the system and which can be run independently from other tests (e.g. NS Filing directory enumeration). A test suite is a group of related tests which can be run together for convenience (e.g. NS Filing).

The Test Apprentice has the following limitations:

1. It can only record and repeat commands to the (standard) Lisp Executive, it does not record mouse operations or commands typed into other windows. It also cannot in general record input which is non-commands typed to the Lisp Executive (e.g. responses to the FILES? function's questions), but some commands record responses on the history list, so they are recorded (e.g. responses to the COMPILE function's questions). The rule of thumb to use here is if a REDO would re-perform those inputs, then the Test Apprentice has recorded them.
2. The number of test steps recorded for a particular test is limited to the size of the history list (usually 100 entries).

The following are the commands and functions available in the Test Apprentice:

| | |
|-----|--|
| ST | Start Test. A Lisp Executive command which starts the Test Apprentice recording the test steps of a test. An implicit ST is performed after an ET command or an ADD-TO-TEST-SUITE function. An ST can be performed at any time to restart recording of a test (and forget any recording in progress). |
| ET | End Test. A Lisp Executive command which completes the recording of the test steps of a test. This must have the name of the test as its single parameter (e.g. "ET NSFiling-1"). Test names only have to be unique within a test suite. |
| ITS | Ignore Test Step. A Lisp Executive command which ignores test steps in the current test. Currently this will only ignore the previous step, but in the future it will ignore specified steps. This is useful for removing mistakes from a test or removing miscellaneous commands not directly related to the tests. If you need to delete anything other than the last test step, then you must edit the test suite manually or restart the test using ST (the former can only be done after an ET, while the latter can only be done before an ET). |
| ITR | Ignore Test Result. A Lisp Executive command which ignores test results from test steps in the current test. Currently this will only ignore results from the previous step, but in the future it will ignore results from specified steps. This is useful for ignoring intermediate results. The Test Apprentice will check to see if the results from re-running tests are EQUAL to the results obtained when the test was recorded. For example, one test step may have returned a window as a result, then the next merely see if the type of the result |

is WINDOW. In this case two different instances of a window will not be EQUAL, so the result of the first step should be ignored. If this becomes a problem EQUALALL may be used in a future version of the Test Apprentice, but even this does not always work (e.g. it does not work on windows).

ADD-TO-TEST-SUITE An NLAMBDA function which adds a test to a test suite. Its one required argument is the name of the test suite. If the test suite did not exist before, then it is created. Test suites are just variables that contain a list in a specified format, they can be saved in files by the File Package command VARS (or UGLYVARS if data structures are used in test results or HORRIBLEVARS if there are circular pointers). This function must be called before any test steps are started. It sets up all future tests to be part of this test suite until changed by another ADD-TO-TEST-SUITE.

EXECUTE-TEST-SUITE A LAMBDA function which executes all tests in a test suite. Its one required argument is the test suite to execute. This returns a true (non-NIL) value iff all tests executed successfully. An error message is printed out for each test which fails.

EXECUTE-TEST A LAMBDA function which executes one specific test in a test suite. Its required arguments are the test suite and the name of the test in the test suite to execute. This returns a true (non-NIL) value iff the test executed successfully.

The following are recommendations on using the Test Apprentice effectively:

1. Don't use absolute command number references, use relative ones (e.g. use "(VALUEOF -2)" not "(VALUEOF 36)" if you are on command number 38). This is so the test will run correctly when it is re-run (if it were re-run and the VALUEOF was done on command number 67, then you would get some unexpected command's result instead of what you wanted).
2. Two different data structures (defined with DATATYPE, ARRAY, etc., such as windows) will never compare EQUAL, even if they contain the same values. This means these should never be used as (non-ignored) test step results. Use ways of looking at the contents of the data structures instead.

The following is an example of a normal session with the Test Apprentice for recording a test suite:

```

34_(ADD-TO-TEST-SUITE NSFiling)
NSFiling
35_ST
Start-of-test-block
36_<test 1 step 1>
<test 1 step 1 result>
37_<test 1 step 2>
<test 1 step 2 result>
...
40_ET NSFiling-1
End-of-test-block
41_ST
Start-of-test-block
42_<test 2 step 1>
<test 2 step 1 result>
43_<test 2 step 2>
<test 2 step 2 result>
...
65_ET NSFiling-2

```

```

End-of-test-block
66_(FILES?)
the variables: NSFiling...to be dumped.
want to say where the above go ? Yes
(variables)
NSFiling  File name: NSFiling
NIL
66_(MAKEFILE '{Erinyes}<Test>Tests>OS>NSFiling)
{Erinyes}<Test>Tests>OS>NSFiling.;1

```

The following is an example of a normal session with the Test Apprentice for repeating a test suite:

```

19_(EXECUTE-TEST-SUITE NSFiling)
Executing NSFiling-1
Executing NSFiling-2
T

```

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 variable 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 closed. 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 than one (or the `times` field of the test is greater than one). The other way is when the input expression is a list starts with the atom `SYSTEMATIC`. In such a case each of the elements of the CDR of the list will be evaluated. The tester expects 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 a record with the fields `CONCEPTSPACENAME`, `ROOTCONCEPT` and `CONCEPTLIST`. `ROOTCONCEPT` is the name of the root concept. `CONCEPTLIST` 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 CONCEPTSPACE-NAME)

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 than one. Such tests will be executed more than one time. There are also tests which have as their INPUT field a list that 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 case 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 occurred 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 EVAEXPR 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 encounters an INPUT field that starts with the atom SYSTEMATIC. It is useful in any case that the user wants to build tests that try 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 evaluation will not break even if an error condition occurs. If an error did not occur the function will return the result of the evaluation of form. If an 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 considered as returning with error condition. The function create a separate 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 scheduling algorithms are non-preemptive, the function is not guaranteed 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 between 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 locks to the tests in TEST-LIST that are locked by the user. Returns all the tests that it succeeded 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 the tests 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 with 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 necessary.

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 link 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 apply 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 redisplay 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 selected 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

Test Utilities

The following are a useful set of utilities for the testing group. They all reside in the TestUtils file on the Tools subdirectory of the standard test directory.

PRINT-TEST-ARS A lambda function which prints out the AR number and subject fields from an AR query. This function has a window and a file as arguments. The window is the main window of an AR query form (labelled "AR Query Window"). The file is the file to print output on (NIL goes to standard output). The query must already have been performed to select those items which are to be printed and the query should have been sorted by status (as the first sort item in the sort list). In order for this to work correctly, you must scroll the "AR Query Browser" window through all the items before using this function. As an example of use, say that you had previously specified a query list of "(System: IS Operating% System) (Subsystem: IS Generic% File% Operations)" and a sort list of "Status:" in an AR query window (and performed the query and scrolled through all the items), then you could point the cursor to the window labelled "AR Query Window" and do "(PRINT-TEST-ARS (WHICHW) '{DSK}<LISPFILES>TEMP)". This would print a list of the ARs in the file {DSK}<LISPFILES>TEMP.