
LispWorks® Reference Manual

Version 5.1



Copyright and Trademarks

LispWorks Reference Manual

Version 5.1

March 2008

Copyright © 2008 by LispWorks Ltd.

All Rights Reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of LispWorks Ltd.

The information in this publication is provided for information only, is subject to change without notice, and should not be construed as a commitment by LispWorks Ltd. LispWorks Ltd assumes no responsibility or liability for any errors or inaccuracies that may appear in this publication. The software described in this book is furnished under license and may only be used or copied in accordance with the terms of that license.

LispWorks and KnowledgeWorks are registered trademarks of LispWorks Ltd.

Adobe and PostScript are registered trademarks of Adobe Systems Incorporated. Other brand or product names are the registered trademarks or trademarks of their respective holders.

The code for walker.lisp and compute-combination-points is excerpted with permission from PCL, Copyright © 1985, 1986, 1987, 1988 Xerox Corporation.

The XP Pretty Printer bears the following copyright notice, which applies to the parts of LispWorks derived therefrom:

Copyright © 1989 by the Massachusetts Institute of Technology, Cambridge, Massachusetts.

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that this copyright and permission notice appear in all copies and supporting documentation, and that the name of M.I.T. not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. M.I.T. makes no representation about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty. M.I.T. disclaims all warranties with regard to this software, including all implied warranties of merchantability and fitness. In no event shall M.I.T. be liable for any special, indirect or consequential damages or any damages whatsoever resulting from loss of use, data or profits, whether in an action of contract, negligence or other tortious action, arising out of or in connection with the use or performance of this software.

LispWorks contains part of ICU software obtained from <http://source.icu-project.org> and which bears the following copyright and permission notice:

ICU License - ICU 1.8.1 and later

COPYRIGHT AND PERMISSION NOTICE

Copyright © 1995-2006 International Business Machines Corporation and others. All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE BE LIABLE FOR ANY CLAIM, OR ANY SPECIAL INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization of the copyright holder. All trademarks and registered trademarks mentioned herein are the property of their respective owners.

US Government Restricted Rights

The LispWorks Software is a commercial computer software program developed at private expense and is provided with restricted rights. The LispWorks Software may not be used, reproduced, or disclosed by the Government except as set forth in the accompanying End User License Agreement and as provided in DFARS 227.7202-1(a), 227.7202-3(a) (1995), FAR 12.212(a)(1995), FAR 52.227-19, and/or FAR 52.227-14 Alt III, as applicable. Rights reserved under the copyright laws of the United States.

Address

LispWorks Ltd
St. John's Innovation Centre
Cowley Road
Cambridge
CB4 0WS
England

Telephone

From North America: 877 759 8839
(toll-free)
From elsewhere: +44 1223 421860

Fax

From North America: 305 468 5262
From elsewhere: +44 870 2206189

www.lispworks.com

Contents

Preface xxvii

1 The CLOS Package 1

break-new-instances-on-access 1
break-on-access 2
class-extra-initargs 3
compute-class-potential-initargs 4
compute-discriminating-function 6
funcallable-standard-object 6
process-a-class-option 7
process-a-slot-option 9
set-make-instance-argument-checking 11
slot-boundp-using-class 12
slot-makunbound-using-class 13
slot-value-using-class 13
trace-new-instances-on-access 14
trace-on-access 16
unbreak-new-instances-on-access 18
unbreak-on-access 19
untrace-new-instances-on-access 19
untrace-on-access 20

2 The COMM Package 21

attach-ssl 21

destroy-ssl 23
destroy-ssl-ctx 23
detach-ssl 24
do-rand-seed 25
ensure-ssl 25
get-host-entry 26
get-socket-address 28
get-socket-peer-address 29
get-verification-mode 29
ip-address-string 30
make-ssl-ctx 31
open-tcp-stream 32
openssl-version 35
pem-read 36
read-dhparams 37
set-verification-mode 38
set-ssl-ctx-dh 40
set-ssl-ctx-options 42
set-ssl-ctx-password-callback 43
set-ssl-library-path 44
socket-error 45
socket-stream 45
socket-stream-address 50
socket-stream-ctx 50
socket-stream-peer-address 51
socket-stream-ssl 52
ssl-cipher-pointer 52
ssl-cipher-pointer-stack 52
ssl-closed 53
ssl-condition 53
ssl-ctx-pointer 53
ssl-error 54
ssl-failure 54
ssl-new 54
ssl-pointer 55
ssl-x509-lookup 55
start-up-server 56

start-up-server-and-mp 60
string-ip-address 61
with-noticed-socket-stream 62

3 The COMMON-LISP Package 63

apropos 63
apropos-list 64
base-string 65
close 65
coerce 66
compile 67
compile-file 68
concatenate 72
declaim 73
declare 74
defclass 77
defpackage 81
describe 82
directory 83
disassemble 87
documentation 88
double-float 89
features 89
input-stream-p 92
interactive-stream-p 93
load-logical-pathname-translations 94
long-float 94
long-site-name 95
loop 95
make-array 97
make-hash-table 98
make-instance 99
make-sequence 100
map 101
merge 102
open 102
open-stream-p 104

output-stream-p 105
proclaim 106
restart-case 107
room 108
short-float 112
short-site-name 113
simple-base-string 113
single-float 113
software-type 114
software-version 115
step 116
stream-element-type 119
string 120
time 121
trace 122
truename 129
untrace 129
with-output-to-string 131

4 The COMPILER Package 133

deftransform 133

5 The DBG Package 137

debug-print-length 137
debug-print-level 138
hidden-packages 139
print-binding-frames 140
print-catch-frames 143
print-handler-frames 144
print-open-frames 145
print-restart-frames 146
terminal-debugger-block-multiprocessing 147

6 The DSPEC Package 151

active-finders 151
at-location 152

canonicalize-dspec 153
def 154
define-dspec-alias 155
define-dspec-class 156
define-form-parser 159
dspec-class 163
dspec-classes 163
dspec-defined-p 164
dspec-definition-locations 164
dspec-equal 165
dspec-name 166
dspec-primary-name 166
dspec-progenitor 167
dspec-subclass-p 168
dspec-undefiner 168
discard-source-info 169
find-dspec-locations 169
find-name-locations 170
get-form-parser 171
local-dspec-p 172
location 173
name-defined-dspecs 173
name-definition-locations 174
name-only-form-parser 175
parse-form-dspec 176
record-definition 176
record-source-files 177
redefinition-action 178
save-tags-database 179
single-form-form-parser 179
single-form-with-options-form-parser 180
traceable-dspec-p 181
tracing-enabled-p 182
tracing-state 182

7 The EXTERNAL-FORMAT Package 185

char-external-code 185

decode-external-string 186
encode-lisp-string 187
external-format-error 188
external-format-foreign-type 188
external-format-type 189
find-external-char 189
valid-external-format-p 190

8 The HCL Package 193

add-special-free-action 193
add-symbol-profiler 194
allocation-in-gen-num 195
avoid-gc 196
binds-who 197
block-promotion 197
building-universal-intermediate-p 199
calls-who 200
cd 200
change-directory 201
check-fragmentation 202
clean-down 203
clean-generation-0 204
collect-generation-2 205
collect-highest-generation 205
compiler-break-on-error 206
compile-file-if-needed 207
copy-to-weak-simple-vector 208
create-universal-binary 209
current-stack-length 210
default-package-use-list 211
default-profiler-collapse 211
default-profiler-cutoff 212
default-profiler-limit 212
default-profiler-sort 213
delete-advice 213
disable-trace 215
dump-form 215

dump-forms-to-file 216
enlarge-generation 217
enlarge-static 218
expand-generation-1 219
extend-current-stack 220
extended-time 221
file-string 223
file-writable-p 224
find-object-size 224
finish-heavy-allocation 225
flag-not-special-free-action 226
flag-special-free-action 227
gc-generation 227
gc-if-needed 231
get-default-generation 231
get-gc-parameters 232
get-working-directory 233
handle-existing-defpackage 234
handle-old-in-package 235
handle-old-in-package-used-as-make-package 236
load-fasl-or-lisp-file 237
mark-and-sweep 238
max-trace-indent 239
normal-gc 240
packages-for-warn-on-redefinition 241
parse-float 241
print-profile-list 242
profile 246
profiler-threshold 247
profile-symbol-list 248
profiler-tree-from-function 248
profiler-tree-to-function 249
references-who 250
remove-special-free-action 251
remove-symbol-profiler 251
reset-profiler 252
save-argument-real-p 253

save-image 254
save-universal-from-script 261
set-array-weak 263
set-default-generation 264
set-hash-table-weak 265
set-gc-parameters 267
set-minimum-free-space 269
set-process-profiling 270
set-profiler-threshold 272
set-promotion-count 273
set-up-profiler 274
sets-who 277
source-debugging-on-p 278
start-profiling 278
stop-profiling 280
sweep-all-objects 281
switch-static-allocation 282
symbol-alloc-gen-num 282
toggle-source-debugging 283
total-allocation 284
traced-arglist 285
traced-results 286
trace-indent-width 287
trace-level 288
trace-print-circle 289
trace-print-length 289
trace-print-level 290
trace-print-pretty 291
trace-verbose 292
try-compact-in-generation 293
try-move-in-generation 295
who-binds 296
who-calls 297
who-references 297
who-sets 298
with-heavy-allocation 299
with-output-to-fasl-file 300

9 The LINK-LOAD Package 303

break-on-unresolved-functions 303
foreign-symbol-address 304
get-foreign-symbol 305
lisp-name-to-foreign-name 306
read-foreign-modules 307

10 The LISPWORKS Package 309

8-bit-string 309
16-bit-string 310
appendf 310
base-character 310
base-character-p 311
base-char-p 311
base-char-code-limit 312
base-string-p 312
browser-location 313
call-next-advice 313
compile-system 315
concatenate-system 317
current-pathname 318
defadvice 320
default-action-list-sort-time 323
default-character-element-type 324
define-action 324
define-action-list 326
defsystem 328
defsystem-verbose 333
delete-directory 333
deliver 334
describe-length 335
describe-level 335
describe-print-length 337
describe-print-level 337
dll-quit 338
dotted-list-length 340

dotted-list-p 341
do-nothing 341
enter-debugger-directly 342
environment-variable 342
errno-value 344
example-file 344
example-compile-file 345
example-load-binary-file 346
execute-actions 346
extended-char 348
extended-character 348
extended-character-p 349
extended-char-p 349
external-formats 350
false 351
file-directory-p 351
find-regexp-in-string 352
function-lambda-list 354
get-inspector-values 355
get-unix-error 357
grep-command 357
grep-command-format 358
grep-fixed-args 359
handle-existing-action-in-action-list 359
handle-existing-action-list 360
handle-missing-action-list 360
handle-missing-action-in-action-list 361
handle-warn-on-redefinition 361
hardcopy-system 362
init-file-name 363
inspect-through-gui 363
lisp-image-name 364
lispworks-directory 365
load-all-patches 366
load-system 367
make-unregistered-action-list 369
make-mt-random-state 370

mt-random 371
mt-random-state 372
mt-random-state 372
mt-random-state-p 372
pathname-location 373
precompile-regexp 374
print-actions 374
print-action-lists 375
print-command 375
print-nickname 376
prompt 376
quit 377
rebinding 379
regexp-find-symbols 380
remove-advice 381
removef 382
require-verbose 383
round-to-single-precision 384
sbchar 385
set-default-character-element-type 385
simple-base-string-p 386
simple-char 387
simple-char-p 387
simple-text-string 387
simple-text-string-p 388
start-tty-listener 388
stchar 389
string-append 390
text-string 391
text-string-p 391
true 392
undefine-action 392
undefine-action-list 393
user-preference 393
when-let 395
when-let* 396
whitespace-char-p 397

with-action-item-error-handling 398
with-action-list-mapping 399
with-unique-names 400

11 The MP Package 403

change-process-priority 403
create-simple-process 404
current-process 406
debug-other-process 407
default-process-priority 407
default-simple-process-priority 407
ensure-process-cleanup 408
find-process-from-name 409
get-current-process 409
get-process 410
initialize-multiprocessing 411
initial-processes 412
list-all-processes 413
lock-name 414
lock-owner 415
mailbox-empty-p 416
mailbox-peek 417
mailbox-read 417
mailbox-reader-process 418
mailbox-send 419
mailbox-wait-for-event 419
main-process 421
make-lock 421
make-mailbox 423
make-named-timer 424
make-timer 425
map-all-processes 426
map-all-processes-backtrace 426
map-process-backtrace 427
map-processes 428
notice-fd 429
process-alive-p 429

process-allow-scheduling 430
process-arrest-reasons 430
process-break 431
process-continue 431
process-idle-time 432
process-initial-bindings 432
process-interrupt 434
process-kill 435
process-lock 435
process-mailbox 436
process-name 437
process-p 437
process-plist 437
process-priority 438
process-reset 439
process-run-function 439
process-run-reasons 441
process-run-time 441
process-send 442
process-stop 443
process-stopped-p 444
process-unlock 444
process-unstop 445
process-wait 446
process-wait-for-event 447
process-wait-function 448
process-wait-with-timeout 448
process-whostate 449
ps 450
schedule-timer 451
schedule-timer-milliseconds 452
schedule-timer-relative 454
schedule-timer-relative-milliseconds 455
simple-process-p 457
symeval-in-process 457
timer-expired-p 458
timer-name 459

unnotice-fd 460
unschedule-timer 461
wait-processing-events 462
with-lock 463
without-interrupts 464
without-preemption 465
yield 465

12 The PARSERGEN Package 467

defparser 467

13 The SERIAL-PORT Package 469

open-serial-port 469
close-serial-port 471
get-serial-port-state 471
serial-port 472
read-serial-port-char 472
read-serial-port-string 473
serial-port-input-available-p 474
set-serial-port-state 475
wait-serial-port-state 475
write-serial-port-char 476
write-serial-port-string 477

14 The SQL Package 479

add-sql-stream 479
attribute-type 480
cache-table-queries 482
cache-table-queries-default 483
commit 483
connect 484
connect-if-exists 491
connected-databases 491
create-index 492
create-table 493
create-view 494

create-view-from-class 496
database-name 496
default-database 497
default-database-type 497
default-update-objects-max-len 498
def-view-class 498
delete-instance-records 505
delete-records 506
delete-sql-stream 507
disable-sql-reader-syntax 508
disconnect 508
do-query 509
drop-index 511
drop-table 511
drop-view 512
drop-view-from-class 512
enable-sql-reader-syntax 513
execute-command 514
find-database 514
initialize-database-type 515
initialized-database-types 516
insert-records 517
instance-refreshed 518
list-attribute-types 519
list-attributes 520
list-classes 521
list-sql-streams 522
list-tables 523
lob-stream 523
locally-disable-sql-reader-syntax 525
locally-enable-sql-reader-syntax 525
loop 526
map-query 528
mysql-library-directories 529
mysql-library-path 530
ora-lob-append 531
ora-lob-assign 532

ora-lob-char-set-form 532
ora-lob-char-set-id 533
ora-lob-close 534
ora-lob-copy 535
ora-lob-create-empty 536
ora-lob-create-temporary 537
ora-lob-disable-buffering 538
ora-lob-element-type 539
ora-lob-enable-buffering 540
ora-lob-env-handle 540
ora-lob-erase 541
ora-lob-file-close 542
ora-lob-file-close-all 543
ora-lob-file-exists 543
ora-lob-file-get-name 544
ora-lob-file-is-open 545
ora-lob-file-open 546
ora-lob-file-set-name 546
ora-lob-flush-buffer 547
ora-lob-free 548
ora-lob-free-temporary 549
ora-lob-get-buffer 550
ora-lob-get-chunk-size 552
ora-lob-get-length 553
ora-lob-internal-lob-p 554
ora-lob-is-equal 554
ora-lob-is-open 555
ora-lob-is-temporary 555
ora-lob-load-from-file 556
ora-lob-lob-locator 557
ora-lob-locator-is-init 558
ora-lob-open 559
ora-lob-read-buffer 560
ora-lob-read-into-plain-file 562
ora-lob-read-foreign-buffer 563
ora-lob-svc-ctx-handle 564
ora-lob-trim 565

ora-lob-write-buffer 566
ora-lob-write-from-plain-file 567
ora-lob-write-foreign-buffer 568
print-query 570
query 571
reconnect 572
restore-sql-reader-syntax-state 573
rollback 574
select 575
simple-do-query 579
sql 581
sql-connection-error 581
sql-database-data-error 582
sql-database-error 582
sql-expression 583
sql-enlarge-static 584
sql-fatal-error 585
sql-libraries 585
sql-loading-verbose 585
sql-operation 586
sql-operator 588
sql-recording-p 588
sql-stream 589
sql-temporary-error 590
sql-timeout-error 590
sql-user-error 591
standard-db-object 591
start-sql-recording 591
status 592
stop-sql-recording 593
table-exists-p 594
update-instance-from-records 595
update-objects-joins 595
update-records 597
update-records-from-instance 598
update-record-from-slot 599
update-slot-from-record 599

with-transaction 600

15 The STREAM Package 603

buffered-stream 603

fundamental-binary-input-stream 605

fundamental-binary-output-stream 606

fundamental-binary-stream 606

fundamental-character-input-stream 607

fundamental-character-output-stream 607

fundamental-character-stream 608

fundamental-input-stream 609

fundamental-output-stream 610

fundamental-stream 610

stream-advance-to-column 611

stream-check-eof-no-hang 612

stream-clear-input 612

stream-clear-output 613

stream-file-position 613

stream-fill-buffer 614

stream-finish-output 615

stream-flush-buffer 616

stream-force-output 617

stream-fresh-line 617

stream-line-column 618

stream-listen 618

stream-output-width 619

stream-peek-char 620

stream-read-buffer 621

stream-read-byte 622

stream-read-char 622

stream-read-char-no-hang 623

stream-read-line 624

stream-read-sequence 624

stream-read-timeout 625

stream-start-line-p 626

stream-terpri 627

stream-unread-char 627

stream-write-buffer 628
stream-write-byte 629
stream-write-char 629
stream-write-sequence 630
stream-write-string 631
with-stream-input-buffer 632
with-stream-output-buffer 633

16 The SYSTEM Package 637

apply-with-allocation-in-gen-num 637
augmented-string 638
augmented-string-p 639
call-system 639
call-system-showing-output 641
cdr-assoc 644
check-network-server 645
coerce-to-gesture-spec 645
copy-preferences-from-older-version 647
count-gen-num-allocation 649
default-eol-style 650
default-stack-group-list-length 650
define-top-loop-command 651
detect-eol-style 653
detect-japanese-encoding-in-file 654
detect-unicode-bom 655
directory-link-transparency 656
extended-spaces 657
file-encoding-detection-algorithm 657
file-encoding-resolution-error 658
file-eol-style-detection-algorithm 659
filename-pattern-encoding-matches 659
find-encoding-option 660
find-filename-pattern-encoding-match 660
gen-num-segments-fragmentation-state 661
gesture-spec-accelerator-bit 663
gesture-spec-control-bit 663
gesture-spec-data 664

gesture-spec-hyper-bit 664
gesture-spec-meta-bit 665
gesture-spec-modifiers 665
gesture-spec-p 666
gesture-spec-shift-bit 667
gesture-spec-super-bit 667
gesture-spec-to-character 668
get-file-stat 668
get-folder-path 670
get-user-profile-directory 672
guess-external-format 673
in-static-area 674
int32 674
int32* 676
int32+ 677
int32- 677
+int32-0+ 678
+int32-1+ 678
int32-1+ 678
int32-1- 679
int32/ 680
int32/= 680
int32< 681
int32<< 682
int32<= 682
int32= 683
int32> 683
int32>= 684
int32>> 685
int32-aref 685
int32-logand 686
int32-logandc1 687
int32-logandc2 687
int32-logbitp 688
int32-logeqv 689
int32-logior 690
int32-lognand 690

int32-lognor 691
int32-lognot 692
int32-logorc1 692
int32-logorc2 693
int32-logtest 694
int32-logxor 694
int32-minusp 695
int32-plusp 695
int32-to-integer 696
int32-zerop 697
integer-to-int32 697
line-arguments-list 698
load-data-file 699
locale-file-encoding 700
make-gesture-spec 701
make-simple-int32-vector 703
make-stderr-stream 704
make-typed-aref-vector 704
marking-gc 705
memory-growth-margin 707
merge-ef-specs 708
object-address 708
open-pipe 709
open-url 712
pid-exit-status 713
pointer-from-address 714
print-pretty-gesture-spec 715
print-symbols-using-bars 716
product-registry-path 717
room-values 718
run-shell-command 719
safe-locale-file-encoding 724
set-automatic-gc-callback 725
set-blocking-gen-num 726
set-default-segment-size 729
set-delay-promotion 730
set-file-dates 731

set-gen-num-gc-threshold 731
set-maximum-memory 733
set-maximum-segment-size 734
set-memory-check 736
set-memory-exhausted-callback 737
set-signal-handler 738
set-spare-keeping-policy 740
sg-default-size 741
simple-augmented-string 742
simple-augmented-string-p 742
simple-int32-vector 743
stack-overflow-behaviour 743
staticp 744
storage-exhausted 745
sweep-gen-num-objects 746
typed-aref 746
wait-for-input-streams 748
wait-for-input-streams-returning-first 749

17 The WIN32 Package (including DDE) 751

dismiss-splash-screen 751
latin-1-code-pages 752
multibyte-code-page-ef 753
set-application-themed 753
close-registry-key 754
collect-registry-subkeys 755
collect-registry-values 756
create-registry-key 758
delete-registry-key 759
enum-registry-value 760
open-registry-key 762
query-registry-key-info 763
query-registry-value 764
registry-key-exists-p 765
registry-value 766
set-registry-value 767
with-registry-key 769

dde-advise-start 770
dde-advise-start* 772
dde-advise-stop 774
dde-advise-stop* 775
dde-client-advise-data 776
dde-connect 777
dde-disconnect 778
dde-execute 778
dde-execute* 779
dde-execute-command 780
dde-execute-command* 781
dde-execute-string 783
dde-execute-string* 784
dde-item 785
dde-item* 787
dde-poke 789
dde-poke* 790
dde-request 792
dde-request* 794
define-dde-client 796
with-dde-conversation 797
dde-server-poke 798
dde-server-request 799
dde-server-topic 800
dde-server-topics 801
dde-topic-items 802
define-dde-dispatch-topic 802
define-dde-server 804
define-dde-server-function 805
start-dde-server 809

18 Dynamic library C functions 811

InitLispWorks 811
LispWorksDlsym 814
LispWorksState 815
SimpleInitLispWorks 816
QuitLispWorks 817

Index 819

Preface

About this manual

Most of the *LispWorks Reference Manual* is organized by package: each chapter contains reference material for the exported symbols in a given package. The chapters are organized alphabetically by package name. The exception is the last chapter which contains reference material for C functions. Within each chapter, the symbols are organized alphabetically (ignoring non-alphanumeric characters that are common in Lisp symbols, such as *). The chapters are:

- Chapter 1, “The CLOS Package”, describes the LispWorks extensions to CLOS, the Common Lisp Object System.
- Chapter 2, “The COMM Package”, describes the functions providing the TCP/IP interface.
- Chapter 3, “The COMMON-LISP Package”, describes the LispWorks extensions to symbols in the `COMMON-LISP` package. You should refer to the Common Lisp Hyperspec, supplied in HTML format with LispWorks, for full documentation about standard Common Lisp symbols.
- Chapter 4, “The COMPILER Package”, describes symbols available in the `COMPILER` package.
- Chapter 5, “The DBG Package”, describes symbols available in the `DBG` package, used to configure the debugging information produced by LispWorks.

- Chapter 6, “The DSPEC Package”, describes the symbols available in the `DSPEC` package, which are used for naming and locating definitions.
- Chapter 7, “The EXTERNAL-FORMAT Package”, describes symbols available in the `EXTERNAL-FORMAT` package.
- Chapter 8, “The HCL Package”, describes symbols available in the `HCL` package.
- Chapter 9, “The LINK-LOAD Package”, describes symbols available in the `LINK-LOAD` package. It applies to LispWorks for UNIX only (not LispWorks for Linux or LispWorks for FreeBSD).
- Chapter 10, “The LISPWORKS Package”, describes symbols available in the `LISPWORKS` package.
- Chapter 11, “The MP Package”, describes symbols available in the `MP` package, giving you access to the multi-processing capabilities of LispWorks.
- Chapter 12, “The PARSERGEN Package”, describes symbols available in the `PARSERGEN` package, the LispWorks parser generator.
- Chapter 13, “The SERIAL-PORT Package” documents the Serial Port API. This is implemented only in LispWorks for Windows.
- Chapter 14, “The SQL Package” documents symbols used in accessing LispWorks ODBC and SQL functionality.
- Chapter 15, “The STREAM Package” documents the symbols available in the `STREAM` package that provide users with the functionality to define their own streams for use by the standard I/O functions.
- Chapter 16, “The SYSTEM Package”, describes symbols available in the `SYSTEM` package.
- Chapter 17, “The WIN32 Package (including DDE)”, describes symbols available in the `WIN32` package. It applies only to LispWorks for Windows.
- Chapter 18, “Dynamic library C functions”, describes C functions available in LispWorks dynamic libraries.

Many chapters in the *LispWorks Reference Manual* should be used in conjunction with relevant chapters in the *LispWorks User Guide* that describe use of the module pertaining to the package.

Reference material for some aspects of LispWorks can be found in places other than this manual.

- Refer to the Common Lisp Hyperspec for documentation about Common Lisp itself.
- Refer to the *CAPI Reference Manual* for details about symbols in the `CAPI` package, and related packages `GP` and `COLOR`.
- Refer to the *LispWorks Foreign Language Interface User Guide and Reference Manual* for details on integrating C and Common Lisp source code.
- Refer to the *LispWorks Editor User Guide* for documentation on the LispWorks Editor.

1

The CLOS Package

This chapter describes the LispWorks extensions to CLOS, the Common Lisp Object System.

	break-new-instances-on-access	<i>Function</i>
Summary	Breaks to the debugger when a new instance of a class is accessed. Note that this function is deprecated.	
Package	<code>clos</code>	
Signature	<code>break-new-instances-on-access <i>class-designator</i> &key <i>read</i> <i>write</i> <i>slot-names</i> <i>when</i> <i>process</i> <i>trace-output</i> <i>entrycond</i> <i>eval-before</i> <i>before</i> <i>backtrace</i> => <i>t</i></code>	
Arguments	<i>class-designator</i> The class to trap. The keyword arguments control which type of access cause a break and are interpreted as described for <code>trace-on-access</code> .	
Values	Returns <code>t</code> .	

Description	Causes a break when new instances of the class given by <i>class-designator</i> are accessed, according to the keyword arguments. Note: this function is deprecated. You should now call <code>trace-new-instances-on-access</code> with <code>:break t</code> instead.
See also	<code>trace-new-instances-on-access</code>

break-on-access *Function*

Summary	Breaks to the debugger when an instance of a class is accessed. Note that this function is deprecated.
Package	<code>clos</code>
Signature	<code>break-on-access instance &key read write slot-names when process trace-output entrycond eval-before before backtrace => t</code>
Arguments	<p><i>instance</i> A CLOS instance. The keyword arguments control which type of access cause a break and are interpreted as described for <code>trace-on-access</code>.</p>
Values	Returns <i>t</i> .
Description	<p>A useful debugging function which causes access to <i>instance</i> to break to the debugger. Accesses include calls to <code>slot-value</code> and also accessor functions defined by the class of <i>instance</i>. Other instances of the same class are unaffected.</p> <p>You can remove the break by calling <code>unbreak-on-access</code>.</p> <p>A common use of this function is to find where a slot is being changed in a complex program.</p> <p>Note: this function is deprecated. You should now call <code>trace-on-access</code> with <code>:break t</code> instead.</p>

See also [trace-on-access](#)

class-extra-initargs

Generic Function

Summary	Extends the valid initialization arguments of a class.	
Package	clos	
Signature	<code>class-extra-initargs prototype => initargs</code>	
Arguments	<code>prototype</code>	A class prototype.
Values	<code>initargs</code>	A list of additional initialization arguments.
Description	The generic function <code>class-extra-initargs</code> lets you extend the set of valid initialization arguments for a class and its subclasses. <code>initargs</code> should be a list of symbols. Each symbol becomes a valid initarg for the class. By default in a non-delivered LispWorks image, <code>make-instance</code> checks that initargs passed to it are valid.	
<p>Note: <code>class-extra-initargs</code> is useful only in complex cases. In most cases other ways of extending the set of valid initargs are simpler and clearer, such as the <code>:extra-initargs</code> class option, described in <code>defclass</code>.</p>		
Example	<p>In this session an illegal initarg <code>:my-keyword</code> is passed, causing <code>make-instance</code> to signal an error.</p> <p>Then <code>:my-keyword</code> is added as an extra initarg, after which <code>make-instance</code> accepts it.</p>	

```
CL-USER 38 > (defclass my-class () ((a :initform nil)))
#<STANDARD-CLASS MY-CLASS 113AAA2F>

CL-USER 39 > (make-instance 'my-class :my-keyword 8)

Error: MAKE-INSTANCE is called with unknown keyword
:MY-KEYWORD among the arguments (MY-CLASS :MY-KEYWORD
8) {no keywords allowed}
1 (continue) Ignore the keyword :MY-KEYWORD
2 (abort) Return to level 0.
3 Return to top loop level 0.

Type :b for backtrace, :c <option number> to proceed,
or :? for other options

CL-USER 40 : 1 > :a

CL-USER 41 > (defmethod clos:class-extra-initargs
                ((x my-class))
                '(:my-keyword))
#<STANDARD-METHOD CLOS:CLASS-EXTRA-INITARGS (MY-CLASS)
1137C763>

CL-USER 42 > (make-instance 'my-class :my-keyword 8)
#<MY-CLASS 11368963>
```

See also

[compute-class-potential-initargs](#)
[defclass](#)
[make-instance](#)
[set-make-instance-argument-checking](#)

compute-class-potential-initargs

Generic Function

Summary Computes the valid initargs of a class.

Package `clos`

Signature `compute-class-potential-initargs class => initargs`

Arguments `class` A class.

Values	<i>initargs</i>	A list of symbols, or <code>t</code> .
Description		<p>The generic function <code>compute-class-potential-initargs</code> is called to compute the initialization arguments of a class. This set of valid initargs is used by <code>make-instance</code> when its arguments are checked.</p> <p><code>class</code> is the class passed to <code>make-instance</code>. That is, <code>compute-class-potential-initargs</code> specializes on the metaclass.</p> <p><i>initargs</i> is either a list of valid initargs, or <code>t</code> meaning that any initialization argument is allowed.</p> <p>There is a supplied method on <code>t</code>, which returns <code>nil</code>.</p> <p>The other supplied method is on <code>standard-class</code>. This consults the Relevant Methods, which are the applicable methods of <code>make-instance</code>, <code>allocate-instance</code>, <code>initialize-instance</code> and <code>shared-initialize</code>. If any of the Relevant Methods have a lambda list containing <code>&allow-other-keys</code> then <i>initargs</i> is <code>t</code>. Otherwise <i>initargs</i> is a list containing:</p> <ul style="list-style-type: none"> • all the <code>&key</code> arguments from Relevant Method lambda lists, and • the initargs of the slots of <code>class</code> and its superclasses, and • any extra initargs specified via the class option <code>:extra-initargs</code> (see <code>defclass</code> for details of this), and • any extra initargs returned by <code>class-extra-initargs</code>. <p>The list <i>initargs</i> contains no duplicates, and the result of <code>compute-class-potential-initargs</code> is cached so that it is not recomputed unless one of the Relevant Methods, the class or its class precedence list is altered.</p>
See also		<p><code>class-extra-initargs</code> <code>make-instance</code> <code>set-make-instance-argument-checking</code></p>

compute-discriminating-function *Generic Function*

Summary Returns the discriminating function.

Package `clos`

Signature `compute-discriminating-function gf => result`

Arguments `gf` A generic function.

Values `result` A function.

Description The generic function `compute-discriminating-function` returns the discriminator as specified in AMOP.

However, there are two discrepancies with the AMOP behavior:

- The discriminating function does not `compute-applicable-methods-using-classes`, since this is not implemented.
- `add-method` does not call `compute-discriminating-function`. Instead, it is called when the generic function is called. This is more efficient than calling `compute-discriminating-function` each time `add-method` is called.

funcallable-standard-object *Class*

Package `clos`

Superclasses `function`
 `standard-object`

Subclasses `generic-function`

Description	<p>The metaclass <code>funcallable-standard-object</code> provides the default <code>:direct-superclasses</code> for instances of <code>funcallable-standard-class</code> and its subclasses.</p> <p><code>funcallable-standard-object</code> is implemented as described in AMOP except for a different order in the class precedence list.</p> <p>In AMOP the class precedence list is</p> <pre>(funcallable-standard-object standard-object function t)</pre> <p>whereas in LispWorks the class precedence list is</p> <pre>(funcallable-standard-object function standard-object t)</pre> <p>LispWorks is like this to be compliant with the rules in the ANSI Common Lisp Standard.</p> <p>The AMOP class precedence list implies a class precedence for <code>generic-function</code> which violates the last sentence in ANSI Common Lisp 4.2.2 Type Relationships. See www.lisp-works.com/reference/HyperSpec/Body/04_bb.htm.</p>
-------------	--

		<i>Generic Function</i>
Summary		Describes how the value of a class option is parsed.
Package		<code>clos</code>
Signature		<code>process-a-class-option metaclass option value => initargs</code>
Arguments	<code>metaclass</code> <code>option</code> <code>value</code>	The metaclass of the class being parsed. The <code>defclass</code> option name. The tail of the <code>defclass</code> option form.
Values	<code>initargs</code>	A plist of initargs describing the option.

Description The generic function `process-a-class-option` describes how the value of a class option is parsed. It is called at `defclass` macroexpansion time. By default LispWorks parses class options as defined in AMOP, but you need to supply a method if you need class options with different behaviour.
`initargs` should be a plist of class initargs and values. These are added to any other initargs for the class.

Example

```
(defclass m1 (standard-class)
  ((title :initarg :title)))
```

For single-valued, evaluated title option, add a method like this:

```
(defmethod clos:process-a-class-option
  ((class m1)
   (name (eql :title))
   value)
  (unless (and value (null (cdr value)))
    (error "m1 :title must have a single value."))
  (list name (car value)))

(defclass my-titled-class ()
  ()
  (:metaclass m1)
  (:title "Initial Title"))
```

If the value is not to be evaluated, the method would look like this:

```
(defmethod clos:process-a-class-option
  ((class m1)
   (name (eql :title))
   value)
  (unless (and value (null (cdr value)))
    (error "m1 :title must have a single value."))
  `(,name ',value))
```

Now suppose we want an option whose value is a list of titles:

```
(defclass m2 (standard-class)
  ((titles-list :initarg :list-of-possible-titles)))
```

If the titles are to be evaluated, add a method like this:

```
(defmethod clos:process-a-class-option
  ((class m2)
   (name (eql :list-of-possible-titles))
   value)
  (list name `(list ,@value)))
```

Or, if the titles should not be evaluated, add a method like this:

```
(defmethod clos:process-a-class-option
  ((class m2)
   (name (eql :list-of-possible-titles))
   value)
  (list name `',value))

(defclass my-multi-titled-class ()
  ()
  (:metaclass m2)
  (:list-of-possible-titles
   "Initial Title 1"
   "Initial Title 2"))
```

See also [defclass](#)
[process-a-slot-option](#)

process-a-slot-option

Generic Function

Summary	Describes how a <code>defclass</code> slot option is parsed.	
Package	<code>clos</code>	
Signature	<code>process-a-slot-option metaclass option value already-processed-other-options slot => processed-options</code>	
Arguments	<code>metaclass</code>	The metaclass of the class being parsed.
	<code>option</code>	The slot option name.
	<code>value</code>	The value of the slot option.
	<i>already-processed-other-options</i>	

	A plist of initargs for non-standard options that have been processed already.
<i>slot</i>	The whole slot description.
Values	<i>processed-options</i> A plist of initargs.
Description	<p>The generic function <code>process-a-slot-option</code> describes how the value of a slot option is parsed. It is called at <code>defclass</code> macroexpansion time. By default LispWorks parses slot options as defined in AMOP, but you need to supply a method if you need slot options with different behaviour.</p> <p><i>processed-options</i> should be a plist of slot initargs and values containing those from <i>already-processed-other-options</i> together with initargs for <i>option</i> as required. These are added to any other initargs for the slot.</p>
Example	<pre>(defclass extended-class (standard-class)()) (defmethod clos:process-a-slot-option ((class extended-class) option value already-processed-options slot) (if (eq option :extended-slot) (list* :extended-slot value already-processed-options) (call-next-method))) (defclass extended-direct-slot-definition (clos:standard-direct-slot-definition) ((extended-slot :initarg :extended-slot :initform nil))) (defmethod clos:direct-slot-definition-class ((x extended-class) &rest initargs) 'extended-direct-slot-definition) (defclass test () ((regular :initform 3) (extended :extended-slot t :initform 4)) (:metaclass extended-class))</pre>

To add a slot option `:special-reader` whose value is a non-evaluated symbol naming a reader:

```
(defmethod clos:process-a-slot-option
  ((class my-metaclass) option value
   already-processed-options slot)
  (if (and (eq option :special-reader)
            (symbolp value))
      (list* :special-reader
             `',(value already-processed-options)
             (call-next-method))))
```

To allow repeated `:special-reader` options which are combined into a list:

```
(defmethod clos:process-a-slot-option
  ((class my-metaclass) option value
   already-processed-options slot)
  (if (and (eq option :special-reader) (symbolp value))
      (let ((existing (getf
                      already-processed-options
                      :special-reader)))
        (if existing ; this is a quoted list of symbols
            (progn
              (setf (cdr (last (cadr existing))) (list
value)))
              already-processed-options)
            (list* :special-reader
                  `'(,value)
                  already-processed-options)))
      (call-next-method)))
```

See also [defclass](#)
[process-a-class-option](#)

Function

Summary Switches initarg checking in `make-instance` on or off.

Package `clos`

Signature `set-make-instance-initarg-checking on => on`

Arguments	<i>on</i>	A boolean.
Description	The function <code>set-make-instance-initarg-checking</code> provides control over whether <code>make-instance</code> checks its initialization arguments.	
	Calling <code>set-make-instance-initarg-checking</code> with <i>on</i> true, causes <code>make-instance</code> to check the initargs. This is the initial state of LispWorks.	
See also	<code>class-extra-initargs</code> <code>compute-class-potential-initargs</code> <code>make-instance</code>	

	slot-boundp-using-class	<i>Generic Function</i>
Summary	Implements <code>slot-boundp</code> .	
Package	<code>clos</code>	
Signature	<code>slot-boundp-using-class class object slot-name => result</code>	
Arguments	<i>class</i>	A class metaobject, the class of <i>object</i> .
	<i>object</i>	An object.
	<i>slot-name</i>	A slot name.
Values	<i>result</i>	A boolean.
Description	The generic function <code>slot-boundp-using-class</code> implements the behavior of the <code>slot-boundp</code> function.	
	The implementation is as described in AMOP, except that the third argument is the slot name, and not a slot definition	

metaobject. The primary methods specialize on `t` for this argument.

See also `slot-makunbound-using-class`
 `slot-value-using-class`

slot-makunbound-using-class

Generic Function

Summary Implements `slot-makunbound`.

Package `clos`

Signature `slot-makunbound-using-class class object slot-name => object`

Arguments `class` A class metaobject, the class of `object`.
`object` An object.
`slot-name` A slot name.

Values `object` The `object` argument.

Description The generic function `slot-makunbound-using-class` implements the behavior of the `slot-makunbound` function.
The implementation is as described in AMOP, except that the third argument is the slot name, and not a slot definition metaobject. The primary methods specialize on `t` for this argument.

See also `slot-boundp-using-class`
 `slot-value-using-class`

slot-value-using-class

Generic Function

Summary Implements `slot-value`.

Package	<code>clos</code>						
Signature	<code>slot-value-using-class class object slot-name => value</code> <code>(setf slot-value-using-class) value class object slot-name => value</code>						
Arguments	<table> <tr> <td><code>class</code></td><td>A class metaobject, the class of <i>object</i>.</td></tr> <tr> <td><code>object</code></td><td>An object.</td></tr> <tr> <td><code>slot-name</code></td><td>A slot name.</td></tr> </table>	<code>class</code>	A class metaobject, the class of <i>object</i> .	<code>object</code>	An object.	<code>slot-name</code>	A slot name.
<code>class</code>	A class metaobject, the class of <i>object</i> .						
<code>object</code>	An object.						
<code>slot-name</code>	A slot name.						
Values	<code>value</code> The value of the slot named by <i>slot-name</i> .						
Description	<p>The generic function <code>slot-value-using-class</code> implements the behavior of the <code>slot-value</code> function.</p> <p>The implementation is as described in AMOP, except that the third argument is the slot name, and not a slot definition metaobject. The primary methods specialize on <code>t</code> for this argument.</p> <p>Note: by default, standard slot accessors are optimized to not call <code>slot-value-using-class</code>. This can be overridden with the <code>:optimize-slot-access</code> class option. See <code>defclass</code> for details.</p>						
See also	<code>defclass</code> <code>slot-boundp-using-class</code> <code>slot-makunbound-using-class</code>						

	<i>Function</i>
Summary	Traces new instances of a given class, based on access modes.
Package	<code>clos</code>

Signature	<code>trace-new-instances-on-access class-designator &key read write slot-names break when process trace-output entrycond eval-before before backtrace => t</code>
Arguments	<i>class-designator</i> The class to trace. The keyword arguments control which type of access are traced, and provide preconditions for tracing, code to run before access, and how to print any trace output. They are interpreted as described for <code>trace-on-access</code> .
Values	Returns t.
Description	Causes new instances of the class given by <i>class-designator</i> to be traced for the access modes given by <i>read</i> , <i>write</i> and <i>slot-names</i> . This function, when used with the <code>:break</code> keyword, replaces the deprecated function <code>break-new-instances-on-access</code> .
Example	(<code>trace-new-instances-on-access 'capi:display-pane :slot-names nil</code>) Suppose you have a bug whereby the slot <code>bar</code> of an instance of your class <code>foo</code> is incorrectly being set to a negative integer value. You could cause entry into the debugger at the point where the slot is set incorrectly by evaluating this form: <code>(clos:trace-new-instances-on-access 'foo :slot-names '(bar) :read nil :when '(and (integerp (car *traced-arglist*)) (< (car *traced-arglist*) 0)) :break t)</code> and running your program.
See also	<code>break-new-instances-on-access</code> <code>untrace-new-instances-on-access</code> <code>trace-on-access</code>

	<i>Function</i>																								
trace-on-access																									
Summary	Invokes the trace facilities when an instance of a class is accessed.																								
Package	<code>clos</code>																								
Signature	<code>trace-on-access instance &key read write slot-names break when process trace-output entrycond eval-before before backtrace => t</code>																								
Arguments	<table> <tr> <td><i>instance</i></td><td>A CLOS instance.</td></tr> <tr> <td><i>read</i></td><td>A generalized boolean.</td></tr> <tr> <td><i>write</i></td><td>A generalized boolean.</td></tr> <tr> <td><i>slot-names</i></td><td>A list of symbols, or <code>t</code>.</td></tr> <tr> <td><i>break</i></td><td>A generalized boolean.</td></tr> <tr> <td><i>when</i></td><td>A form.</td></tr> <tr> <td><i>process</i></td><td>A form.</td></tr> <tr> <td><i>trace-output</i></td><td>A form.</td></tr> <tr> <td><i>entrycond</i></td><td>A form.</td></tr> <tr> <td><i>eval-before</i></td><td>A list of forms.</td></tr> <tr> <td><i>before</i></td><td>A list of forms.</td></tr> <tr> <td><i>backtrace</i></td><td>A keyword, <code>t</code> or <code>nil</code>.</td></tr> </table>	<i>instance</i>	A CLOS instance.	<i>read</i>	A generalized boolean.	<i>write</i>	A generalized boolean.	<i>slot-names</i>	A list of symbols, or <code>t</code> .	<i>break</i>	A generalized boolean.	<i>when</i>	A form.	<i>process</i>	A form.	<i>trace-output</i>	A form.	<i>entrycond</i>	A form.	<i>eval-before</i>	A list of forms.	<i>before</i>	A list of forms.	<i>backtrace</i>	A keyword, <code>t</code> or <code>nil</code> .
<i>instance</i>	A CLOS instance.																								
<i>read</i>	A generalized boolean.																								
<i>write</i>	A generalized boolean.																								
<i>slot-names</i>	A list of symbols, or <code>t</code> .																								
<i>break</i>	A generalized boolean.																								
<i>when</i>	A form.																								
<i>process</i>	A form.																								
<i>trace-output</i>	A form.																								
<i>entrycond</i>	A form.																								
<i>eval-before</i>	A list of forms.																								
<i>before</i>	A list of forms.																								
<i>backtrace</i>	A keyword, <code>t</code> or <code>nil</code> .																								
Values	Returns <code>t</code> .																								
Description	<p>A useful debugging function which causes access to <i>instance</i> to invoke the trace facilities. Accesses include calls to <code>slot-value</code> and accessor functions defined by the class of <i>instance</i>.</p> <p>The keyword arguments control which type of access are traced, and provide preconditions for tracing, code to run before access, and how to print any trace output. They are</p>																								

similar to those supported by the `trace` macro (but note that these CLOS symbols are functions, so the keyword values are evaluated immediately, unlike in `trace`).

read controls whether reading slots is traced. The default is `t`.

write controls whether writing slots is traced. The default is `t`.

slot-names controls which slots to trace access for. It can be a list of symbols which are the slot-names. The default value, `t`, means trace access to all slots.

break controls whether the debugger is entered when a traced slot in *instance* is accessed. When `nil`, the debugger is not invoked and messages are printed to `*trace-output*`. The default value is `nil`.

when is evaluated during slot access to determine whether any tracing should occur. The default value is `t`.

process is evaluated during slot access to determine whether any tracing should occur in the current process. The form should evaluate to either `nil` (meaning trace in all processes), a string naming the process in which tracing should occur (see `process-name`, `find-process-from-name`), or a list of strings naming the processes in which tracing should occur. The default value is `nil`.

trace-output is evaluated during slot access to determine the stream on which to print tracing messages. If this is `nil` then the value of `*trace-output*` is used. The default value is `nil`.

entrycond is evaluated during slot access to determine whether the default tracing messages should be printed.

eval-before is a list of forms which are evaluated during slot access.

before is a list of forms which are evaluated during slot access. The first value returned by each form is printed.

`backtrace` controls what kind of backtrace to print. If this is `nil` then no backtrace is printed, and this is the default value. Otherwise it can be any of the following values:

- `:quick` Like the `:bq` debugger command.
- `t` Like the `:b` debugger command.
- `:verbose` Like the `:b :verbose` debugger command.
- `:bug-form` Like the `:bug-form` debugger command.

Other instances of the same class are unaffected and you can remove the trace by calling `untrace-on-access`.

The variable `*traced-arglist*` is bound to a list of arguments for the slot access during evaluation of the options above, that is `(instance slot-name)` when reading a slot and `(new-value instance slot-name)` when writing a slot.

A common use of this function is to find where a slot is being changed in a complex program.

This function, when called with `:break t`, replaces the deprecated function `break-on-access`.

See also	<code>untrace-on-access</code> <code>trace-new-instances-on-access</code> <code>break-on-access</code>
----------	--

	Function
Summary	Removes the trapping installed by <code>break-new-instances-on-access</code> . Note that this function is deprecated.
Package	<code>clos</code>
Signature	<code>unbreak-new-instances-on-access class-designator => t</code>
Arguments	<code>class-designator</code> The class whose trap you want to remove.

Values	Returns t.
Description	Removes the trapping installed by <code>break-new-instances-on-access</code> . Note that this function is deprecated. You should now use <code>untrace-new-instances-on-access</code> instead.
See also	<code>untrace-new-instances-on-access</code>

unbreak-on-access *Function*

Summary	Removes the trapping installed by <code>break-on-access</code> . Note that this function is deprecated.
Package	<code>clos</code>
Signature	<code>unbreak-on-access instance</code>
Arguments	<code>instance</code> A class instance
Values	Returns t.
Description	Removes any break installed on <code>instance</code> by <code>break-on-access</code> . See <code>untrace-on-access</code> for details. Note: this function is deprecated. You should now use <code>untrace-on-access</code> instead.
See also	<code>untrace-on-access</code>

untrace-new-instances-on-access *Function*

Summary	Removes the tracing installed by <code>trace-new-instances-on-access</code> .
Package	<code>clos</code>

Signature	<code>untrace-new-instances-on-access class-designator => t</code>
Arguments	<i>class-designator</i> The class whose trap you want to remove.
Values	Returns <i>t</i> .
Description	Removes the tracing installed by <code>trace-new-instances-on-access</code> .
See also	<code>trace-new-instances-on-access</code> <code>untrace-on-access</code>

untrace-on-access *Function*

Summary	Removes the tracing installed by <code>trace-on-access</code> .
Package	<code>clos</code>
Signature	<code>untrace-on-access instance => t</code>
Arguments	<i>instance</i> A class instance
Values	Returns <i>t</i> .
Description	Removes any trace installed on <i>instance</i> by <code>trace-on-access</code> .
See also	<code>trace-on-access</code> <code>untrace-new-instances-on-access</code>

2

The COMM Package

This chapter provides reference entries for the functions in the `comm` package.

The `comm` package provides the TCP/IP interface. TCP/IP sockets can be used to communicate between processes and machines. The TCP/IP mechanism allows LispWorks to connect to or implement a server. It also allows using Secure Sockets Layer (SSL) processing in the socket.

Before the interface can be used the module "`comm`" must be loaded using

```
(require "comm")
```

attach-ssl	<i>Function</i>
Summary	Attaches SSL to a socket stream.
Signature	<code>attach-ssl <i>socket-stream</i> &key <i>ssl-ctx</i> <i>ssl-side</i> <i>ctx-configure-callback</i> <i>ssl-configure-callback</i> => <i>ssl</i></code>
Arguments	<p><code><i>socket-stream</i></code> A <code>socket-stream</code>.</p> <p><code><i>ssl-ctx</i></code> A symbol or a foreign pointer.</p> <p><code><i>ssl-side</i></code> One of the keywords <code>:client</code>, <code>:server</code> or <code>:both</code>.</p>

	<i>ctx-configure-callback</i>	A function designator or <code>nil</code> . The default value is <code>nil</code> .
	<i>ssl-configure-callback</i>	A function designator or <code>nil</code> . The default value is <code>nil</code> .
Values	<code>ssl</code>	A foreign pointer of type <code>ssl-pointer</code> .
Description	<p>The function <code>attach-ssl</code> attaches SSL to the socket-stream socket-stream.</p> <p>The allowed values and meaning of the keyword arguments are as described for <code>socket-stream</code>.</p> <p>Note that <code>attach-ssl</code> is used by</p> <pre>(make-instance 'comm:socket-stream :ssl-ctx ...)</pre> <p>and by</p> <pre>(comm:open-tcp-stream ... :ssl-ctx ...)</pre> <p>but you can also call it explicitly.</p> <p>Before starting to create objects, <code>attach-ssl</code> ensures the SSL library (by calling <code>ensure-ssl</code>) and calls <code>do-rand-seed</code> to seed the Pseudo Random Number Generator (PRNG), so normally you do not need to worry about these.</p> <p>If <code>ssl-ctx</code> is a symbol, it creates the <code>SSL-CTX</code> and calls <code>ctx-configure-callback</code> if this is non-<code>nil</code>. If <code>ssl-ctx</code> is not a <code>ssl-pointer</code>, it creates the <code>SSL</code> object, calls <code>ssl-configure-callback</code> if this is non-<code>nil</code>, and sets the ACCEPT/CONNECT state if the value of <code>ssl-side</code> is not <code>:both</code>. Then it uses <code>SSL-set-fd</code> to attach the <code>SSL</code> to the socket, and records this in the socket stream. It returns the <code>SSL</code>.</p> <p>The default value of <code>ssl-ctx</code> is <code>t</code> and the default value of <code>ssl-side</code> is <code>:server</code>.</p> <p>When a <code>socket-stream</code> is closed, <code>detach-ssl</code> is called with <code>:retry-count nil</code>, which, if the stream is attached to SSL,</p>	

calls `SSL_shutdown` and then frees the object (or objects) that were automatically allocated.

If SSL is already attached to `socket-stream` then `attach-ssl` signals an error.

See also `detach-ssl`

destroy-ssl

Function

Summary Frees a `SSL`.

Package `comm`

Signature `destroy-ssl ssl-pointer`

Arguments `ssl-pointer` A foreign pointer of type `ssl-pointer`.

Description The function `destroy-ssl` frees the `SSL` pointed to by `ssl-pointer` and also frees any LispWorks cached values associated with it.

See also `ssl-pointer`

destroy-ssl-ctx

Function

Summary Frees a `SSL_CTX`.

Package `comm`

Signature `destroy-ssl-ctx ssl-ctx-pointer`

Arguments	<i>ssl-ctx-pointer</i> A foreign pointer of type ssl-ctx-pointer .
Description	The function destroy-ssl-ctx frees the ssl_ctx pointed to by <i>ssl-ctx-pointer</i> and also frees any LispWorks cached values associated with it.
See also	ssl-ctx-pointer

detach-ssl	<i>Function</i>
Summary	Detaches the SSL from a socket stream.
Signature	detach-ssl <i>socket-stream</i> &key <i>retry-count</i> <i>retry-timeout</i>
Arguments	<i>socket-stream</i> A socket-stream . <i>retry-count</i> A non-negative integer. <i>retry-timeout</i>
Description	<p>The function detach-ssl detaches the SSL from the socket-stream <i>socket-stream</i>. If <i>socket-stream</i> is not attached to an SSL, detach-ssl just returns immediately. Otherwise, it detaches the SSL from <i>socket-stream</i>, tries to shut down the SSL cleanly, and then frees the objects that were allocated by attach-ssl.</p> <p><i>retry-count</i> specifies how many additional times to call SSL_shutdown after the second to attempt to get a successful shutdown. The default value of <i>retry-count</i> is 5.</p> <p><i>retry-timeout</i> specifies the time in seconds to wait between each of the calls to SSL_shutdown. If it fails to get a successful shutdown after these attempts, detach-ssl signals an error. The default value of <i>retry-timeout</i> is 0.1.</p> <p>Note that the shutdown calls happen after the SSL has been detached from <i>socket-stream</i> as far as LispWorks is concerned, so if an error occurs at this point and is aborted, <i>socket-stream</i></p>

can be used in `attach-ssl` again (assuming that the peer can cope with this situation).

If `retry-count` is `nil`, `detach-ssl` does not try to get a successful shutdown call. This value is used when the stream is closed, but should not be used normally.

See also `attach-ssl`

do-rand-seed

Function

Summary	Calls the SSL function <code>RAND_seed</code> .
Package	<code>comm</code>
Signature	<code>do-rand-seed</code>
Arguments	<code>do-rand-seed</code> takes no arguments.
Values	<code>do-rand-seed</code> returns no values.
Description	The function <code>do-rand-seed</code> calls the SSL function <code>RAND_seed</code> with some suitable value, which is dependent in a non-trivial way on the current time, the history of the current process and the history of the machine it is running on. If the machine that it runs on has the file <code>/dev/urandom</code> , <code>do-rand-seed</code> does nothing.

See also `attach-ssl`

ensure-ssl

Function

Summary	Initializes SSL.
---------	------------------

Signature	<code>ensure-ssl &key library-path already-done</code>	
Arguments	<i>library-path</i>	A string or a list of strings.
	<i>already-done</i>	A generalized boolean.
Description	<p>The function <code>ensure-ssl</code> initializes SSL. If it was already called in the image, <code>ensure-ssl</code> does nothing. Otherwise it loads the library, calls <code>SSL_library_init</code>, calls <code>SSL_load_error_strings</code> and performs internal initializations.</p> <p>If <i>already-done</i> is true, <code>ensure-ssl</code> does only the internal initializations. The default value of <i>already-done</i> is <code>nil</code>.</p> <p>If <i>library-path</i> is passed, it needs to be either a string, specifying one library, or a list of strings specifying multiple libraries. The default value of <i>library-path</i> is platform-specific. The initial default value is described in the section "Loading the OpenSSL libraries" in the <i>LispWorks User Guide</i> and this default may be changed by calling <code>set-ssl-library-path</code>.</p>	
See also	<code>openssl-version</code> <code>set-ssl-library-path</code>	

get-host-entry

Function

Summary	Returns address or name information about a given host.	
Package	<code>comm</code>	
Signature	<code>get-host-entry host &key fields => field-values</code>	
Arguments	<i>host</i>	A number or a string.
	<i>fields</i>	A list of keywords.

Values	<i>field-values</i>	Values, one for each field.								
Description		<p>Using whatever host naming services are configured on the current machine, <code>get-host-entry</code> returns address or name information about the given host. <code>nil</code> is returned if the host is unknown.</p> <p>The <i>host</i> argument can be one of the following:</p> <ul style="list-style-type: none"> • a name string, for example "<code>www.foobar.com</code>" • a dotted inet address string, for example "<code>209.130.14.246</code>" • a integer representing the inet address, for example <code>#xD1820EF6</code> <p>The <i>fields</i> argument is a list of keywords describing what information to return for the host. If <code>get-host-entry</code> succeeds, it returns multiple values, one value for each field specified. The following fields are allowed:</p> <table> <tr> <td><code>:address</code></td> <td>The primary IP address as an integer.</td> </tr> <tr> <td><code>:addresses</code></td> <td>A list of all the IP addresses as integers.</td> </tr> <tr> <td><code>:name</code></td> <td>The primary name as a string.</td> </tr> <tr> <td><code>:aliases</code></td> <td>The alias names as a list of strings.</td> </tr> </table> <p>Note: although the results of <code>get-host-entry</code> are not cached by LispWorks, the operating System might cache them.</p>	<code>:address</code>	The primary IP address as an integer.	<code>:addresses</code>	A list of all the IP addresses as integers.	<code>:name</code>	The primary name as a string.	<code>:aliases</code>	The alias names as a list of strings.
<code>:address</code>	The primary IP address as an integer.									
<code>:addresses</code>	A list of all the IP addresses as integers.									
<code>:name</code>	The primary name as a string.									
<code>:aliases</code>	The alias names as a list of strings.									

Examples

```
CL-USER 16 > (comm:get-host-entry "www.altavista.com"
                                     :fields '(:address))
3511264349

CL-USER 17 > (comm:get-host-entry 3511264349
                                     :fields '(:name))
"altavista.com"

CL-USER 18 > (comm:get-host-entry "altavista.com"
                                     :fields '(:name
                                               :address
                                               :aliases))
"altavista.com"
3511264349
("www.altavista.com" "www.altavista.com")
```

get-socket-address *Function*

Summary	Returns the local address and port number of a given socket.	
Package	<code>comm</code>	
Signature	<code>get-socket-address <i>socket</i> => <i>address, port</i></code>	
Arguments	<code><i>socket</i></code>	A socket handle.
Values	<code><i>address</i></code>	The local host address of the socket or <code>nil</code> if not connected.
	<code><i>port</i></code>	The local port number of the socket or <code>nil</code> if not connected.
Description	Connected sockets have two addresses, local and remote. The <code>get-socket-address</code> function returns the local address.	
See also	<code>get-socket-peer-address</code> <code>socket-stream-address</code>	

		<i>Function</i>
Summary		Returns the remote address and port number of a given socket.
Package		<code>comm</code>
Signature		<code>get-socket-peer-address socket => address, port</code>
Arguments	<i>socket</i>	A socket handle.
Values	<i>address</i>	The remote host address of the socket or <code>nil</code> if not connected.
	<i>port</i>	The remote port number of the socket or <code>nil</code> if not connected.
Description		Connected sockets have two addresses, local and remote. The <code>get-socket-peer-address</code> function returns the remote address.
See also		<code>get-socket-address</code> <code>socket-stream-peer-address</code>

		<i>Function</i>
Summary		Returns the mode of the SSL.
Package		<code>comm</code>
Signature		<code>get-verification-mode ssl-or-ssl-ctx => result</code>
Arguments	<i>ssl-or-ssl-ctx</i>	A foreign pointer of type <code>ssl-pointer</code> or <code>ssl-ctx-pointer</code> .

Values	<i>result</i>	A list of symbols.
Description	The function <code>get-verification-mode</code> returns the mode of the <code>ssl-pointer</code> or <code>ssl-ctx-pointer</code> as a list of symbols. <i>result</i> is a list containing zero or more of the symbols <code>:verify-client-once</code> , <code>:verify-peer</code> and <code>:fail-if-no-peer-cert</code> , corresponding to the C constants <code>VERIFY_CLIENT_ONCE</code> , <code>VERIFY_PEER</code> and <code>FAIL_IF_NO_PEER_CERT</code> respectively.	
See also	<code>set-verification-mode</code>	

		<i>Function</i>
Summary	Returns the dotted IP address string from the integer IP address.	
Package	<code>comm</code>	
Signature	<code>ip-address-string ip-address => string-ip-address</code>	
Arguments	<i>ip-address</i>	An integer.
Values	<i>string-ip-address</i>	The dotted string format of the given IP address.
Description	The <code>ip-address-string</code> function converts its argument to a string in the standard dotted IP address notation <code>a.b.c.d</code> .	
See also	<code>string-ip-address</code>	

make-ssl-ctx

Function

Summary	Makes a <code>ssl_ctx</code> object.				
Package	<code>comm</code>				
Signature	<code>make-ssl-ctx &key ssl-ctx ssl-side => ssl-ctx-ptr</code>				
Arguments	<table><tr><td><code>ssl-ctx</code></td><td>A symbol or a foreign pointer.</td></tr><tr><td><code>ssl-side</code></td><td>One of the keywords <code>:client</code>, <code>:server</code> or <code>:both</code>.</td></tr></table>	<code>ssl-ctx</code>	A symbol or a foreign pointer.	<code>ssl-side</code>	One of the keywords <code>:client</code> , <code>:server</code> or <code>:both</code> .
<code>ssl-ctx</code>	A symbol or a foreign pointer.				
<code>ssl-side</code>	One of the keywords <code>:client</code> , <code>:server</code> or <code>:both</code> .				
Values	<code>ssl-ctx-ptr</code> A foreign pointer of type <code>ssl-ctx-pointer</code> .				
Description	<p>The function <code>make-ssl-ctx</code> first calls <code>ensure-ssl</code>, and returns a foreign pointer of type <code>ssl-ctx-pointer</code>.</p> <p>If the value of <code>ssl-ctx</code> is <code>t</code>, <code>:default</code>, <code>:v2</code>, <code>:v3</code>, <code>:v23</code> or <code>:tls-v1</code>, <code>make-ssl-ctx</code> creates a <code>ssl_ctx</code> object and returns a pointer to it.</p> <p>The value of <code>ssl-ctx</code> can also be a foreign pointer of type <code>ssl-ctx-pointer</code>, in which case it is simply returned. If <code>ssl-ctx</code> is a foreign pointer of type <code>ssl-pointer</code>, then <code>make-ssl-ctx</code> signals an error.</p> <p>The meaning of the keyword arguments <code>ssl-ctx</code> and <code>ssl-side</code> is as described for <code>socket-stream</code>. The default value of <code>ssl-ctx</code> is <code>t</code> and the default value of <code>ssl-side</code> is <code>:server</code>.</p>				
See also	<code>ensure-ssl</code> <code>socket-stream</code> <code>ssl-ctx-pointer</code>				

	open-tcp-stream	<i>Function</i>
Summary	Attempts to connect to a socket on another machine and returns a stream object for the connection.	
Package	<code>comm</code>	
Signature	<code>open-tcp-stream hostname service &key direction element-type errorp read-timeout write-timeout timeout ssl-ctx ctx-configure- callback ssl-configure-callback local-address local-port nodelay keepalive => stream-object</code>	
Arguments	<p><code>hostname</code> An integer or string.</p> <p><code>service</code> A string or a fixnum.</p> <p><code>direction</code> One of <code>:input</code>, <code>:output</code> or <code>:io</code>.</p> <p><code>element-type</code> <code>base-char</code> or a subtype of <code>integer</code>.</p> <p><code>errorp</code> A boolean.</p> <p><code>read-timeout</code> A positive number, or <code>nil</code>.</p> <p><code>write-timeout</code> A positive number, or <code>nil</code>.</p> <p><code>timeout</code> A positive number, or <code>nil</code>.</p> <p><code>ssl-ctx</code> A symbol or a foreign pointer.</p> <p><code>ctx-configure-callback</code> A function designator or <code>nil</code>. The default value is <code>nil</code>.</p> <p><code>ssl-configure-callback</code> A function designator or <code>nil</code>. The default value is <code>nil</code>.</p> <p><code>local-address</code> <code>nil</code>, an integer or a string.</p> <p><code>local-port</code> <code>nil</code>, a string or a fixnum.</p> <p><code>nodelay</code> A generalized boolean.</p> <p><code>keepalive</code> A generalized boolean.</p>	

Values	<i>stream-object</i>	A socket stream.
Description	<p>The <code>open-tcp-stream</code> function attempts to connect to a socket on another machine and returns <i>stream-object</i> for the connection if successful. The server machine to connect to is given by <i>hostname</i>, which can be one of the following:</p> <ul style="list-style-type: none"> • A string naming the host, for example <code>"www.nowhere.com"</code> • A string providing the IP address, for example <code>"204.71.177.75"</code> • An integer IP address in network order, for example <code>#xCC47B14B</code> <p>The name of the service to provide is given by <i>service</i>. If <i>service</i> is a string, the location of the file specifying the names of the services available varies, but typically on Windows 98 it is called <code>SERVICES</code> and is stored in the <code>Windows</code> directory, and on Windows NT-based systems it is the file</p> <pre>%SystemRoot%\system32\drivers\etc\SERVICES</pre> <p>The <i>service</i> can also be a fixnum representing the port number of the desired connection.</p> <p>The direction of the connection is given by <i>direction</i>. Its default value is <code>:io</code>. The element type of the connection is determined from <i>element-type</i>, and is <code>base-char</code> by default.</p> <p>If <i>errorp</i> is <code>nil</code>, failure to connect (possibly after <i>timeout</i> seconds) returns <code>nil</code>, otherwise an error is signaled.</p> <p><i>timeout</i> specifies a connection timeout. <code>open-tcp-stream</code> waits for at most <i>timeout</i> seconds for the TCP connection to be made. If <i>timeout</i> is <code>nil</code> it waits until the connection attempt succeeds or fails. On failure, <code>open-tcp-stream</code> signals an error or returns <code>nil</code> according to the value of <i>errorp</i>. To provide a timeout for reads after the connection is made, see <i>read-timeout</i>. The default value of <i>timeout</i> is <code>nil</code>.</p>	

read-timeout specifies the read timeout of the stream. If it is `nil` (the default), the stream does not time out during reads, and these may hang. See `socket-stream` for more details. To provide a connection timeout, see `timeout`.

write-timeout is similar to *read-timeout*, but for writes. See `socket-stream` for more details.

`ssl-ctx`, `ctx-configure-callback` and `ssl-configure-callback` are interpreted as described for `socket-stream`. Unlike the other ways of creating a socket stream with SSL processing, `open-tcp-stream` does not take the *ssl-side* argument and always uses the value `:client`.

If *local-address* is `nil` then the operating system chooses the local address of the socket. Otherwise the string or integer value is interpreted as for `hostname` and specifies the local address of the socket. The default value of *local-address* is `nil`.

If *local-port* is `nil` then the operating system chooses the local port of the socket. Otherwise the string or fixnum value is interpreted as for `service` and specifies the local port of the socket. The default value of *local-port* is `nil`.

If *keepalive* is true, `SO_KEEPALIVE` is set on the socket. The default value of *keepalive* is `nil`.

If *nodelay* is true, `TCP_NODELAY` is set on the socket. The default value of *nodelay* is `t`.

Example

The following example opens an HTTP connection to a given host, and retrieves the root page:

```

(with-open-stream (http (comm:open-tcp-stream
                           "www.lispworks.com" 80))
  (format http "GET / HTTP/1.0~C~C~C"
          (code-char 13) (code-char 10)
          (code-char 13) (code-char 10))
  (force-output http)
  (write-string "Waiting to reply...")
  (loop for ch = (read-char-no-hang http nil :eof)
        until ch
        do (write-char #\.)
        (sleep 0.25)
        finally (unless (eq ch :eof)
                  (unread-char ch http)))
  (terpri)
  (loop for line = (read-line http nil nil)
        while line
        do (write-line line)))

```

See also **`start-up-server`**
`socket-stream`

openssl-version

Function

Summary	Returns the version of the loaded OpenSSL library.	
Package	<code>comm</code>	
Signature	<code>openssl-version &optional what => result</code>	
Arguments	<code>what</code>	One of the keywords <code>:version</code> , <code>:directory</code> , <code>:platform</code> , <code>:cflags</code> and <code>:built-on</code> .
Values	<code>result</code>	A string.
Description	The function <code>openssl-version</code> returns a string specifying the version of the loaded OpenSSL library.	
	The argument <code>what</code> takes these values:	

<code>:version</code>	<i>result</i> is the version string, which looks like: "OpenSSL 0.9.7i 14 Oct 2005" or "OpenSSL 0.9.8a 11 Oct 2005"
<code>:built-on</code>	Returns a string specifying when it was built.
<code>:directory</code>	Returns where OpenSSL thinks it is installed.
<code>:platform</code>	Returns OpenSSL's idea of which platforms it is.
<code>:cflags</code>	The compilation command.

The default value of *what* is `:version`.

See also

`ensure-ssl`

pem-read

Function

Summary	An interface to the SSL <code>PEM_read_bio_*</code> functions.		
Package	<code>comm</code>		
Signature	<code>pem-read thing-to-read filename &key pass-phrase callback errorp => result</code>		
Arguments	<code>thing-to-read</code>	A string.	
	<code>filename</code>	A pathname designator.	
	<code>pass-phrase</code>	A string, or <code>nil</code> .	
	<code>callback</code>	A function designator, or <code>nil</code> .	
	<code>errorp</code>	A generalized boolean.	
Values	<code>result</code>	A foreign pointer or <code>nil</code> .	

Description	<p>The function <code>pem-read</code> is an interface to the <code>PEM_read_bio_*</code> set of functions. See the manual entry for <code>pem</code> for specifications of these functions.</p> <p><i>thing-to-read</i> defines which function is required. <code>pem-read</code> concatenates <i>thing-to-read</i> with the string " <code>PEM_read_bio_</code>" to form the name of the <code>pem</code> function to call.</p> <p><i>filename</i> specifies the file to load.</p> <p>If <i>pass-phrase</i> is non-<code>nil</code>, it must be a string, which is passed to the <code>pem</code> function. The default value of <i>pass-phrase</i> is <code>nil</code>.</p> <p>If <i>callback</i> is non-<code>nil</code>, it must be a function with signature:</p> <pre>callback maximum-length rwflag => pass-phrase</pre> <p>where <i>maximum-length</i> is an integer, <i>rwflag</i> is a boolean and <i>pass-phrase</i> is the pass-phrase to use. The default value of <i>callback</i> is <code>nil</code>, but you cannot pass non-<code>nil</code> values for both <i>pass-phrase</i> and <i>callback</i>.</p> <p>If it succeeds, <code>pem-read</code> returns a foreign pointer to the structure that was returned by the <code>pem</code> function. If <code>pem-read</code> fails, if <i>errorp</i> is non-<code>nil</code> it signals an error, otherwise it returns <code>nil</code>. The default value of <i>errorp</i> is <code>nil</code>.</p>
-------------	---

read-dhparams

Function

Summary	Reads or uses cached SSL DH parameters.	
Package	<code>comm</code>	
Signature	<code>read-dhparams filename &key pass-phrase callback errorp force => dh-ptr</code>	
Arguments	<code>filename</code>	A pathname designator.
	<code>pass-phrase</code>	A string, or <code>nil</code> .

	<i>callback</i>	A function designator, or <code>nil</code> .
	<i>errorp</i>	A generalized boolean.
	<i>force</i>	A generalized boolean.
Values	<i>dh-ptr</i>	A foreign pointer or <code>nil</code> .
Description	The function <code>read-dhparams</code> reads or uses cached DH parameters. <i>filename</i> specifies the file to check. Unless <i>force</i> is true, <code>read-dhparams</code> checks if the file <i>filename</i> has already been loaded, and if it has been loaded, uses the cached value. If <i>force</i> is true, or if there is no cached value for <i>filename</i> , <code>read-dhparams</code> loads the file by calling <code>pem-read</code> with <i>thing-to-read</i> argument "DHparams", <i>pass-phrase</i> , <i>callback</i> and <i>errorp</i> . <code>read-dhparams</code> caches and returns a foreign pointer to the resulting DH structure (that is, a pointer corresponding to the C type <code>DH*</code>). If <code>read-dhparams</code> fails to load the file <i>filename</i> , if <i>errorp</i> is true it signals an error, otherwise it returns <code>nil</code> . The default value of <i>errorp</i> is <code>t</code> .	
See also	<code>pem-read</code>	

set-verification-mode

Function

Summary	Sets the verification mode for CTX.
Package	<code>comm</code>
Signature	<code>set-verification-mode ssl-ctx ssl-side mode &optional callback</code>

Arguments	<i>ssl-ctx</i>	A foreign pointer of type <code>ssl-pointer</code> or <code>ssl-ctx-pointer</code> .
	<i>ssl-side</i>	<code>:server</code> or <code>:client</code> .
	<i>mode</i>	An integer, one of the symbols <code>:never</code> , <code>:always</code> , <code>:once</code> , or a list of keywords.
	<i>callback</i>	A foreign function.
Values	<i>result</i>	A list of symbols.
Description	The function <code>set-verification-mode</code> sets the verification mode for CTX according to arguments <i>ssl-side</i> and <i>mode</i> . When <i>ssl-side</i> is <code>:server</code> , <i>mode</i> can be:	
	An integer	<i>mode</i> is passed directly to <code>SSL_set_verify</code> or <code>SSL_CTX_set_verify</code> .
	<code>:never</code>	The server will not send a client certificate request to the client, so the client will not send a certificate.
	<code>:always</code>	The server sends a client certificate request to the client. The certificate returned (if any) is checked. If the verification process fails, the TLS/SSL handshake is immediately terminated with an alert message containing the reason for the verification failure.
	<code>:once</code>	Same as <code>:always</code> except that the client certificate is checked only on the initial TLS/SSL handshake, and not again in case of renegotiation.
	A list	The list contains (some of) the keywords <code>:verify-client-once</code> , <code>:verify-peer</code> and <code>:fail-if-no-peer-cert</code> . These keywords map to the corresponding C constants <code>VERIFY_CLIENT_ONCE</code> , <code>VERIFY_PEER</code> and

`FAIL_IF_NO_PEER_CERT` respectively. See the manual entry for `ssl_ctx_set_verify` for the meaning of the constants.

When `ssl-side` is `:client`, `mode` can be:

An integer	<code>mode</code> is passed directly as for <code>ssl-side :server</code> .
<code>:never</code>	If not using an anonymous cipher, the server will send a certificate which will be checked by the client. The handshake will be continued regardless of the verification result.
<code>:always</code>	The server certificate is verified. If the verification process fails, the TLS/SSL handshake is immediately terminated with an alert message containing the reason for the verification failure. If no server certificate is sent because an anonymous cipher is used, verification is ignored.
A list	The list contains keywords as described above for <code>ssl-side :server</code> .
	If non- <code>nil</code> <code>callback</code> should be a symbol, function, string or foreign pointer designating a foreign function that is called to perform verification. The default value of <code>callback</code> is <code>nil</code> .

See also [get-verification-mode](#)

set-ssl-ctx-dh

Function

Summary Sets the DH parameters for a `ssl_ctx`.

Package `comm`

Signature	<code>set-ssl-ctx-dh ssl-ctx &key dh filename func filename-list pass-phrase callback => result</code>	
Arguments	<i>ssl-ctx</i>	A foreign pointer.
	<i>filename</i>	A pathname designator or <code>nil</code> .
	<i>func</i>	A function designator or <code>nil</code> .
	<i>filename-list</i>	An association list.
	<i>pass-phrase</i>	A string, or <code>nil</code> .
	<i>callback</i>	A function designator, or <code>nil</code> .
Values	<i>result</i>	A boolean.
Description	<p>The function <code>set-ssl-ctx-dh</code> sets the DH parameters for a <code>SSL_CTX</code>.</p> <p><code>ssl-ctx</code> can be either a foreign pointer of type <code>ssl-ctx-pointer</code> or a foreign pointer of type <code>ssl-pointer</code>.</p> <p>The value to use is specified by one of the parameters <code>dh</code>, <code>filename</code>, <code>func</code> or <code>filename-list</code>.</p> <p>If <code>dh</code> is non-<code>nil</code>, it must be a foreign pointer to a DH (corresponding to the C type <code>DH*</code>), and this DH is used as-is. The default value of <code>dh</code> is <code>nil</code>.</p> <p>Otherwise, if <code>filename</code> is non-<code>nil</code>, it must be a pathname designator for a file containing DH parameters, which is loaded (by <code>read-dhparams</code>) and then used. In this case, <code>pass-phrase</code> and <code>callback</code> can be used, and are passed to <code>pem-read</code>.</p> <p>Otherwise, if <code>func</code> is non-<code>nil</code>, it must be a function with signature:</p> <pre><code>func is-export keylength => dh-ptr</code></pre> <p>where <code>is-export</code> is a boolean, <code>keylength</code> is an integer, and <code>dh-ptr</code> is a pointer to an appropriate DH structure. <code>set-ssl-ctx-dh</code> installs <code>func</code> as the DH callback.</p>	

Otherwise (that is, if each of *dh*, *filename* and *func* are `nil`) then *filename-list* must be a non-`nil` association list of keylengths and filenames, sorted by the keylengths in ascending order (that is, larger keylengths are towards the end of the list). `set-ssl-ctx-dh` installs a DH callback which when called finds the first keylength which is equal or bigger than the required keylength, loads the associated file (by calling `read-dhparams`), and returns it. It also loads the first file of the list immediately.

result is `t` on success, `nil` otherwise.

See also

`pem-read`
`read-dhparams`
`ssl-ctx-pointer`
`ssl-pointer`

set-ssl-ctx-options

Function

Summary	Sets the options in a <code>ssl_CTX</code> .				
Package	<code>comm</code>				
Signature	<code>set-ssl-ctx-options ssl-ctx &key microsoft_sess_id_bug netscape_challenge_bug netscape_reuse_cipher_change_bug sslref2_reuse_cert_type_bug microsoft_big_sslv3_buffer msie_sslv2_rsa_padding ssleay_080_client_dh_bug tls_d5_bug tls_block_padding_bug dont_insert_empty_fragments_all no_session_resumption_on_renegotiation single_dh_use ephemeral_rsa cipher_server_preference tls_rollback_bug no_sslv2 no_sslv3 no_tls1 pkcs1_check_1 pkcs1_check_2 netscape_ca_dn_bug netscape_demo_cipher_change_bug</code>				
Arguments	<table> <tr> <td><code>ssl-ctx</code></td> <td>A foreign pointer.</td> </tr> <tr> <td>Each of the keyword arguments is a generalized boolean defaulting to <code>nil</code>.</td> <td></td> </tr> </table>	<code>ssl-ctx</code>	A foreign pointer.	Each of the keyword arguments is a generalized boolean defaulting to <code>nil</code> .	
<code>ssl-ctx</code>	A foreign pointer.				
Each of the keyword arguments is a generalized boolean defaulting to <code>nil</code> .					

Description	The function <code>set-ssl-ctx-options</code> sets the options in a <code>SSL_CTX</code> . <code>ssl-ctx</code> can be either a foreign pointer of type <code>ssl-ctx-pointer</code> or a foreign pointer of type <code>ssl-pointer</code> . The option that is set is the <code>logior</code> of all the options that are passed to <code>set-ssl-ctx-options</code> via the keyword arguments. The value used for each non- <code>nil</code> keyword <code>keyword</code> is the value of <code>SSL_OP_</code> <i>keyword</i> . The meaning of the options is specified in the OpenSSL manual page for <code>SSL_set_options</code> .
See also	<code>ssl-ctx-pointer</code> <code>ssl-pointer</code>

set-ssl-ctx-password-callback

Function

Summary	Sets the password for a <code>SSL_CTX</code> .						
Package	<code>comm</code>						
Signature	<code>set-ssl-ctx-password-callback ssl-ctx &key callback password</code>						
Arguments	<table> <tr> <td><code>ssl-ctx</code></td><td>A foreign pointer.</td></tr> <tr> <td><code>callback</code></td><td>A function designator, or <code>nil</code>.</td></tr> <tr> <td><code>password</code></td><td>A string, or <code>nil</code>.</td></tr> </table>	<code>ssl-ctx</code>	A foreign pointer.	<code>callback</code>	A function designator, or <code>nil</code> .	<code>password</code>	A string, or <code>nil</code> .
<code>ssl-ctx</code>	A foreign pointer.						
<code>callback</code>	A function designator, or <code>nil</code> .						
<code>password</code>	A string, or <code>nil</code> .						
Description	<p>The function <code>set-ssl-ctx-password-callback</code> sets the password for a <code>SSL_CTX</code>, either to a callback or a password.</p> <p><code>ssl-ctx</code> should be a foreign pointer of type <code>ssl-ctx-pointer</code>.</p> <p>If <code>callback</code> is non-<code>nil</code>, it must be a function with signature:</p> $\text{callback } \text{maximum-length } \text{rwflag} \Rightarrow \text{result}$						

where *maximum-length* is an integer, *rwflag* is a boolean and *result* is a string. The default value of *callback* is `nil`.

If *password* is non-`nil` and *callback* is `nil`, a callback is installed that simply returns *password*. The default value of *password* is `nil`.

If both *callback* and *password* are `nil`, `set-ssl-ctx-password-callback` signals an error.

See also `ssl-ctx-pointer`

`set-ssl-library-path`

Function

Summary Sets the SSL library path.

Package `comm`

Signature `set-ssl-library-path library-path`

Arguments *library-path* A string or a list of strings.

Description The function `set-ssl-library-path` sets the SSL library path.

library-path should a string or a list of strings. Each string specifies a library to load. The libraries are loaded in the order they are in the list.

Note that in contrast to `ensure-ssl`, the effect of `set-ssl-library-path` persists after saving and restarting the image.

See also `ensure-ssl`

		<i>Class</i>
socket-error		
Summary		The condition class for socket errors.
Superclasses		<code>simple-error</code>
Subclasses		<code>ssl-condition</code>
Initargs	<code>:stream</code>	A <code>socket-stream</code> .
Description		The condition class for socket errors.
socket-stream		
Summary		The socket stream class.
Superclasses		<code>buffered-stream</code>
Initargs	<code>:socket</code>	A socket handle.
	<code>:direction</code>	One of <code>:input</code> , <code>:output</code> , or <code>:io</code> .
	<code>:element-type</code>	An element type.
	<code>:read-timeout</code>	A positive number or <code>nil</code> .
	<code>:write-timeout</code>	A positive number or <code>nil</code> .
	<code>:ssl-ctx</code>	A keyword, <code>t</code> or <code>nil</code> , or a foreign pointer of type <code>ssl-ctx-pointer</code> or <code>ssl-pointer</code> .
	<code>:ssl-side</code>	One of the keywords <code>:client</code> , <code>:server</code> or <code>:both</code> . The default value is <code>:server</code> .
	<code>:ctx-configure-callback</code>	A function designator or <code>nil</code> .
	<code>:ssl-configure-callback</code>	A function designator or <code>nil</code> .
Accessors		<code>socket-stream-socket</code>

Readers	<code>stream:stream-read-timeout</code> <code>stream:stream-write-timeout</code>
Description	<p>The <code>socket-stream</code> class implements a buffered stream connected to a socket. The socket handle, specified by <code>:socket</code>, and the direction, specified by <code>:direction</code>, must be passed for a meaningful stream to be constructed. Common Lisp input functions such as <code>read-char</code> will see <code>end-of-file</code> if the other end of the socket is closed.</p> <p>The <code>:element-type</code> keyword determines the expected element type of the stream traffic. However, stream input and output functions for character and binary data generally work in the obvious way on a <code>socket-stream</code> with <code>element-type base-char</code>, (<code>unsigned-byte 8</code>) or (<code>signed-byte 8</code>). For example, <code>read-sequence</code> can be called with a string buffer and a binary <code>socket-stream</code>: the character data is constructed from the input as if by <code>code-char</code>. Similarly <code>write-sequence</code> can be called with a string buffer and a binary <code>socket-stream</code>: the output is converted from the character data as if by <code>char-code</code>. Also, 8-bit binary data can be read and written to a <code>base-char</code> <code>socket-stream</code>.</p> <p>All standard stream I/O functions except for <code>write-byte</code> and <code>read-byte</code> have this flexibility.</p> <p>The <code>:read-timeout</code> initarg specifies the read timeout in seconds, or is <code>nil</code>, meaning there are no timeouts during reads (this is the default).</p> <p>The <code>read-timeout</code> property is intended for use when a socket connection might hang during a call to any Common Lisp input function. The <code>read-timeout</code> can be set by <code>make-instance</code> or by <code>open-tcp-stream</code>. It can also be modified by <code>(setf stream:stream-read-timeout)</code>. When <code>read-timeout</code> is <code>nil</code>, there is no timeout during reads and the call may hang. When <code>read-timeout</code> is not <code>nil</code>, and there is no input from the socket for more than <code>read-timeout</code> seconds, any reading function returns <code>end-of-file</code>. The <code>read-timeout</code> does not limit the time inside <code>read</code>, but the time between successful extractions</p>

of data from the socket. Therefore, if the reading needs several rounds it may take longer than *read-timeout*.

Using `(setf stream:stream-read-timeout)` on the stream while it is inside a read function has undefined effects. However, the setf function can be used between calls to read functions. The *read-timeout* property of a stream can be read by

```
(stream:stream-read-timeout stream)
```

The `:write-timeout` initarg specifies the write timeout in seconds, or is `nil`, meaning that there are no timeouts during writes (this is the default).

The *write-timeout* property is similar to *read-timeout*, but for write operations. If flushing the stream buffer takes too long then `error` is called.

The keyword arguments `:ssl-ctx`, `:ssl-side`, `:ctx-configure-callback` and `:ssl-configure-callback` can be passed to create and configure socket streams with SSL processing.

`ssl-ctx`, if non-`nil`, specifies that the stream uses SSL and further specifies the `ssl_ctx` object to use. The value of `ssl-ctx` can be a symbol which, together with `ssl-side`, specifies which protocol to use. The value `t` or `:default` means use the default, which is currently the same as `:v23`. The values `:v2`, `:v3`, `:v23` and `:tls-v1` are mapped to the `SSLv2_*`, `SSLv3_*`, `SSLv23_*` and `TLSv1_*` methods respectively. With these symbol values of `ssl-ctx`, LispWorks makes a new `ssl_ctx` object and uses it and frees it when the stream is closed.

The value of `ssl-ctx` can also be a foreign pointer of type `ssl-ctx-pointer` (which corresponds to the C type `SSL_CTX*`). This is used and is not freed when the stream is closed. Also an SSL object is made and used, and this object is freed when the stream is closed. The foreign pointer may be a result of a call to `make-ssl-ctx`, but it can also a result of user code, provided that it points to a valid `ssl_ctx` and has the type `ssl-ctx-pointer`.

The value of *ssl-ctx* can also be a foreign pointer of type **ssl-pointer** (which corresponds to the C type `SSL*`). This specifies the SSL to use. This maybe a result of a call to **ssl-new** but can also be the result of user code, provided that it points to a valid SSL object and has the type **ssl-pointer**. The SSL is used and is not freed when the stream is closed.

When you pass a **ssl-ctx-pointer** or a **ssl-pointer** foreign pointer as the *ssl-ctx* argument, it must have already been set up correctly.

ssl-side specifies which side the socket stream is. The value of *ssl-side* is used in two cases:

- When a new **ssl_ctx** object is created, it is used to select the method:

```
:client => *_client_method
:server => *_server_method
:both   => *_method
```

- When a new SSL object is created, when *ssl-side* is either **:client** or **:server**, LispWorks calls **SSL_set_connect_state** or **SSL_set_accept_state** respectively.

If the value of *ssl-ctx* is a **ssl-pointer**, *ssl-side* is ignored.

ctx-configure-callback specifies a callback, a function which takes a foreign pointer of type **ssl-ctx-pointer**. This is called immediately after a new **ssl_ctx** is created. If the value of *ssl-ctx* is not a symbol, *ctx-configure-callback* is ignored.

ssl-configure-callback specifies a callback, a function which takes a foreign pointer of type **ssl-pointer**. This is called immediately after a new SSL is created. If the value of *ssl-ctx* is not a **ssl-pointer**, *ssl-configure-callback* is ignored.

Example

The following makes a bidirectional stream connected to a socket specified by *handle*.

```
(make-instance 'comm:socket-stream
               :socket handle
               :direction :io
               :element-type 'base-char)
```

This example creates a socket stream with a read-timeout:

```
(make-instance 'comm:socket-stream
               :handle handle
               :direction :input
               :read-timeout 42)
```

The following form illustrates character I/O in a binary **socket-stream**:

```
(with-open-stream (x
                   (comm:open-tcp-stream
                     "localhost" 80
                     :element-type '(unsigned-byte 8)))
                   (write-sequence (format nil "GET / HTTP/1.0~%~%") x)
                   (force-output x)
                   (let ((res (make-array 20 :element-type 'base-char)))
                     (values (read-sequence res x) res)))
```

The following form illustrates binary I/O in a **base-char socket-stream**:

```
(with-open-stream (x
                   (comm:open-tcp-stream
                     "localhost" 80
                     :element-type 'base-char))
                   (write-sequence
                     (map #'(simple-array (unsigned-byte 8) 1)
                           'char-code
                           (format nil "GET / HTTP/1.0~%~%"))
                     x)
                   (force-output x)
                   (let ((res (make-array 20
                                         :element-type
                                         '(unsigned-byte 8))))
                     (values (read-sequence res x)
                             (map 'string 'code-char res))))
```

See also

[open-tcp-stream](#)
[start-up-server](#)

```
stream-read-timeout
wait-for-input-streams
```

		<i>Function</i>
socket-stream-address		
Summary	Returns the local address and port number of a given socket stream.	
Package	<code>comm</code>	
Signature	<code>socket-stream-address stream => address, port</code>	
Arguments	<code>stream</code>	A socket stream.
Values	<code>address</code>	The local host address of the socket stream or <code>nil</code> if not connected.
	<code>port</code>	The local port number of the socket stream or <code>nil</code> if not connected.
Description	Connected socket streams have two addresses, local and remote. The <code>socket-stream-address</code> function returns the local address.	
See also	<code>socket-stream-peer-address</code> <code>get-socket-address</code>	

		<i>Function</i>
socket-stream-ctx		
Summary	Accesses the <code>ssl_ctx</code> attached to a socket stream.	
Package	<code>comm</code>	
Signature	<code>socket-stream-ctx socket-stream => ssl-ctx-pointer</code>	
Arguments	<code>socket-stream</code>	A <code>socket-stream</code> .

Values	<code>ssl-ctx-pointer</code>	A foreign pointer of type <code>ssl-ctx-pointer</code> , or <code>nil</code> .
Description	The function <code>socket-stream-ctx</code> accesses the <code>SSL_CTX</code> that is attached to the <code>socket-stream</code> <i>socket-stream</i> . It returns <code>nil</code> if SSL is not attached.	
See also		<code>socket-stream</code> <code>ssl-ctx-pointer</code>

socket-stream-peer-address

Function

Summary	Returns the remote address and port number of a given socket stream.	
Package	<code>comm</code>	
Signature	<code>socket-stream-peer-address stream => address, port</code>	
Arguments	<code>stream</code>	A socket stream.
Values	<code>address</code>	The remote host address of the socket stream or <code>nil</code> if not connected.
	<code>port</code>	The remote port number of the socket stream or <code>nil</code> if not connected.
Description	Connected socket streams have two addresses, local and remote. The <code>socket-stream-peer-address</code> function returns the remote address.	
See also	<code>socket-stream-address</code> <code>get-socket-peer-address</code>	

socket-stream-ssl		<i>Function</i>
Summary	Accesses the <code>ssl</code> attached to a socket stream.	
Package	<code>comm</code>	
Signature	<code>socket-stream-ssl socket-stream => ssl-pointer</code>	
Arguments	<code>socket-stream</code>	A <code>socket-stream</code> .
Values	<code>ssl-pointer</code>	A foreign pointer of type <code>ssl-pointer</code> , or <code>nil</code> .
Description	The function <code>socket-stream-ssl</code> accesses the <code>ssl</code> that is attached to the <code>socket-stream</code> <code>socket-stream</code> . It returns <code>nil</code> if SSL is not attached.	
See also	<code>socket-stream</code> <code>ssl-pointer</code>	
ssl-cipher-pointer		<i>FLI type specifier</i>
Summary	An FLI type for use with SSL.	
Package	<code>comm</code>	
Signature	<code>ssl-cipher-pointer</code>	
Description	The FLI type <code>ssl-cipher-pointer</code> corresponds to the C type <code>SSL_CIPHER*</code> .	
ssl-cipher-pointer-stack		<i>FLI type specifier</i>
Summary	An FLI type for use with SSL.	

Package	<code>comm</code>
Signature	<code>ssl-cipher-pointer-stack</code>
Description	The FLI type <code>ssl-cipher-pointer-stack</code> corresponds to the C type <code>STACK_OF(SSL_CIPHER)</code> .

ssl-closed *Class*

Summary	The class for SSL errors corresponding to <code>SSL_ERROR_ZERO_RETURN</code> .
Superclasses	<code>ssl-condition</code>
Description	The condition class <code>ssl-closed</code> corresponds to <code>SSL_ERROR_ZERO_RETURN</code> . It means the underlying socket is dead.

ssl-condition *Class*

Summary	The condition class for SSL errors.
Superclasses	<code>socket-error</code>
Subclasses	<code>ssl-closed</code> <code>ssl-error</code> <code>ssl-failure</code> <code>ssl-x509-lookup</code>
Description	The condition class for errors inside SSL.

ssl-ctx-pointer *FLI type specifier*

Summary	An FLI type for use with SSL.
---------	-------------------------------

Package	<code>comm</code>
Signature	<code>ssl-ctx-pointer</code>
Description	The FLI type <code>ssl-ctx-pointer</code> corresponds to the C type <code>SSL_CTX*</code> .

ssl-error *Class*

Summary	The class for SSL errors corresponding to <code>SSL_ERROR_SYSCALL</code> .
Superclasses	<code>ssl-condition</code>
Description	The condition class <code>ssl-error</code> corresponds to <code>SSL_ERROR_SYSCALL</code> . It means that something got broken.

ssl-failure *Class*

Summary	The class for SSL errors corresponding to <code>SSL_ERROR_SSL</code> .
Superclasses	<code>ssl-condition</code>
Description	The condition class <code>ssl-failure</code> corresponds to <code>SSL_ERROR_SSL</code> . This means a failure in processing the input, typically due to a mismatch between the client and the server. You get this error when trying to use a SSL connection to a non-secure peer.

ssl-new *Function*

Summary	Creates a <code>ssl</code> .
---------	------------------------------

Package	<code>comm</code>
Signature	<code>ssl-new ssl-ctx-pointer => ssl-pointer</code>
Arguments	<code>ssl-ctx-pointer</code> A foreign pointer of type <code>ssl-ctx-pointer</code> .
Values	<code>ssl-pointer</code> A foreign pointer of type <code>ssl-pointer</code> .
Description	<p>The function <code>ssl-new</code> creates a <code>SSL</code> by a direct call to the C function <code>SSL_new</code>.</p> <p>It returns a pointer to the new <code>SSL</code>.</p>
See also	<code>ssl-ctx-pointer</code> <code>ssl-pointer</code>

ssl-pointer		<i>FLI type specifier</i>
Summary	An FLI type for use with SSL.	
Package	<code>comm</code>	
Signature	<code>ssl-pointer</code>	
Description	The FLI type <code>ssl-pointer</code> corresponds to the C type <code>SSL*</code> .	

ssl-x509-lookup		<i>Class</i>
Summary	The class for SSL errors corresponding to <code>SSL_ERROR_WANT_X509_LOOKUP</code> .	
Superclasses	<code>ssl-condition</code>	
Description	The condition class <code>ssl-x509-lookup</code> corresponds to <code>SSL_ERROR_WANT_X509_LOOKUP</code> . It happens when a certificate is rejected by a user callback.	

		<i>Function</i>
	start-up-server	
Summary		Starts a TCP server.
Package		<code>comm</code>
Signature		<code>start-up-server &key function announce service address nodelay keepalive process-name wait error => process, startup-condition</code>
Arguments	<i>function</i>	A function name.
	<i>announce</i>	An output stream, <code>t</code> , <code>nil</code> or a function.
	<i>service</i>	An integer, a string or <code>nil</code> .
	<i>address</i>	An integer, a string or <code>nil</code> .
	<i>nodelay</i>	A generalized boolean.
	<i>keepalive</i>	A generalized boolean.
	<i>process-name</i>	A symbol or expression.
	<i>wait</i>	A boolean.
	<i>error</i>	A boolean.
Values	<i>process</i>	A process, or <code>nil</code> .
	<i>startup-condition</i>	A condition object, or <code>nil</code> .
Description		<p>The <code>start-up-server</code> function starts a TCP server. Use <code>process-kill</code> to kill the server, and <code>open-tcp-stream</code> to send messages from another client to the server.</p> <p>The <i>function</i> argument provides the name of the function that processes connections. When a connection is made <i>function</i> is called with the connected socket handle, at which point you can make a stream using <code>make-instance</code> and communicate with the client. The server does not accept more connections until <i>function</i> returns, so normally it should create another light-weight process to handle the connection. However, the operating system typically provides a small queue of</p>

partially accepted connections, which prevents connection failure for new clients until the server is ready to accept more connections. If *function* is not specified the built-in Lisp listener server is used. See the examples section below.

If *announce* is a stream or `t` (denoting `*standard-output*`), a message appears on the stream when the server is started.

If *announce* is a function it is called when the server is started. *announce* should take two arguments: *socket* and *condition*. *socket* is the socket used by the server: *announce* can therefore be used to record this socket. *condition* describes the error if there is one. *announce* can be called with *socket nil* and a condition only if *error* is *nil*. If the process is killed, *announce* is called with *socket nil* and *condition nil*.

The default for *announce* is *nil*, meaning there is no message.

If *service* is a string or positive integer, it specifies the name of the service. The location of the file specifying the names of services available varies, but typically on Windows 98 it is called `SERVICES` and is stored in the `windows` directory, and on Windows NT-based systems it is the file

```
%SystemRoot%\system32\drivers\etc\SERVICES
```

If *service* is *nil* or 0, then `start-up-server` chooses a free port. The default value for *service* is "lispworks".

If *address* is a string or integer that can be resolved to an IP address, then the server only receives connections for that IP address. This must be one of the addresses associated with the machine and allowed values are a string naming a host, such as "`www.nowhere.com`", a string providing the IP address, such as "`204.71.177.75`", or an integer IP address in network order, such as `#xCC47B14B`.

If *address* is *nil* or 0, then the server will receive connections to all IP addresses on the machine. This is the default.

If *keepalive* is true, `SO_KEEPALIVE` is set on the socket. The default value of *keepalive* is *nil*.

If *nodelay* is true, TCP_NODELAY is set on the socket. The default value of *nodelay* is t.

The *process-name* specifies the process name. The default is constructed from the service name in the following fashion:

```
(format nil "~S server" service)
```

The *wait* argument controls whether *start-up-server* waits for the server to start or returns immediately. When *wait* is non-nil and an error was signalled, *process* is nil and the error is returned in *startup-condition*. Otherwise just one value, the server process, is returned. The default for *wait* is nil.

The *error* argument controls what happens if an error is signalled in the server thread. If *error* is nil then the thread is terminated. If *error* is non-nil then the debugger is entered. The default value for *error* is (not *wait*).

Note: some versions of Microsoft Windows fail to detect the case where more than one server binds a given port, so an error will not be raised in this situation.

Examples

The following example uses the built-in Lisp listener server:

```
(comm:start-up-server :service 10243)
```

It makes a Lisp listener server on port 10243 (check with local network managers that this port number is safe to use). When a client connects to this, Lisp calls *read*. The client should send a string using Common Lisp syntax followed by a newline. This string is used to name a new light-weight process that runs a Lisp listener. When this has been created, the server waits for more connections.

The next example illustrates the use of the *function* argument. For each line of input read by the server it writes the line back with a message. The stream generates EOF if the other end closes the connection.

```
(defvar *talk-port* 10244) ; a free TCP port number
```

```

(defun make-stream-and-talk (handle)
  (let ((stream (make-instance 'comm:socket-stream
                               :socket handle
                               :direction :io
                               :element-type
                               'base-char)))
    (mp:process-run-function (format nil "talk ~D"
                                      handle)
                             '()
                             'talk-on-stream stream)))

(defun talk-on-stream (stream)
  (unwind-protect
      (loop for line = (read-line stream nil nil)
            while line
            do
            (format stream "You sent: '~A'~%" line)
            (force-output stream))
    (close stream)))

(comm:start-up-server :function 'make-stream-and-talk
                      :service *talk-port*)

```

This is a client which uses the talk server:

```

(defun talking-to-myself ()
  (with-open-stream
    (talk (comm:open-tcp-stream "localhost"
                                *talk-port*))
    (dolist (monolog
              '("Hello self."
                "Why don't you say something original?"
                "Talk to you later then. Bye."))
      (write-line monolog talk)
      (force-output talk)
      (format t "I said: \"~A\"~%"
              monolog)
      (format t "Self replied: \"~A\"~%"
              (read-line talk nil nil))))))

```

```
(talking-to-myself)
=>
I said: "Hello self."
Self replied: "You sent: 'Hello self.'"
I said: "Why don't you say something original?"
Self replied: "You sent: 'Why don't you say something
original?''"
I said: "Talk to you later then. Bye."
Self replied: "You sent: 'Talk to you later then.
Bye.'"
```

This example illustrates a server which picks a free port and records the socket. The last form queries the socket for the port used.

```
(defvar *my-socket* nil)

(defun my-announce-function (socket condition)
  (if socket
      (setf *my-socket* socket)
      (my-log-error condition)))

(comm:start-up-server :service nil
                      :error nil
                      :announce 'my-announce-function)

(multiple-value-bind (address port)
    (comm:get-socket-address *my-socket*)
  port)
```

See also [open-tcp-stream](#)
[socket-stream](#)

start-up-server-and-mp		<i>Function</i>
Package	comm	
Signature	start-up-server-and-mp &key <i>function announce service address process-name</i>	
Arguments	<i>function</i>	A function name.
	<i>announce</i>	An output stream, t, nil or a function.

	<i>service</i>	An integer, a string or <code>nil</code> .
	<i>address</i>	An integer, a string or <code>nil</code> .
	<i>process-name</i>	A symbol or expression.
Description	The <code>start-up-server-and-mp</code> function starts multiprocess-ing if it has not already been started and then calls <code>start-up-server</code> with the supplied <i>function</i> , <i>announce</i> , <i>service</i> , <i>address</i> and <i>process-name</i> arguments.	
Note: <code>start-up-server-and-mp</code> is implemented only on Unix/Linux/Mac OS X platforms.		
See also	<code>start-up-server</code>	

	string-ip-address	<i>Function</i>
Summary	Returns the integer IP address from the given dotted IP address string.	
Package	<code>comm</code>	
Signature	<code>string-ip-address ip-address-string => ip-address</code>	
Arguments	<i>string-ip-address</i> A string denoting an IP address in dotted format.	
Values	<i>ip-address</i>	An integer IP address.
Description	The <code>string-ip-address</code> function takes a string in the standard dotted IP address notation <code>a.b.c.d</code> and returns the corresponding integer IP address.	
See also	<code>ip-address-string</code>	

	with-noticed-socket-stream	<i>Macro</i>
Package	<code>comm</code>	
Signature	<code>with-noticed-socket-stream (stream) &body body</code>	
Arguments	<p><code>stream</code> A stream created using <code>open-tcp-stream</code>.</p> <p><code>body</code> Code to be executed while the stream is “noticed”.</p>	
Description	<p>If you do a <code>process-wait</code> inside the body of some code, and the LispWorks system has no more processes to run, it will wait for some external event. This macro alerts the system to the fact that any event on the socket associated with the given stream should cause the system wake and check the predicates for processes doing a <code>process-wait</code>.</p> <p>The net effect is that, without using this macro, data coming into the socket will not cause a wake-up. The socket will only be checked again when another event (such as a mouse event) causes the system to wake and re-check the wait function for each waiting processes.</p> <p>The macro is designed to be used with streams created by the function <code>open-tcp-stream</code>.</p> <p>Note: <code>with-noticed-socket-stream</code> is implemented only on Unix/Linux/Mac OS X platforms.</p>	
See also	<code>open-tcp-stream</code>	

3

The COMMON-LISP Package

This chapter describes the LispWorks extensions to symbols in the `COMMON-LISP` package, which is used by default. This chapter notes only those differences between LispWorks and the ANSI Common Lisp standard. You should refer to this standard (an HTML version, the Common Lisp Hyperspec, is supplied with LispWorks) for full documentation about standard Common Lisp symbols.

	<i>Function</i>
apropos	
Summary	Searches for interned symbols.
Package	<code>common-lisp</code>
Signature	<code>apropos string &optional package external-only => <no values></code>
Arguments	
	<code>string</code> A string designator.
	<code>package</code> A package designator or <code>nil</code> .
	<code>external-only</code> A generalised boolean.

Description	The function <code>apropos</code> behaves as specified in ANSI Common Lisp. There is an additional optional argument <code>external-only</code> , which if true restricts the search to symbols which are external in the searched package or packages. The default value of <code>external-only</code> is <code>nil</code> .
See also	<code>apropos-list</code> <code>*describe-print-length*</code> <code>*describe-print-level*</code> <code>regexp-find-symbols</code>

	<i>Function</i>						
apropos-list							
Summary	Searches for interned symbols.						
Package	<code>common-lisp</code>						
Signature	<code>apropos-list string &optional package external-only => symbols</code>						
Arguments	<table> <tr> <td><code>string</code></td><td>A string designator.</td></tr> <tr> <td><code>package</code></td><td>A package designator or <code>nil</code>.</td></tr> <tr> <td><code>external-only</code></td><td>A generalised boolean.</td></tr> </table>	<code>string</code>	A string designator.	<code>package</code>	A package designator or <code>nil</code> .	<code>external-only</code>	A generalised boolean.
<code>string</code>	A string designator.						
<code>package</code>	A package designator or <code>nil</code> .						
<code>external-only</code>	A generalised boolean.						
Values	<code>symbols</code> A list of symbols.						
Description	The function <code>apropos-list</code> behaves as specified in ANSI Common Lisp. There is an additional optional argument <code>external-only</code> , which if true restricts the search to symbols which are external in the searched package or packages. The default value of <code>external-only</code> is <code>nil</code> .						
See also	<code>apropos</code>						

		<i>Type</i>
base-string		
Summary	The base string type.	
Package	<code>common-lisp</code>	
Signature	<code>base-string length</code>	
Arguments	<code>length</code>	The length of the string (or *, meaning any).
Description	The type of base strings.	
close		<i>Generic Function</i>
Summary	The <code>close</code> function is implemented as a generic function.	
Package	<code>common-lisp</code>	
Signature	<code>close stream &key abort => result</code>	
Arguments	<code>stream</code>	A stream.
	<code>abort</code>	A generalized boolean.
Values	<code>result</code>	A boolean.
Description	The standard function <code>close</code> is implemented as a generic function. All external resources used by the stream should be freed and true returned when that has been done. The result value for <code>close</code> is as per the Common Lisp ANSI specification.	
	When <code>stream</code> is an instance of a subclass of <code>buffered-stream</code> , if <code>abort</code> is true then any remaining data in the buffer can be discarded. There are two built-in methods on <code>buffered-stream</code> . The primary method specialized on <code>buffered-stream</code> returns t. The other, an :around method specialized	

on **buffered-stream**, flushes the stream buffer if *abort* is `nil`, calls the next method and marks the stream as closed if that method returns true. Thus the only requirement for a primary method specialized on a subclass of **buffered-stream** is that it must close any underlying data source and return true.

The **close** method on the **fundamental-stream** class sets a flag for **open-stream-p**

See also	buffered-stream fundamental-stream open-stream-p
----------	---

	<i>Function</i>				
coerce					
Summary	Extends the standard coerce function, allowing it to take any Common Lisp type specifier.				
Package	common-lisp				
Signature	coerce object result-type => result				
Arguments	<table> <tr> <td><i>object</i></td><td>A Lisp object.</td></tr> <tr> <td><i>result-type</i></td><td>A type specifier.</td></tr> </table>	<i>object</i>	A Lisp object.	<i>result-type</i>	A type specifier.
<i>object</i>	A Lisp object.				
<i>result-type</i>	A type specifier.				
Values	<i>result</i> An object of type <i>result-type</i>				
Description	The coerce function still performs those conversions required by the standard, but a larger set of type specifiers is allowed for coercion. A type-error is signalled if <i>result</i> cannot be returned as the <i>result-type</i> specifies.				
See also	concatenate				

compile	<i>Function</i>
Summary	Compiles a lambda expression into a code vector.
Package	<code>common-lisp</code>
Signature	<code>compile name &optional definition => name, function</code>
Arguments	<p><i>definition</i> If supplied, this is a lambda-expression to be compiled. If not supplied, then the lambda-expression used is the current definition of the name (in this case <i>name</i> must be a non-<code>nil</code> symbol with an uncompiled definition).</p> <p><i>name</i> If not <code>nil</code>, this is the symbol that is to receive the compiled function as its global function definition.</p>
Values	A single value is returned, being the <i>name</i> symbol if supplied, or when <i>name</i> is <code>nil</code> the compiled function definition itself. Such compiled-function objects are not printable (but see <code>disassemble</code>) other than as <code>#<compiled function for SYMBOL></code> .
Description	<code>compile</code> calls the compiler to translate a lambda expression into a code vector containing an equivalent sequence of host specific machine code. A compiled function typically runs between 10 and 100 times faster. It is generally worth compiling the most frequently called Lisp functions in a large application during the development phase. The compiler detects a large number of programming errors, and the resulting code runs sufficiently faster to justify the compilation time, even during development.
Examples	<pre>(defun fn (...) ...) ; interpreted definition for fn (compile 'fn) ; replace with compiled ; definition</pre>

```
(compile nil '(lambda (x) (* x x)))
; returns compiled squaring function

(compile 'cube '(lambda (x) (* x x x)))
; defun and compile in one
```

Notes	See <code>declare</code> for a list of the declarations that alter the behavior of the compiler.
See also	<code>compile-file</code> <code>disassemble</code> <code>declare</code>

compile-file *Function*

Summary	Compile a Lisp source file into a form that both loads and runs faster.	
Package	<code>common-lisp</code>	
Signature	<code>compile-file input-file &key output-file verbose print external-format load => output-truename, warnings-p, failure-p</code>	
Arguments	<i>input-file</i>	A pathname designator.
	<i>output-file</i>	A pathname designator, or <code>:temp</code> .
	<i>verbose</i>	A generalized boolean.
	<i>print</i>	A generalized boolean.
	<i>external-format</i>	An external format specification.
	<i>load</i>	A generalized boolean.
Values	<i>output-truename</i>	A pathname or <code>nil</code> .
	<i>warnings-p</i>	A generalized boolean.
	<i>failure-p</i>	A generalized boolean.

Description `compile-file` calls the compiler to translate a Lisp source file into a form that both loads and runs faster. A compiled function typically runs more than ten times faster than when interpreted (assuming that it is not spending most of its work calling already compiled functions). A source file with a `.lisp` or `.lsp` extension compiles to produce a file with a `.*fasl` extension (the actual extension depends on the host machine CPU). Subsequent use of `load` loads the compiled version (which is in LispWorks's FASL or Fast Load format) in preference to the source.

In compiling a file the compiler has to both compile each function and top level form in the file, and to produce the appropriate FASL directives so that loading has the desired effect. In particular objects need to have space allocated for them, and top level forms are called as they are loaded.

output-file specifies the location of the output file. This argument is useful if you are using a non-default file extension for binary files. If you use a non-default file extensions for binary files, you must inform LispWorks of this by pushing the file extension string onto the variable `sys::*binary-file-types*`. If you fail to do this, LispWorks assumes that these files are text rather than compiled files. See the example below.

The special value *output-file* `:temp` offers a convenient way to specify that the output file is a temporary file in a location that is likely to be writable.

verbose controls the printing of messages describing the file being compiled, the current optimization settings, and other information. If *verbose* is `nil`, there are no messages. If *verbose* is `0`, only the "Compiling file..." message is printed. For all other true values of *verbose*, messages are also printed about:

- compiler optimization settings before the file is processed, and

- multiple matches when *input-file* does not specify the pathname type, and
- any clean down (garbage collection) that occurs during the compilation.

The default value is the value of `*compile-verbose*`, which defaults to `t`.

print controls the printing of information about the compilation. It can have the following values. If *print* is `nil`, no information is printed. If *print* is a non-positive number, then only warnings are printed. If *print* is a positive number no greater than 1, or if *print* is any non-number object, then the information printed consists of all warning messages and one line of information per function that is compiled. If *print* is a number greater than 1, then full information is printed. The default value of *print* is the value of `*compile-print*`, which has the default value 1.

external-format is interpreted as for `open`. The default value is `:default`.

If *load* is true, then the file is loaded after compilation.

output-truename is the `truename` of the output file, or `nil` if that cannot be created.

warnings-p is `nil` if no conditions of type `error` or `warning` were detected during compilation. Otherwise *warnings-p* is a list containing the conditions.

failure-p is `nil` if no conditions of type `error` or `warning` (other than `style-warning`) were detected by the compiler, and `t` otherwise.

Examples

```
(compile-file "devel/fred.lisp")
  ;; compile fred.lisp to fred.fas1
(compile-file "devel/fred")
  ;; does the same thing

(compile-file "test" :load t)
  ;; compile test.lisp, then load if successful
```

```

  (compile-file "program" :output-file "program.abc")
    ;; compile "program.lisp" to "program.abc"

  (push "abc" sys::*binary-file-types*)
    ;; tells LispWorks that files with extension
    ;; ".abc" are binaries

```

Notes See `declare` for a list of the declarations that alter the behavior of the compiler.

The act of compiling a file should have no side effects, other than the creation of symbols and packages as the input file is read by the reader.

By default a form is skipped if an error occurs during compilation. If you need to debug an error during compilation by `compile-file`, set `*compiler-break-on-error*`.

During compilation of a file `foo.lisp` (on an Intel Macintosh, for example) a temporary output file named `t_foo.xfasl` is used, so that an unsuccessful compile does not overwrite an existing `foo.xfasl`.

LispWorks uses the following naming conventions for fasl files, and it is recommended that you should use them too, to ensure correct operation of `load` and so on.

Table 3.1 Naming conventions for FASL files

Machine/Implementation	Fasl Extension
x86 Windows/32-bit LispWorks	.ofasl
x64 Windows/64-bit LispWorks	.64ofasl
x86 Linux/32-bit LispWorks	.ufasl
amd64 Linux/64-bit LispWorks	.64ufasl
x86 FreeBSD/32-bit LispWorks	.ffasl
HP-PA/32-bit LispWorks	.pfasl

Table 3.1 Naming conventions for FASL files

Machine/Implementation	Fasl Extension
SPARC/32-bit LispWorks	.wfasl
SPARC/64-bit LispWorks	.64wfasl
Intel Macintosh/32-bit LispWorks	.xfasl
PowerPC Macintosh/32-bit LispWorks	.nfasl
Intel Macintosh/64-bit LispWorks	.64xfasl
PowerPC Macintosh/64-bit LispWorks	.64nfasl

You can find the fasl file extension appropriate for your machine by looking at the variable `system::*binary-file-type*`. The variable `system::*binary-file-types*` contains a list of all the file extensions currently recognized by `load` and `load-data-file`.

Compatibility Note	In LispWorks for Windows 4.4 and previous, the fasl file extension is <code>.fasl</code> . This changed in LispWorks 5.0. In LispWorks for Linux 4.4 and previous, the fasl file extension is <code>.ufs1</code> . This changed in LispWorks 5.0.
--------------------	--

See also	<code>compile</code> <code>compile-file-if-needed</code> <code>*compiler-break-on-error*</code> <code>disassemble</code>
----------	---

concatenate	<i>Function</i>
Summary	Extends the standard <code>concatenate</code> function allowing it to take any Common Lisp type.
Package	<code>common-lisp</code>

Signature	<code>concatenate result-type &rest sequences => result-sequence</code>	
Arguments	<code>result-type</code>	A type specifier.
	<code>sequences</code>	A sequence.
Values	<code>result-sequence</code>	A sequence.
Description	The <code>concatenate</code> function has been extended to take any Common Lisp type. The <code>result-sequence</code> will be of type <code>result-type</code> unless this is not possible, in which case a <code>type-error</code> is signalled.	
See also	<code>coerce</code>	

declaim	<i>Macro</i>
Summary	Established a specified declarations.
Package	<code>common-lisp</code>
Signature	<code>declaim &rest declarations</code>
Arguments	<code>declarations</code> Declaration forms.
Description	The macro <code>declaim</code> behaves as specified in the ANSI Common Lisp Standard with one exception: for a top-level call to <code>declaim</code> , optimize declarations are omitted from the compiled binary file. This is useful because you are unlikely to want to change these settings outside of that file.
See also	<code>compile-file</code> <code>declare</code> <code>proclaim</code>

declare	<i>Special Form</i>
Summary	Declares a variable as special, provides advice to the Common Lisp system, or helps the programmer to optimize code.
Package	<code>common-lisp</code>
Signature	<code>declare declaration*</code>
Arguments	<code>declaration</code> A declaration specifier, not evaluated.
Values	<p>The special form <code>declare</code> behaves computationally as if it is not present (other than to affect the semantics), and is only allowed in certain contexts, such as after the variable list in a <code>let</code>, <code>do</code>, <code>defun</code> and so on.</p> <p>(Consult the syntax definition of each special form to see if it takes <code>declare</code> forms and/or documentation strings.)</p>
Description	<p>There are three distinct uses of <code>declare</code>: one is to declare Lisp variables as “special” (this affects the semantics of the appropriate bindings of the variables), the second is to provide advice to help the Common Lisp system (in reality the compiler) run your Lisp code faster or with more sophisticated debugging options, and the third (using the <code>:explain</code> declaration) is to help you optimize your code.</p> <p>If you use <code>declare</code> to specify types (and so eliminate type-checking for the specified symbols) and then supply the wrong type, you may obtain a “Segmentation Violation”. You can check this by interpreting the code (rather than compiling it).</p> <p>The following are extensions to the Common Lisp definition of <code>declare</code>:</p> <ul style="list-style-type: none"> • <code>lambda-list</code> specifies the value to be returned when a programmer asks for the arglist of a function.

- **values** specifies the value to be returned when you ask for a description of the results of a function.
- **hcl:invisible-frame** specifies that calls to this function should not appear in a debugger backtrace.
- **hcl:alias** specifies that calls to this function should be displayed as calls to an alternative function in a debugger backtrace.
- **:explain** controls messages printed by the compiler while it is processing forms.

The remainder of this description documents the syntax and use of **:explain** declarations.

```
declaration ::= (:explain option*)
option ::= optionkey | (optionkey optionvalue)
optionkey ::= :none | :variables | :types | :floats |
:non-floats | :all-calls | :all-calls-with-arg-types | 
:calls | :boxing | :print-original-form | :print-
expanded-form | :print-length | :print-level
```

The **:explain** declaration controls messages printed by the compiler while it is processing forms. The declaration can be used with **proclaim** or **declare** as a top level form to give it global or file scope. It can also be used at the start of a **#'lambda** form or function body to give it the scope of that function. The declaration has unspecified effect when used in other contexts, for example in the body of a **let** form.

An **:explain** declaration consists of a set of options of the form (*optionkey* *optionvalue*) which associates *optionvalue* with *optionkey* or *optionkey* which associates **t** with *optionkey*. By default, all of the *optionkeys* have an associated value **nil**. All *optionkeys* not specified by a declaration remain unchanged (except for the special action of the **:none** *optionkey* described below).

The *optionkey* should be one of the following:

:none	Set value associated with all <i>optionkeys</i> to <code>nil</code> . This turns off all explanations.
:variables	If <i>optionvalue</i> is non- <code>nil</code> , list all the variables of each function, specifying whether they are floating point or not.
:types	If <i>optionvalue</i> is non- <code>nil</code> , print information about compiler transformations that depend on declared or deduced type information.
:floats	If <i>optionvalue</i> is non- <code>nil</code> , print information about calls to functions that may allocate floats.
:non-floats	If <i>optionvalue</i> is non- <code>nil</code> , print information about calls to functions that may allocate non-float numbers, for example bignums.
:all-calls	If <i>optionvalue</i> is non- <code>nil</code> , print information about calls to normal functions.
:all-calls-with-arg-types	If <i>optionvalue</i> is non- <code>nil</code> , print the argument types for calls to normal functions. Must be combined with :all-calls .
:calls	A synonym for :all-calls .
:boxing	If <i>optionvalue</i> is non- <code>nil</code> , print information about calls to functions that may allocate numbers, for example floats or bignums.
:print-original-form	If <i>optionvalue</i> is non- <code>nil</code> , modifies the :all-calls , :floats and :non-floats explanations to include the original source code form that contains the call.
:print-expanded-form	

	If <i>optionvalue</i> is non- <code>nil</code> , modifies the <code>:all-calls</code> , <code>:floats</code> and <code>:non-floats</code> explanations to include the macroexpanded source code form that contains the call.
<code>:print-length</code>	Use the <i>optionvalue</i> as the value of <code>*print-length*</code> for <code>:all-calls</code> , <code>:floats</code> and <code>:non-floats</code> explanations.
<code>:print-level</code>	Use the <i>optionvalue</i> as the value of <code>*print-level*</code> for <code>:all-calls</code> , <code>:floats</code> and <code>:non-floats</code> explanations.

Example	(defun <code>foo</code> (<code>arg</code>) (declare (<code>:explain :variables</code>) (optimize (float 0))) (let* ((<code>double-arg</code> (coerce <code>arg</code> 'double-float)) (<code>next</code> (+ <code>double-arg</code> 1d0)) (<code>other</code> (* <code>double-arg</code> 1/2))) (values <code>next</code> <code>other</code>))) ; <i>-- Variables with non-floating point types:</i> ; <i>-- ARG OTHER</i> ; <i>-- Variables with floating point types:</i> ; <i>-- DOUBLE-ARG NEXT</i>
---------	--

See also	<code>compile</code> <code>compile-file</code> <code>proclaim</code>
----------	--

defclass	<i>Macro</i>
Summary	Remains as defined in ANSI Common Lisp, but extra control over parsing of class options and slot options, optimization of slot access, and checking of initargs, is provided.
Package	<code>common-lisp</code>
Description	The macro <code>defclass</code> is as defined in the ANSI standard with the following extensions.

For extra class options, you may need to define the way these are parsed at `defclass` macroexpansion time. See `process-a-class-option` for details.

For non-standard slot options, you may need to define the way these are parsed at `defclass` macroexpansion time. See `process-a-slot-option` for details.

By default, standard slot accessors are optimized such that they do not call `slot-value-using-class`. This optimization can be switched off using the `:optimize-slot-access nil` class option.

To add valid initialization arguments for the class, use the class option `:extra-initargs`. The argument passed via this option is evaluated, and should return a list of extra initialization arguments for the class. `make-instance` will treat these as valid when checking its arguments.

Compatibility When a class is redefined, its extra initargs are always reset.
Note In early versions of LispWorks 4.3, extra initargs were not reset when a class was redefined without specifying extra initargs.

Example This session illustrates the effects of the `:optimize-slot-access` class option. When true, slot access is more efficient but note that `slot-value-using-class` is not called.

```

CL-USER 26 > (compile '(defclass foo ()
                         ((a :type fixnum
                            :initarg :a
                            :reader foo-a))))
NIL

CL-USER 27 > (compile '(defclass bar ()
                         ((a :type fixnum
                            :initarg :a
                            :reader bar-a))
                         (:optimize-slot-access nil)))
NIL

CL-USER 28 > (setf *foo*
                  (make-instance 'foo :a 42)
                  *bar* (make-instance 'bar :a 99))
#<BAR 21D33D4C>

CL-USER 29 > (progn
                (time (dotimes (i 1000000)
                          (foo-a *foo*)))
                (time (dotimes (i 1000000)
                          (bar-a *bar*))))
Timing the evaluation of (DOTIMES (I 1000000) (FOO-A
*FOO*))

user time      =      0.328
system time    =      0.015
Elapsed time   =  0:00:00
Allocation     = 2280 bytes standard / 11002882 bytes
conses
0 Page faults
Timing the evaluation of (DOTIMES (I 1000000) (BAR-A
*BAR*))

user time      =      0.406
system time    =      0.015
Elapsed time   =  0:00:00
Allocation     = 4304 bytes standard / 11004521 bytes
conses
0 Page faults
NIL

CL-USER 30 > (trace
                 (clos:slot-value-using-class
                  :when
                  (and (member (first *traced-arglist*)

```

```

(list (find-class 'foo)
      (find-class 'bar)))
(eq (third *traced-arglist*) 'a))))
(CLOS:SLOT-VALUE-USING-CLASS)

CL-USER 31 > (foo-a *foo*)
42

CL-USER 32 > (bar-a *bar*)
0 CLOS:SLOT-VALUE-USING-CLASS > ...
  >> CLASS : #<STANDARD-CLASS BAR 214897F4>
  >> CLOS::OBJECT : #<BAR 2148820C>
  >> CLOS::SLOT-NAME : A
0 CLOS:SLOT-VALUE-USING-CLASS < ...
  << VALUE-0 : 99
99

```

This session illustrates the `:extra-initargs` class option:

```

CL-USER 46 > (defclass a () ()
                  (:extra-initargs '(:a-initarg)))
#<STANDARD-CLASS A 21C2E4FC>

CL-USER 47 > (defclass b (a) ()
                  (:extra-initargs '(:b-initarg)))
#<STANDARD-CLASS B 2068573C>

CL-USER 48 > (defclass c (a) ())
#<STANDARD-CLASS C 22829D44>

CL-USER 49 > (make-instance 'b :a-initarg "A" :b-
initarg "B")
#<B 2068BCE4>

CL-USER 50 > (make-instance 'c :a-initarg "A" :b-
initarg "B")
Error: MAKE-INSTANCE is called with unknown keyword :B-
INITARG among the arguments (C :A-INITARG "A" :B-
INITARG "B") which is not one of (:A-INITARG).
1 (continue) Ignore the keyword :B-INITARG
2 (abort) Return to level 0.
3 Return to top loop level 0.

Type :b for backtrace, :c <option number> to proceed,
or :? for other options

CL-USER 51 : 1 >

```

See also [process-a-class-option](#)
[process-a-slot-option](#)

defpackage *Macro*

Summary	Remains as defined in Common Lisp, but see *handle-existing-defpackage* for an extension.
Package	<code>common-lisp</code>
Signature	<code>defpackage <i>defined-package-name</i> [[<i>option</i>]] => <i>package</i></code>
Arguments	<i>defined-package-name</i> A string. <i>option</i> Keyword options. <i>add-use-defaults</i> A keyword
Values	<i>package</i> A package.
Description	The macro <code>defpackage</code> is as defined in the ANSI standard, with the inclusion of the <code>:add-use-defaults</code> keyword. However, the standard explicitly declines to define what <code>defpackage</code> does if <i>defined-package-name</i> already exists and is in a state that differs from that described by the <code>defpackage</code> form. Therefore an extension has been written that allows you to select between alternative behaviors. See *handle-existing-defpackage* for full details. One non-standard <i>option</i> is supported. <code>:add-use-defaults</code> , with a true argument, causes the package <i>defined-package-name</i> to inherit from the following packages (as well as any explicitly specified by the <code>:use</code> option):
	<ul style="list-style-type: none">• <code>common-lisp</code>• <code>lispworks</code>

- `harlequin-common-lisp`

Example `(defpackage "MY-PACKAGE" (:use "CAPI")
 (:add-use-defaults t))

 (package-use-list "MY-PACKAGE")
 =>
 (#<PACKAGE COMMON-LISP> #<PACKAGE LISPWORKS> #<PACKAGE HARLEQUIN-COMMON-LISP> #<PACKAGE CAPI>)`

See also `*handle-existing-defpackage*`

describe	<i>Function</i>				
Summary	Remains as defined in ANSI Common Lisp. Additionally, you can control the depth at which slots inside arrays, structures and conses are described.				
Package	<code>common-lisp</code>				
Signature	<code>describe <i>object</i> &optional <i>stream</i> => <no-values></code>				
Arguments	<table border="0"> <tr> <td><i>object</i></td> <td>An object.</td> </tr> <tr> <td><i>stream</i></td> <td>An output stream designator.</td> </tr> </table>	<i>object</i>	An object.	<i>stream</i>	An output stream designator.
<i>object</i>	An object.				
<i>stream</i>	An output stream designator.				
Description	<p>The function <code>describe</code> displays information about the object <i>object</i> to the stream indicated by <i>stream</i>, as specified in ANSI Common Lisp.</p> <p>Arrays, structures and conses are <code>described</code> recursively up to the depth given in the value of the variable <code>*describe-level*</code>. Beyond that depth, objects are simply printed.</p>				
See also	<code>*describe-length*</code> <code>*describe-level*</code> <code>*describe-print-length*</code> <code>*describe-print-level*</code>				

directory	<i>Function</i>
Summary	Determines which files on the system have names matching a given pathname.
Package	<code>common-lisp</code>
Signature	<code>directory pathname &key test directories flat-file-namestring link-transparency non-existent-link-destinations => pathnames</code>
Arguments	<p><i>pathname</i> A pathname, string, or file-stream.</p> <p><i>test</i> Filtering test (only pathnames matching the test are collected).</p> <p><i>directories</i> A boolean controlling whether non-matching directories are included in the result.</p> <p><i>flat-file-namestring</i> A generalized boolean.</p> <p><i>link-transparency</i> If <code>nil</code>, then symbolic links are not followed. This means that returned names are not necessarily truenames, but has the useful feature that the <code>pathname-directory</code> of each pathname returned is the directory supplied as argument.</p> <p>The default value of <i>link-transparency</i> is given by the special variable, <code>*directory-link-transparency*</code>, which has initial value <code>t</code> on UNIX/Linux/Mac OS X. By setting this variable to <code>nil</code>, you can get the old behavior of <code>directory</code>. On Windows, where the file system does not normally support symbolic links, this variable is initially <code>nil</code>.</p>

non-existent-link-destinations

If this is `non-nil`, then the pathname pointed to by a symbolic link appears in the output whether or not this file actually exists. If `:link-transparency` is `non-nil` and `:non-existent-link-destinations` is `nil` (this is the default on UNIX/Linux/Mac OS X), then symbolic links to nonexistent files do not appear.

The default value is `nil`.

Values	<i>pathnames</i>	A list of physical pathnames.
Description	<p><code>directory</code> collects all the pathnames matching the given pathname.</p> <p><code>directory</code> returns truenames, conforming to the ANSI specification for Common Lisp. Some programs may depend on the old behavior, however (and <code>directory</code> is slower if it has to find the truename for every file in the directory), and so two keyword arguments are available so that the old behavior can still be used: <code>link-transparency</code> and <code>non-existent-link-destinations</code>.</p> <p>Because truenames are now returned, the entries <code>.</code> and <code>..</code> no longer show up in the output of <code>directory</code>. This means, for instance, that</p>	<pre>(directory #P"/usr/users/")</pre> <p>does not include <code>#P"/usr"</code>, which is the truename of <code>#P"/usr/users/.."</code></p>

The specification is unclear as to the appropriate behavior of `directory` in the presence of links to non-existent files or directories. For example, if the directory contains `foo`, which is a symbolic link to `bar`, and there is no file named `bar`, should `bar` show up in the directory listing? A keyword

argument has been added which lets you control this behavior.

`directory` returns a single pathname if called with a non-wild (fully-specified) *pathname*. LispWorks `truenames` are fully-specified, so this affects recursive calls to `directory`.

`directories`, if non-`nil`, causes paths of directories that are sub-directories of the directory of the argument *pathname* to be included in the result, even if they do not match *pathname* in the name, type or version components. The default value of `directories` is `nil`.

When `flat-file-namestring` is non-`nil`, `directory` matches the `file-namestring` of *pathname* as a flat string, rather than a pathname name and pathname type. The default value of `flat-file-namestring` is `nil`.

Note: The Search files tool in the LispWorks IDE uses this option when the **Match flat file-namestring** option is selected. See the *Common LispWorks User Guide* for more information about the Search Files tool.

Note: File names containing the character `*` cannot be handled by LispWorks. This is because LispWorks uses `*` as a wildcard, so there can be confusion if a file name containing `*` is created, for example in the *pathnames* returned by `directory`.

Compatibility Note	The <code>:check-for-subs</code> argument, implemented in LispWorks 4.0.1 and previous versions, has been removed. This argument controlled whether directories in the result have null name components. This option is no longer valid since ANSI Common Lisp specifies that <code>directory</code> returns truenames.
--------------------	---

Example

```
CL-USER 16 > (pprint (directory "**.*"))

(#P"C:/Program Files/LispWorks/readme-4450.txt"
 #P"C:/Program Files/LispWorks/Msvcrt.dll"
 #P"C:/Program Files/LispWorks/LW4450.isu"
 #P"C:/Program Files/LispWorks/lispworks-4450.exe"
 #P"C:/Program Files/LispWorks/license-4450.txt"
 #P"C:/Program Files/LispWorks/lib/")
```

This session illustrates the effect of the *directories* argument:

```
CL-USER 5 > (pprint (directory "/tmp/t*"))

(#P"/tmp/test.lisp" #P"/tmp/test2/" #P"/tmp/test1/")

CL-USER 6 > (pprint (directory "/tmp/t*" :directories
t))

(#P"/tmp/patches/"
 #P"/tmp/test.lisp"
 #P"/tmp/test2/"
 #P"/tmp/opengl/"
 #P"/tmp/test1/"
 #P"/tmp/mnt/")
```

This example illustrates directory returning a single pathname in its result when given a full-specified pathname:

```
CL-USER 1 > (directory
  (make-pathname :host "H"
    :device :unspecific
    :directory (list :absolute
      "tmp")
    :name :unspecific
    :type :unspecific
    :version :unspecific))
  (#P"H:/tmp/")
```

The next two examples illustrate the effect of *flat-file-namestring*. Suppose the directory *dir* contains files *interp.exe* and *file.lisp*.

This call matches *interp.exe*, where the name *interp* ends with *p*, but does not match *file.lisp*, where the name *file* ends with *e*:

```
(directory "dir/*p")
```

The next call matches `file.lisp`, where the namestring `file.lisp` ends with `p`, but does not match `interp.exe`, where the namestring `interp.exe` ends with `e`:

```
(directory "dir/*p" :flat-file-namestring t)
```

See also `truename`

	<i>Function</i>
disassemble	
Summary	Prints the machine code of a compiled function.
Package	<code>common-lisp</code>
Signature	<code>disassemble name-or-function => nil</code>
Arguments	<code>name-or-function</code> Either a function object, a lambda expression or a symbol with a function definition.
Description	This function prints the machine code of a compiled function, to <code>*standard-output*</code> . On UNIX and Mac OS X, the number of instructions in the disassembly is also printed, at the end. If the function denoted by <code>name-or-function</code> is not compiled then it is first compiled using the function <code>compile</code> . This happens if <code>name-or-function</code> is a lambda expression or a symbol naming an interpreted function. An error is signalled if <code>name-or-function</code> is not suitable.
Examples	<pre>(disassemble #'(lambda (x) (progn x))) (disassemble 'cons) (disassemble #'map)</pre>
Notes	The output from <code>disassemble</code> lacks useful information such as local and lexical variable names and symbol names. The representation of integers or characters or Lisp objects in gen-

eral is not easily readable without detailed knowledge of the internals of the Lisp system and the host machine instruction set.

See also

`compile`
`compile-file`

documentation	<i>Generic Function</i>
----------------------	-------------------------

Summary	Returns the documentation string if available.
---------	--

Package	<code>common-lisp</code>
---------	--------------------------

Signature	<code>documentation x doc-type => documentation</code> <code>(setf documentation) new-value x doc-type => new-value</code>
-----------	---

Description	The generic function documentation operates as specified in the ANSI Common Lisp standard. Additional methods with signatures:
-------------	--

```
documentation (dspec t) (doc-type (eql 'dspec:dspec))
(setf documentation) new-value (dspec t) (doc-type (eql
'dspec:dspec))
```

are provided.

This method allows finding or setting the documentation string when you know the dspec. See the *LispWorks User Guide* for an introduction to dspecs.

dspec must be a dspec, but it can be non-canonical. This method canonicalizes *dspec* and then calls `documentation` with the name as the first argument and the appropriate dspec class name as the second, thereby calling a standard `documentation` method.

If you define your own type of definitions (`def-saved-value` for example) with `define-dspec-class` you can add methods on documentation for your dspec class:

```
(documentation (dspec t) (doc-type (eql 'def-saved-value)))
```

This allows LispWorks IDE commands such as **Expression > Documentation** to display the documentation.

double-float	<i>Type</i>
---------------------	-------------

Summary	A subtype of <code>float</code> .
---------	-----------------------------------

Package	<code>common-lisp</code>
---------	--------------------------

Signature	<code>double-float</code>
-----------	---------------------------

Description	<code>double-float</code> is disjoint from <code>short-float</code> and <code>single-float</code> in all LispWorks implementations in version 5.0 and later.
-------------	--

Compatibility Note	In LispWorks 4.4 and previous on Windows and Linux platforms, all floats are of type <code>double-float</code> . However, there are distinct specialized array types (<code>array single-float</code>), with single precision, and (<code>array double-float</code>), with double precision.
--------------------	--

See also	<code>long-float</code> <code>parse-float</code> <code>short-float</code> <code>single-float</code>
----------	--

features	<i>Variable</i>
-------------------	-----------------

Summary	The features list.
---------	--------------------

Package	<code>common-lisp</code>
Initial Value	A list containing <code>:lispworks</code> . The actual value varies depending on the platform.
Description	The following features can be used to distinguish between platforms, or characteristics of the platform or of the Lisp-Works implementation.
<code>:solaris2</code>	Solaris2
<code>:hp-ux</code>	HP-UX
<code>:svr4</code>	System 5 Release 4 machine (for example Solaris2)
<code>:linux</code>	Linux
<code>:darwin</code>	The variant of FreeBSD underlying Mac OS X.
<code>:unix</code>	Unix, including all of the above.
<code>:mswindows</code>	Microsoft Windows, including 32-bit and 64-bit.
<code>:lispworks-64bit</code>	64-bit LispWorks.
<code>:lispworks-32bit</code>	32-bit LispWorks.
<code>:x86</code>	All images that run on the x86 architecture have this feature. This includes Intel Macintosh, FreeBSD, Linux (32-bit) and Windows (32-bit). Note: 64-bit LispWorks does not have this feature.
<code>:amd64, :x86-64, :x64</code>	

Images that run on the amd64/x86_64/x64 architecture have each of these feautures. This includes Linux (64-bit) and Windows (64-bit).

- `:sparc` Images that run on SPARC architecture.
- `:powerpc` Images that run on PowerPC architecture.
- `:hppa` Images that run in HP PA-RISC architecture.
- `:little-endian` The compiler targets a little endian machine, for instance x86.

Code can distinguish the twelve current LispWorks implementations like this:

```
#+(and :mswindows :lispworks-32bit)
" LispWorks (32-bit) for Windows"
#+(and :mswindows :lispworks-64bit)
" LispWorks (64-bit) for Windows"
#+(and :linux :lispworks-32bit)
" LispWorks (32-bit) for Linux"
#+(and :linux :lispworks-64bit)
" LispWorks (64-bit) for Linux"
#+freebsd
" LispWorks for FreeBSD"
#+(and :darwin :x86)
" LispWorks (32-bit) for Macintosh (running on Intel)"
#+(and :darwin :powerpc :lispworks-32bit)
" LispWorks (32-bit) for Macintosh (running on PowerPC)"
#+(and :darwin :x86-64 :lispworks-64bit)
" LispWorks (64-bit) for Macintosh (running on Intel)"
#+(and :darwin :powerpc :lispworks-64bit)
" LispWorks (64-bit) for Macintosh (running on PowerPC)"
#+(and :sparc :lispworks-32bit)
" LispWorks (32-bit) for Solaris"
#+(and :sparc :lispworks-64bit)
" LispWorks (64-bit) for Solaris"
#+:hppa
" LispWorks for HP PA"
```

The following features can be used to distinguish between versions of LispWorks:

- `:lispworks4` All major version 4 releases.

`:lispworks4.4` Release 4.4.x

`:lispworks5` All major version 5 releases.

`:lispworks5.0` Release 5.0.x

`:lispworks5.1` Release 5.1.x

Every LispWorks 5 image has exactly one of the features

`:lispworks-32bit` and `:lispworks-64bit`.

For `:sparc`, `:powerpc` and `:mswindows`, there are two LispWorks architectures: 32-bit and 64-bit, which can be distinguished by `:lispworks-32bit` or `:lispworks-64bit`.

The following features are present in LispWorks with the meanings defined for ANSI CL:

`:ansi-cl`

`:common-lisp`

`:ieee-floating-point`

Note that sometimes it is necessary to write code that examines `*features*` at load time or run time. For example this is true when you put platform-dependent code in fasl files that are shared between multiple platforms.

For a LispWorks image with the CAPI loaded, `:capi` will appear on `*features*`.

Note: LispWorks (32-bit) for Macintosh supports the native Mac OS X Cocoa-based GUI and the X11/Motif GUI. If you need to test for which of these libraries is loaded, check for the features `:cocoa` and `:x11-motif` respectively.

input-stream-p

Generic Function

Summary A generic function that determines if an object is an input stream.

Package `common-lisp`

Signature	<code>input-stream-p stream => result</code>	
Arguments	<code>stream</code>	A stream.
Values	<code>result</code>	A generalized boolean.
Description	The predicate <code>input-stream-p</code> is implemented as a generic function. The default method returns <code>t</code> if <code>stream</code> is an input stream. If the user wants to implement a stream with no inherent directionality (and thus does not include <code>fundamental-input-stream</code> or <code>fundamental-output-stream</code>) but for which the directionality depends on the instance, then a method should be provided for <code>input-stream-p</code> .	
See also	<code>fundamental-input-stream</code> <code>output-stream-p</code>	

interactive-stream-p *Function*

Summary	A generic function that determines if an object is an interactive stream.	
Package	<code>cl</code>	
Signature	<code>interactive-stream-p stream -> bool</code>	
Arguments	<code>stream</code>	A stream.
Values	<code>bool</code>	A generalized boolean.
Description	The predicate <code>interactive-stream-p</code> is implemented as a generic function. The <code>fundamental-stream</code> class provides a default method that returns <code>nil</code> .	
See also	<code>input-stream-p</code> <code>output-stream-p</code>	

		<i>Function</i>
Summary	Searches for and loads the definition of a logical host, if not already defined.	
Package	<code>cl</code>	
Signature	<code>load-logical-pathname-translations host => just-loaded</code>	
Arguments	<code>host</code>	A logical host, expressed as a string.
Values	<code>just-loaded</code>	A generalized boolean
Description		This function loads the translations for <code>host</code> by loading the file <code>host.lisp</code> from the LispWorks directory <code>translations</code> .
Example		<code>(load-logical-pathname-translations "EDITOR-SRC")</code>
		<i>Type</i>
long-float		
Summary	A subtype of <code>float</code> .	
Package	<code>common-lisp</code>	
Signature	<code>long-float</code>	
Description		<code>long-float</code> is the same type as <code>double-float</code> in LispWorks, on all platforms.
See also		<code>double-float</code> <code>parse-float</code> <code>short-float</code> <code>single-float</code>

	<i>Function</i>
Summary	Identifies the physical location of the computer.
Package	<code>common-lisp</code>
Signature	<code>long-site-name => description</code> <code>(setf long-site-name) description => description</code>
Arguments	<i>description</i> A string or <code>nil</code> .
Description	The function <code>long-site-name</code> returns a string identifying the physical location of the computer. This should be set using <code>(setf long-site-name)</code> when you configure your Lisp-Works image.
See also	<code>short-site-name</code>

	<i>Macro</i>
Summary	A macro that performs iteration.
Package	<code>cl</code>
Signature	<code>loop {for as} var [type-spec]</code> <code> being {the each}{records record}</code> <code> {in of} query-expression => result</code>
Arguments	<i>var</i> A variable. <i>query-expression</i> An SQL query-statement
Values	<i>result</i> An object.
Description	The Common Lisp <code>loop</code> macro has been extended with a clause for iterating over query results. This extension is available only when the SQL interface has been loaded. See Chap-

ter 14, “The SQL Package”. For a full description of the rest of the Common Lisp `loop` facility, see the Common Lisp Hyperspec.

Each iteration of the loop assigns the next record of the table to the variable `var`. The record is represented in Lisp as a list. Destructuring can be used in `var` to bind variables to specific attributes of the records resulting from *query-expression*. In conjunction with the panoply of existing clauses available from the `loop` macro, the new iteration clause provides an integrated report generation facility.

Example	This extended <code>loop</code> example, on each record returned as a result of the query, binds <code>name</code> , finds the salary (if any) from an associated hash-table, and for salaries greater than 20000: increments a count, accumulates the salary, and prints the details. Finally, the average salary is printed.
---------	--

```
(loop
  for (name) being each record in
  [select [ename] :from [emp]]
  as salary = (gethash name *salary-table*)
  initially (format t "~&~20A~10D" 'name 'salary)
  when (and salary (> salary 20000))
    count salary into salaries
    and sum salary into total
    and do (format t "~&~20A~10D" name salary)
  else
    do (format t "~&~20A~10A" name "N/A")
  finally
    (format t "~2&Av Salary: ~10D" (/ total salaries)))
```

See also

`do-query`
`map-query`
`query`
`select`

make-array

Function

Summary

Creates and returns a new array which, in addition to the standard functionality, can be a weak array or statically allocated.

Package

`common-lisp`

Signature

`make-array dimensions &key element-type initial-element initial-contents adjustable fill-pointer displaced-to displaced-index-offset weak allocation => new-array`

Arguments

`weak` A generalized boolean.

`allocation` A fixnum, or one of `nil`, `:new`, `:static`, `:old`, or `:long-lived`.

Description

The standard definition of `make-array` is extended to accept the keyword arguments `:weak` and `:allocation`.

If `weak` is `nil`, then `displaced-to` must be `nil` and if `element-type` is supplied it must have `upgraded-array-element-type t`, otherwise an error is signalled. That is, you cannot make a weak array which is displaced or has `array-element-type` other than `t`. When `weak` is `nil`, it makes `new-array` weak.

If `weak` is `nil`, then `make-array` behaves in the standard way, and `new-array` is not weak. The value `weak` defaults to `nil`.

See `set-array-weak` for a description of weak arrays.

The possible values for `allocation` have the following meanings:

`:new` Allocate the array normally.

<code>nil</code>	Same meaning as <code>:new</code> . This is the default value.
<code>:static</code>	Allocate the array in a static segment.
<code>:long-lived</code>	Allocate the array assuming it is going to be long-lived.
<code>:old</code>	Same meaning as <code>:long-lived</code>
A fixnum <i>n</i>	Allocate the array in generation <i>n</i> .
	Arrays (including strings) that are passed by address to foreign functions must be static, and so must should be created with <code>:allocation :static</code> .
	Allocation with <code>:old</code> or <code>:long-lived</code> is useful when you know that the array will be long-lived, because your program will avoid the overhead of promoting it to the older generations.

See also [set-array-weak](#)

make-hash-table *Function*

Summary	Creates and returns a new hash table which, in addition to the standard functionality, can have a user-defined test, a user-defined hash function, and be a weak hash table.	
Package	<code>common-lisp</code>	
Signature	<code>make-hash-table &key test size rehash-size rehash-threshold hash-function weak-kind => hash-table</code>	
Arguments	<code>test</code>	A designator for a function of two arguments, which returns <code>t</code> if they should be regarded as the same and <code>nil</code> otherwise.
	<code>hash-function</code>	A designator for a function of one argument, which returns a hash value.

weak-kind	One of <code>:value</code> , <code>t</code> , <code>:key</code> , <code>:both</code> , <code>:one</code> , <code>:either</code> , <code>nil</code> . The default is <code>nil</code> .
Description	<p>The standard definition of <code>make-hash-table</code> is extended such that <i>test</i> can be any suitable user-defined function, except that it must not call <code>process-wait</code> or similar <code>mp</code> package functions which suspend the current process. If <i>test</i> is not one of the standard test functions (<code>eq</code>, <code>eq1</code>, <code>equal</code> and <code>equalp</code>), and if <i>hash-function</i> is not supplied, then the hash value is the same as would be used if <i>test</i> were <code>equalp</code>.</p> <p><i>hash-function</i> may be supplied only if <i>test</i> is not one of the standard test functions. It takes a hash key as its argument and returns a hash value to use for hashing.</p> <p>If <i>weak-kind</i> is non-<code>nil</code>, it makes <i>hash-table</i> weak. Its semantics are the same as the second argument of <code>set-hash-table-weak</code>, that is:</p> <pre>(make-hash-table :weak-kind <i>weak-kind</i> <other-args>)</pre> <p>is equivalent to</p> <pre>(let ((ht (make-hash-table <other-args>))) (set-hash-table-weak ht <i>weak-kind</i>) ht)</pre>
See also	<code>set-hash-table-weak</code>

	<i>Generic Function</i>		
Summary	Creates and returns a new instance of a class.		
Package	<code>common-lisp</code>		
Signature	<code>make-instance <i>class</i> &rest <i>initargs</i> &key &allow-other-keys => <i>instance</i></code>		
Arguments	<table> <tr> <td><code><i>class</i></code></td><td>A class, or a symbol that names a class.</td></tr> </table>	<code><i>class</i></code>	A class, or a symbol that names a class.
<code><i>class</i></code>	A class, or a symbol that names a class.		

	<i>initargs</i>	An initialization argument list.
Values	<i>instance</i>	A fresh instance of class <i>class</i> .
Description	<p><code>make-instance</code> behaves as specified in ANSI Common Lisp. In particular it checks the initialization arguments as calculated by <code>compute-class-potential-initargs</code>. This check can be suppressed by passing <code>:allow-other-keys t</code>. In addition, LispWorks provides global control over the initarg checking via <code>set-make-instance-argument-checking</code> and per-class control via <code>class-extra-initargs</code>.</p> <p>Note: in a delivered image, <code>make-instance</code> does not check the initialization arguments.</p>	
Compatibility	In LispWorks 4.2 and previous versions, <code>make-instance</code> does not check the initargs. If your code contains invalid initargs, you could use one of the techniques mentioned above to resolve it.	
Note		
See also	<code>class-extra-initargs</code> <code>compute-class-potential-initargs</code> <code>set-make-instance-argument-checking</code>	

make-sequence *Function*

Summary	Extends the standard <code>make-sequence</code> function allowing it to take any type specifier.	
Package	<code>common-lisp</code>	
Signature	<code>make-sequence result-type size &key initial-element => sequence</code>	
Arguments	<i>result-type</i>	A type specifier.
	<i>size</i>	A non-negative integer.

	<i>initial-element</i>	An object.
Values	<i>sequence</i>	A sequence.
Description		The <code>make-sequence</code> function has been extended to take any Common Lisp type. The <i>sequence</i> will be of type <i>result-type</i> unless this is not possible, in which case a <code>type-error</code> is signalled.
See also		<code>concatenate</code> <code>map</code> <code>merge</code>

map *Function*

Summary	Redefines the standard <code>map</code> function allowing it to take any type specifier.	
Package	<code>common-lisp</code>	
Signature	<code>map result-type function &rest sequences => result</code>	
Arguments	<i>result-type</i>	A sequence type specifier or <code>nil</code> .
	<i>function</i>	A function designator.
	<i>sequence</i>	A sequence.
Values	<i>result</i>	A sequence.
Description	The <code>map</code> function has been extended to take any Common Lisp type. The <i>result</i> will be of type <i>result-type</i> unless this is not possible, in which case a <code>type-error</code> is signalled.	
See also	<code>concatenate</code> <code>make-sequence</code> <code>merge</code>	

		<i>Function</i>
merge		
Summary	Redefines the standard <code>merge</code> function allowing it to take any type specifier.	
Package	<code>common-lisp</code>	
Signature	<code>merge result-type sequence1 sequence2 predicate &key key => sequence</code>	
Arguments	<p><i>result-type</i> A type specifier.</p> <p><i>sequence1</i> A sequence.</p> <p><i>sequence2</i> A sequence.</p> <p><i>predicate</i> A designator for a function.</p> <p><i>key</i> A designator for a function or <code>nil</code>.</p>	
Values	<i>sequence</i> A sequence.	
Description	The <code>merge</code> function has been extended to take any Common Lisp type. The <i>sequence</i> will be of type <i>result-type</i> unless this is not possible, in which case a <code>type-error</code> is signalled.	
See also	<code>concatenate</code> <code>make-sequence</code> <code>map</code>	

		<i>Function</i>
open		
Summary	Creates, opens, and returns a file stream that is connected to a specified file.	
Package	<code>common-lisp</code>	
Signature	<code>open filespec &key direction element-type external-format if-exists if-does-not-exist => stream</code>	

Arguments	<i>filespec</i>	A file designator.
	<i>direction</i>	If <i>direction</i> is <code>:probe</code> , <i>external-format</i> is ignored. The element type and external format of the returned stream are undefined.
	<i>element-type</i>	By default, the value of <code>*default-character-element-type*</code> (the ANSI standard default is <code>character</code>).
	<i>external-format</i>	An external file format designator. By default, this is <code>:default</code> .
	<i>if-exists</i>	What to do if the file stream already exists. The possible values for this are as in the ANSI standard.
	<i>if-does-not-exist</i>	What to do if the file stream does not already exist. The possible values for this are as in the ANSI standard.
Values	<i>stream</i>	A file stream, or <code>nil</code> .
Description	<p>If <i>external-format</i> has a name which is not <code>:default</code> and the parameters include <code>:eol-style</code>, it is used as is.</p> <p>Otherwise, the system decides which external format to use via <code>guess-external-format</code>. By default, this finds a match based on the filename; or (if that fails), looks in the EMACS-style <code>(-*-)</code> attribute line for an option called <code>encoding</code> or <code>external-format</code>; or (if that fails), chooses from among likely encodings by analyzing the bytes near the start of the file. By default, it then also analyses the start of the file for byte patterns indicating the end-of-line style, and uses a default end-of-line style if no such pattern is found. This behavior is configurable.</p> <p>After the external-format has been determined, it is verified using <code>valid-external-format-p</code>; and an error is signalled if this check fails.</p>	

If `open` gets `:default` as its *element-type* arg, it chooses the type on the basis of the external format. If `open` gets an *element-type* other than `:default` and the direction is `:input` or `:io`, the argument must be a supertype of the type of characters produced by the external format; if the direction is `:output` or `:io`, it must be a subtype of the type of characters accepted by the external format; if it does not satisfy these requirements, an error is signalled.

Standard stream input and output functions for character and binary data generally work in the obvious way on a `file-stream` with *element-type* `base-char`, `(unsigned-byte 8)` or `(signed-byte 8)`. For example, `read-sequence` can be called with a string buffer and a binary `file-stream`: the character data is constructed from the input as if by `code-char`. Similarly `write-sequence` can be called with a string buffer and a binary `file-stream`: the output is converted from the character data as if by `char-code`. Also, 8-bit binary data can be read from and written to a `base-char file-stream`.

All standard stream I/O functions except for `write-byte` and `read-byte` have this flexibility.

See also

- `*default-character-element-type*`
- `guess-external-format`
- `set-file-dates`
- `valid-external-format-p`

open-stream-p

Generic Function

Summary A generic function that determines if a stream has been closed.

Package `common-lisp`

Signature `open-stream-p stream => result`

Arguments	<i>stream</i>	A stream.
Values	<i>result</i>	A generalized boolean.
Description		The function <code>open-stream-p</code> is generic. The default method provided by the class <code>fundamental-stream</code> returns <code>t</code> if <code>close</code> has not been called on the stream.
See also		<code>close</code> <code>fundamental-stream</code>

		<i>Generic Function</i>
Summary		A generic function that determines if an object is an output stream.
Package		<code>common-lisp</code>
Signature		<code>output-stream-p <i>stream</i> => <i>result</i></code>
Arguments	<i>stream</i>	A stream.
Values	<i>result</i>	A generalized boolean.
Description		The predicate <code>output-stream-p</code> is implemented as a generic function. The default method returns <code>t</code> if <code>stream</code> is an output stream. If the user wants to implement a stream with no inherent directionality (and thus does not include <code>fundamental-input-stream</code> or <code>fundamental-output-stream</code>) but for which the directionality depends on the instance, then a method should be provided for <code>output-stream-p</code> .
See also		<code>fundamental-output-stream</code> <code>input-stream-p</code>

proclaim	<i>Function</i>
Summary	Established a specified declaration in the global environment.
Package	<code>common-lisp</code>
Signature	<code>proclaim <i>declaration-list</i> => nil</code>
Arguments	<code>declaration-list</code> A list of declaration forms to be put into immediate and pervasive effect.
Values	Returns <code>nil</code> .
Description	Unlike <code>declare</code> , <code>proclaim</code> is a function that parses the declarations in the list (usually a quoted list), and puts their semantics and advice into global effect. This can be useful when compiling a file for speedy execution, since a proclamation such as:
	<pre>(proclaim '(optimize (speed 3) (space 0) (debug 0)))</pre> means the rest of the file is compiled with these optimization levels in effect. Other ways of doing this are:
	<ul style="list-style-type: none"> • use the <code>:optimize</code> option in <code>defsystem</code> to establish default optimization qualities for every member of the system, when compiled via <code>compile-system</code>. • add appropriate <code>declare</code> declarations in every function in the file.
	Note: For a top-level call to <code>proclaim</code> or <code>declaim</code> , optimize declarations are omitted from the compiled binary file. This deviates from the ANSI Common Lisp Standard but is useful because you are unlikely to want to change settings outside of that file. To make the global settings, you can call a function which calls <code>proclaim</code> (so it is not a top-level call).
	See the <i>LispWorks User Guide</i> for a more extended description of the compiler optimize qualities.

Examples	<pre>(proclaim '(special *fred*)) (proclaim '(type single-float x y z)) (proclaim '(optimize (safety 0) (speed 3)))</pre>
Notes	<p>As <code>proclaim</code> involves parsing a list of lists of symbols and is intended to be used a few times per file, its implementation is not optimized for speed — it makes little sense to use it other than at top level.</p> <p>Remember to quote the argument list if it is a constant list. <code>(proclaim (special x))</code> attempts to call function <code>special</code>.</p> <p>Exercise caution if you declare or proclaim variables to be special without regard to the naming convention that surrounds their names with asterisks.</p>
See also	<code>compile</code> <code>compile-file</code> <code>declare</code> <code>declaim</code>

	<i>Macro</i>
Summary	Evaluates a restartable form in a special dynamic environment.
Package	<code>common-lisp</code>
Signature	<code>restart-case restartable-form {clause} => result*</code> <code>clause ::= (case-name lambda-list [[:interactive interactive-expression :report report-expression :test test-expression]] declaration* form*)</code>
Description	<p>The macro <code>restart-case</code> behaves as specified in the ANSI Common Lisp standard.</p> <p>In addition to that specification, <code>report-expression</code> may be a form whose <code>car</code> is <code>list</code>. Such a form is evaluated when the</p>

restart is set up and is expected to return a list of a format string and format arguments. When the restart is asked to report, this is done by calling `format` on the stream, the format string and the format arguments. This is more efficient than specifying an equivalent function, because no function object is created.

room	<i>Function</i>
Summary	Print information about the state of internal storage and its management.
Package	<code>common-lisp</code>
Signature	<code>room &optional x</code>
Arguments	<code>x</code> One of <code>nil</code> , <code>t</code> , or <code>:default</code> .
Values	<code>room</code> returns no values.
Description	This function provides statistics on the current state of the storage, including the amount of space currently allocated, and the amount available for allocation. As outlined in the Common Lisp Hyperspec, the <code>room</code> function takes an optional argument which controls the level of detail it produces. Given an argument of <code>nil</code> , a summary of the total allocation in the entire heap (in kilobytes) is produced. The “allocated” figure only represents the amount of space allocated in heap segments that are writable, as opposed to read-only segments that hold some of the system code such as the garbage collector itself. The free space figure covers all the free space in all segments.

When called without an argument, `room` additionally prints information on the distribution of space between the generations of the heap.

When called with argument τ , a breakdown of allocation in the individual segments of each generation is produced. Each segment is identified by its start address in memory. For each segment there is a free space threshold (the “minimum free space”)—when the available space in the segment falls below this value, the garbage collector takes action to attempt to free more space in this segment.

Two statistics about promotion are also reported on a per-segment basis: the number of sweeps that an object must survive in this generation before becoming eligible for promotion, and the total volume of objects that have survived for that long and are consequently awaiting promotion to the next generation. These statistics are not relevant for static segments, which are indicated as “static”.

`room` prints numbers in decimal format, except for the segment start addresses which it prints in hexadecimal format.

Examples

```

CL-USER 23 > (room nil)

Total Size 50752K, Allocated 42868K, Free 7522K

CL-USER 24 > (room)
Generation 0: Total Size 2778K, Allocated 519K, Free
2251K
Generation 1: Total Size 3958K, Allocated 2524K, Free
1413K
Generation 2: Total Size 24324K, Allocated 20730K,
Free 3572K
Generation 3: Total Size 19391K, Allocated 19266K,
Free 112K

Total Size 50752K, Allocated 43040K, Free 7349K

CL-USER 25 > (room t)
Generation 0: Total Size 2778K, Allocated 561K, Free
2208K
Segment 200877D8: Total Size 507K, Allocated
457K, Free 46K
minimum free space 64K,
Awaiting promotion = 1K, sweeps
before promotion =10
Segment 22F58548: Total Size 2270K, Allocated
104K, Free 2162K
minimum free space 0K,
Awaiting promotion = 0K, sweeps
before promotion =2
Generation 1: Total Size 3958K, Allocated 2524K, Free
1413K
Segment 21C08548: Total Size 1472K, Allocated
1423K, Free 44K
minimum free space 0K,
Awaiting promotion = 0K, sweeps
before promotion =4
Segment 22198548: Total Size 1088K, Allocated
778K, Free 305K
minimum free space 0K,
Awaiting promotion = 0K, sweeps
before promotion =4
Segment 20706770: Total Size 68K, Allocated
3K, Free 60K
minimum free space 3K,
Awaiting promotion = 0K, sweeps
before promotion =4
Segment 216D8548: Total Size 1088K, Allocated

```

```

213K, Free 870K
    minimum free space 0K,
    Awaiting promotion = 0K, sweeps
before promotion =4
    Segment 2004AFA8: Total Size 242K, Allocated
105K, Free 132K
    minimum free space 0K, static
Generation 2: Total Size 24324K, Allocated 20730K,
Free 3572K
    Segment 222A8548: Total Size 12992K, Allocated
9527K, Free 3460K
    minimum free space 0K,
    Awaiting promotion = 0K, sweeps
before promotion =4
    Segment 21D78548: Total Size 4224K, Allocated
4110K, Free 109K
    minimum free space 0K,
    Awaiting promotion = 0K, sweeps
before promotion =4
    Segment 21418548: Total Size 2816K, Allocated
2811K, Free 0K
    minimum free space 0K,
    Awaiting promotion = 0K, sweeps
before promotion =4
    Segment 217E8548: Total Size 4224K, Allocated
4218K, Free 1K
    minimum free space 0K,
    Awaiting promotion = 0K, sweeps
before promotion =4
    Segment 20DBDDC8: Total Size 68K, Allocated
63K, Free 0K
    minimum free space 117K,
    Awaiting promotion = 0K, sweeps
before promotion =4
    Generation 3: Total Size 19391K, Allocated 19266K,
Free 112K
    Segment 20106770: Total Size 6144K, Allocated
6139K, Free 0K
    minimum free space 3K,
    Awaiting promotion = 0K, sweeps
before promotion =10
    Segment 20DCEE40: Total Size 6437K, Allocated
6321K, Free 112K
    minimum free space 0K,
    Awaiting promotion = 0K, sweeps
before promotion =10
    Segment 207177E8: Total Size 6809K, Allocated

```

```
6805K, Free 0K
               minimum free space 0K,
               Awaiting promotion = 0K, sweeps
before promotion =10

Total Size 50752K, Allocated 43083K, Free 7307K
```

See also [find-object-size](#)
[room-values](#)
[total-allocation](#)

	Type
Summary	A subtype of float .
Package	common-lisp
Signature	short-float
Description	A short float is an immediate object. short-float is disjoint from double-float in all LispWorks implementations in version 5.0 and later. short-float is disjoint from single-float in all 32-bit LispWorks implementations, version 5.0 and later. In 64-bit LispWorks short-float is the same type as single-float .
Compatibility Note	In LispWorks 4.4 and previous on Windows and Linux platforms, short-float is the same type as double-float .
See also	double-float long-float parse-float single-float

	<i>Function</i>
Summary	Identifies the physical location of the computer.
Package	<code>common-lisp</code>
Signature	<code>short-site-name => description</code> <code>(setf short-site-name) description => description</code>
Arguments	<code>description</code> A string or <code>nil</code> .
Description	The function <code>short-site-name</code> returns a string briefly identifying the physical location of the computer. This should be set using <code>(setf short-site-name)</code> when you configure your LispWorks image.
See also	<code>long-site-name</code>

	<i>Type</i>
Summary	The simple base string type.
Package	<code>common-lisp</code>
Signature	<code>simple-base-string length</code>
Arguments	<code>length</code> The length of the string (or <code>*</code> , meaning any).
Description	The type of simple base strings.

	<i>Type</i>
Summary	A subtype of <code>float</code> .
Package	<code>common-lisp</code>

Signature	<code>single-float</code>
Description	A <code>single-float</code> is an immediate object in 64-bit LispWorks. A <code>single-float</code> is a boxed object in 32-bit LispWorks. <code>single-float</code> is disjoint from <code>double-float</code> in all LispWorks implementations, version 5.0 and later. <code>single-float</code> is disjoint from <code>short-float</code> in all 32-bit LispWorks implementations in version 5.0 and later. In 64-bit LispWorks <code>single-float</code> is the same type as <code>short-float</code> .
Compatibility	In LispWorks 4.4 and previous on Windows and Linux platforms, <code>single-float</code> is the same type as <code>double-float</code> .
Note	However, there are distinct specialized array types (<code>array single-float</code>), with single precision, and (<code>array double-float</code>), with double precision.
See also	<code>double-float</code> <code>long-float</code> <code>parse-float</code> <code>short-float</code>

software-type	<i>Function</i>
Summary	Identifies the Operating System.
Package	<code>common-lisp</code>
Signature	<code>software-type => description</code>
Values	<code>description</code> A string or <code>nil</code> .
Description	The function <code>software-type</code> returns a string representing a generic name of the Operating System, or <code>nil</code> if this cannot be determined.

On Microsoft Windows 98 and Millennium, `software-type` returns "Windows". On Windows 2000 and Windows XP, `software-type` returns "Windows NT". For more detail, use `software-version`.

See also `software-version`

	<i>Function</i>
software-version	
Summary	Identifies the version of the Operating System.
Package	<code>common-lisp</code>
Signature	<code>software-version => description</code>
Values	<code>description</code> A string or <code>nil</code> .
Description	The function <code>software-version</code> returns a string giving the version of the Operating System, or <code>nil</code> if this cannot be determined. On Microsoft Windows systems, <code>description</code> begins with the specific Operating System. This is "Windows 98", "Windows Millennium", "Windows 2000", "Windows XP", "Windows XP x64 Edition", "Windows 2003", "Windows Vista", "Windows Server \\"Longhorn\\"", or "Some Windows NT derivative". This is followed by the version numbers (Major.Minor), build number and optionally service pack.
Example	<pre>(software-version) => "Windows Vista: 6.0 (build 6000) " (software-version) => "Windows XP: 5.1 (build 2600) Service Pack 2"</pre>

```
(software-version)
=>
"Windows Millennium: 4.90 (build 3000)"
```

See also `software-type`

step *Macro*

Summary Steps through the evaluation of a form.

Package `common-lisp`

Signature `step form => result`

Arguments `form` A form to be stepped and evaluated.

Values `result` The values returned by *form*.

Description `step` evaluates a form and allows you to single-step through it. You can include a call to `step` inside a tricky definition to invoke the stepper every time the definition is used. `step` can also optionally step through macros.

The commands shown below are available. When certain stepper variables (as described below) are set, some of these commands are not relevant and are therefore not available. Use `:help` to get a list of the commands.

- `:s n` Step this form and all of its subforms (optional positive integer argument).
- `:st` Step this form without stepping its sub-forms.
- `:su` Step up out of this form without stepping its subforms.
- `:sr` Return a value to use for this form.
- `:sq` Quit from the current stepper level.

<code>:redo</code>	Redo one of the previous commands.
<code>:get</code>	Get an item from the history list and put it in a variable.
<code>:help</code>	List available commands.
<code>:use</code>	Replace one form with another form in previous command and redo it.
<code>:his</code>	List the commands history.

The optional integer argument *n* for `:s` means do `:s n` times.

Note: `step` is a Listener-based form stepper. LispWorks also offers a graphical source-code Stepper tool. See the *Common LispWorks User Guide* for details of that.

Examples The following examples illustrate some of these commands.

```

USER 12 > (step (+ 1 (* 2 3) 4))
(+ 1 (* 2 3) 4) -> :s
  1 -> :s
  1
  (* 2 3) -> :su
  6
  4 -> :s
  4
11
11

USER 13 > (defun foo (x y) (+ x y))
FOO

USER 14 > step (foo (+ 1 1) 2)
(FOO (+ 1 1) 2) -> :st
  (+ 1 1) -> :s
    1 -> :s
    1
    1 -> :s
    1
    2
    2 -> :s
    2
    4
    4

```

```
USER 15 > :redo (STEP (FOO # 2))
(FOO (+ 1 1) 2) -> :s
(+ 1 1) -> :s
  1 -> :s
    1
  2
  2 -> :s
  2
(+ X Y) -> :s
  X -> :s
    2
  Y -> :s
    2
  4
4
4
```

You can interact when an evaluated form returns, by setting the variable `*no-step-out*` to `nil`. The prompt changes as shown below:

```
USER 36 > step (cons 1 2)
(CONS 1 2) -> :s
  1 -> :s
  1 = 1 <- :sr 3
  2 -> :s
  2 = 2 <- :sr 4
(CONS 1 2) = (3 . 4) <- :s
(3 . 4)
```

To allow expansion of macros, set the variable `*step-macros*` to `t`.

To step through the function calls in compiled code, set the variable `hcl:*step-compiled*` to `t`.

If required, the stepper can print out the step level: set the variable `*print-step-level*` to `t`, as shown in this session:

```

USER 21 > (setq *print-step-level* t)
T
USER 22 > step (cons 1 2)
[1](cons 1 2) -> :s
[2]   1 -> :s      1
[2]   2 -> :s
      2
(1 . 2)
(1 . 2)

```

It is not advisable to try to step certain compiled functions, such as `car` and `format`. The variable `hcl:*step-filter*` contains a list of functions which should not be stepped. If you get deep stack overflows inside the stepper, you may need to add a function name to `hcl:*step-filter*`.

By default, the stepper uses the same printing environment as the rest of LispWorks (the same settings of the `*print-...` variables). To control the stepper printing environment independently, set the variable `hcl:*step-print-env*` to `t`.

The values of the variables `hcl:*step-print-...` are then used instead of the variables `*print-...`.

stream-element-type

Generic Function

Summary	Implements <code>stream-element-type</code> as a generic function.	
Package	<code>common-lisp</code>	
Signature	<code>stream-element-type stream => type</code>	
Arguments	<code>stream</code>	A stream.
Values	<code>type</code>	A type specifier.
Description	The function <code>stream-element-type</code> is implemented as a generic function. Depending on the stream, a method should	

be defined for this generic function that returns the element type of the stream.

Methods must be implemented for all subclasses of `buffered-stream`. Typically for character streams, the implementation can return the `array-element-type` of the buffer.

For the class `fundamental-character-stream` a default method is provided which returns `character`. A method should be defined for stream classes based on the `fundamental-binary-stream` class.

See also `buffered-stream`
 `fundamental-binary-stream`
 `fundamental-character-stream`

string	<i>Type</i>				
Summary	The string type.				
Package	<code>common-lisp</code>				
Signature	<code>string length element-type</code>				
Arguments	<table border="0"> <tr> <td><code>length</code></td> <td>The length of the string (or <code>*</code>, meaning any).</td> </tr> <tr> <td><code>element-type</code></td> <td>The type of string element. The default is the value of <code>*default-character-element-type*</code> rather than <code>*</code>.</td> </tr> </table>	<code>length</code>	The length of the string (or <code>*</code> , meaning any).	<code>element-type</code>	The type of string element. The default is the value of <code>*default-character-element-type*</code> rather than <code>*</code> .
<code>length</code>	The length of the string (or <code>*</code> , meaning any).				
<code>element-type</code>	The type of string element. The default is the value of <code>*default-character-element-type*</code> rather than <code>*</code> .				
Description	<p>The union of all string types as specified in the standard, but extended with an extra parameter: <code>element-type</code>.</p> <p><code>(string length element-type)</code> means all string types whose element type is a subtype of <code>element-type</code>. That is:</p> <pre>(string * base-char) = (vector base-char *) (string * lw:simple-char) = (or (vector base-char *) (vector lw:simple-char *))</pre>				

```
(string * character)      = (or (vector base-char *)
                                (vector lw:simple-char *)
                                (vector character *))
```

Example

```
CL-USER 235 > (lw:set-default-character-element-type
                  'base-char)
BASE-CHAR
CL-USER 236 > (concatenate 'string "f" "o" "o")
"foo"
CL-USER 237 > (type-of *)
SIMPLE-BASE-STRING
```

See also

```
*default-character-element-type*
set-default-character-element-type
```

time *Macro*

Summary

Determines the execution time of a form in the current environment.

Package

common-lisp

Signature

`time form => result`

Arguments

`form` A form to be evaluated

Values

`result` The result of the evaluation of the form.

Description

`time` can be used to determine execution times. The macro evaluates the form given to it as argument, and prints out some timing and size data: the user and system times (in seconds), the elapsed time (in hours, minutes and seconds), and the total amount of heap space allocated in executing the form (in bytes).

The timing and size data covers all stack groups, not just the one that invokes `time`. Note that `time` itself uses a small, constant amount of heap space.

Note: `time` measures all threads, so to test accurately for consistency in your code `code` you need to do:

```
(without-interrupts (time code))
```

This is particularly important when using the Common Lisp-Works IDE.

Examples

```
CL-USER 33 > (time (progn (format t "start")
                           (dotimes (a 100) nil)
                           (format t "end")))
Timing the evaluation of (PROGN (FORMAT T "start")
(DOTIMES (A 100) NIL) (FORMAT T "end"))
startend
User time      =          0.000
System time    =          0.000
Elapsed time   =          0.000
Allocation     = 5468 bytes
0 Page faults
Calls to %EVAL    1160
NIL
```

See also

[extended-time](#)

trace

Macro

Summary

Invoke the Common Lisp tracing facility on the named functions.

Package

`common-lisp`

Signature

`trace {function-name|tracing-desc}* => trace-result`

`tracing-desc ::= (dspec {keyword form}*)`

`dspec ::= function-name |
(method generic-function-name [qualifier] (class*))`

```

keyword ::= :after | :allocation | :before | :backtrace |
           :eval-after | :eval-before | :break |
           :break-on-exit | :entrycond | :exitcond |
           :inside | :process | :trace-output | :step |
           :when

qualifier ::= :after | :before | :around

function-name ::= symbol | (setf symbol)

```

Arguments	function-name	A symbol whose symbol-function is to be traced, or a setf function name. Functions, macros and generic functions may be specified this way.
	dspec	Specifies the functional definition which is to be traced. This either has the same form as above, or specifies a method by the name of its generic function and by a list of classes to specialize the arguments to the method. In this latter case the list of classes must correspond exactly to the classes of the specialized parameters of an existing method, and then only this method is traced (as opposed to the corresponding generic function).
	tracing-desc	Specifies the functional definition which is to be traced and specifies any additional options that are required.
		<p>:after is followed by a list of forms; these are evaluated upon returning from the function. The values of these forms are also printed out by the tracer. The forms are evaluated after printing out the results of the function call, and if they modify hcl:*traced-results* then the values received by the caller of the function are correspondingly altered (see also hcl:*traced-results*).</p> <p>:allocation — if non-nil, the memory allocation made during a function-call is printed upon exit from the function. This allocation is counted in bytes. If it is any other symbol (except nil), trace uses the symbol to accumulate the</p>

amount of allocation made between entering and exiting the function. Upon exit from the function, the symbol contains the number of bytes allocated during the function-call. For example,

```
(trace (print :entrycond nil
              :exitcond nil
              :allocation $$print-allocation))
```

results in `$$print-allocation` containing the sum of the allocation made inside `print`.

Note that if the function is called again, `trace` continues to use `$$print-allocation` as an accumulator of memory allocation. It adds to the present value rather than re-initializing it each time the function is called.

`:backtrace` generates a backtrace on each call to the traced function. It is followed by a keyword that can be any of the following values:

- `:quick` Like the `:bq` debugger command.
- `t` Like the `:b` debugger command.
- `:verbose` Like the `:b :verbose` debugger command.
- `:bug-form` Like the `:bug-form` debugger command.
- `:before` is followed by a list of forms; these are evaluated upon entering the function and their values are printed out by the tracer. The forms are evaluated after printing out the arguments to the function, and if they alter `*traced-arglist*` then the values received by the body of the function are changed accordingly (see also `*traced-arglist*`).
- `:eval-after` and `:eval-before` are similar to `:after` and `:before`, without output.
- `:break` is followed by a form. This is evaluated after printing the standard information caused by entering the function, and after executing any `:before` forms; if it returns `nil` then tracing continues normally, otherwise `break` is called. This provides a way of entering the debugger through the tracer.

:break-on-exit is followed by a form. This is evaluated after printing the standard information caused by returning from the function, and before executing any **:after** forms; if it returns `nil` then tracing continues normally, otherwise **break** is called. This provides a second way of entering the debugger through the tracer.

:entrycond controls the printing of the standard entry message (including the function's arguments). If the form following it evaluates to give a non-`nil` value when the function is entered, then the entry message is printed (but otherwise it is not). If this option is not present then the standard entry message is always printed upon calling the function. See also the **:when** option.

:exitcond controls the printing of the standard exit message (including the function's results). If the form following it evaluates to give a non-`nil` value when the function is exited, then the exit message is printed (but otherwise it is not). If this option is not present then the standard exit message is always printed upon returning from the function. See also the **:when** option.

:inside restricts the tracing to within one of the functions given as an argument. A single symbolic function name is treated as a list of one element, i.e. **:inside format** is equivalent to **:inside (format)**.

:process may be used to restrict the tracing to a particular process. If it is followed by a process then the function is only traced when it is invoked from within that process. If it is followed by `t` then it is traced from all processes — this is the default. In any other cases the function is not traced at all.

:trace-output should be followed by a stream. All the output from tracing the function is sent to this stream. By default output from the tracer is sent to ***trace-output***. Use of this argument allows you to dispatch traced output from different functions to different places.

:step, when non-`nil`, invokes the stepper (for evaluated functions).

:when overrides all other keywords. It is followed by an expression, and tracing only occurs when that expression evaluates to `non-nil`. It is useful if you want to combine **:entrycond** and **:exitcond**.

Values	<i>trace-result</i>	If <code>trace</code> is called with no arguments then it returns a list of the names of all the functions currently being traced. When called with one or more arguments, it returns the symbols of the functions specified in those arguments.
Description	<p><code>trace</code> is the macro used to invoke the tracing facility. This is a useful debugging tool that enables information about selected calls to be generated by the system. The standard way of invoking <code>trace</code> is to call it with the names of the functions, macros and methods that are to be monitored in this way. Calls to these produce a record of the function that was called, the arguments it received and the results it produced.</p> <p>The arguments to <code>trace</code> each specify a function (or a macro or a method) to be traced. They may also contain further instructions to control how the tracing output is displayed, or to cause particular actions to occur when the functions is called or exited. If <code>trace</code> is called with a function that is already being traced, then the new tracing specification for that function replaces the old version.</p> <p>Note: <code>trace</code> works by tracing function names, not function objects. Therefore tracing function objects, for example by</p> <pre>(trace #'foo)</pre> <p>will not yield any trace output. Also, if the symbol <code>foo</code> is traced, then invoking the function <code>foo</code> by</p> <pre>(funcall (symbol-function 'foo) ...)</pre>	

will not produce any trace output.

Note: for detailed information about the current tracing state, call **tracing-state**.

Example 1

```
USER 1 > (defvar *number-of-calls-to-max* 0)
*NUMBER-OF-CALLS-TO-MAX*

USER 2 > (trace (max :after
                     ((incf *number-of-calls-to-max*))))
(MAX)

USER 3 > (dotimes (i 2) (max i 1))
0 MAX > (0 1)
0 MAX < (1)
1
0 MAX > (1 1)
0 MAX < (1)
2
NIL

USER 4 > *number-of-calls-to-max*
2

USER 5 > (trace (max
                  :entrycond
                  (> (length compiler:*traced-arglist*)
                      2)
                  :exitcond nil))
(MAX)

USER 6 > (max 2 3 (max 4 5))
0 MAX > (2 3 5)
5
```

Example 2

This example illustrates the use of **:inside**.

```
CL-USER 2 > (defun outer ()
                 (inner))
OUTER

CL-USER 3 > (defun inner ()
                 10)
INNER

CL-USER 4 > (trace (inner :inside outer))
               ;, only trace when inside OUTER
(INNER)
```

```
CL-USER 5 > (inner)
      ;; no tracing occurs since we are not inside OUTER
10

CL-USER 6 > (outer)  ;; INNER is traced inside OUTER
0 INNER > NIL
0 INNER < (10)
10

CL-USER 7 >
```

Example 3 To trace a method:

```
(defmethod foo (x) x)
(trace ((method foo (t))))
```

Example 4 To trace a setf function:

```
CL-USER 56 > (defvar *a* 0)
*A*

CL-USER 57 > (defun (setf foo) (x y) (set y x))
(SETF FOO)

CL-USER 58 > (trace (setf foo))
((SETF FOO))

CL-USER 59 > (setf (foo '*a*) 42)
0 (SETF FOO) > (42 *A*)
  >> X : 42
  >> Y : *A*
0 (SETF FOO) < (42)
42
```

See also

disable-trace
max-trace-indent
trace-indent-width
trace-level
trace-new-instances-on-access
trace-on-access
trace-print-circle
trace-print-length
trace-print-level
trace-print-pretty

```
*trace-verbose*
*traced-arglist*
*traced-results*
tracing-enabled-p
tracing-state
untrace
```

truename	<i>Function</i>
-----------------	-----------------

truename	<i>Function</i>	
Summary	Returns the truename of a pathname.	
Package	<code>common-lisp</code>	
Signature	<code>truename filespec => truename</code>	
Arguments	<code>filespec</code>	A pathname designator.
Values	<code>truename</code>	A fully-specified physical pathname.
Description	<p>The function <code>truename</code> behaves as specified in ANSI Common Lisp. The returned value is a fully-specified pathname. Truenames are always fully-specified in LispWorks (this prevents them from ever being corrupted by <code>*default-pathname-defaults*</code>). Note that this means that the paths returned by <code>directory</code> are always fully specified.</p>	
See also	<code>directory</code>	

untrace	<i>Macro</i>
----------------	--------------

Summary	Turns off the Common Lisp tracing facility on the named functions.
Package	<code>common-lisp</code>

Signature	<code>untrace {function-name / method-desc}* => untrace-list</code>
Arguments	<p><code>function-name</code> A symbol whose symbol-function is no longer to be traced.</p> <p><code>method-desc</code> Is a method description, as described in the entry for <code>trace</code>. See <code>trace</code> for more details.</p>
Values	When called with no arguments, it returns the symbols of all functions currently being traced. When called with one or more functions as arguments, <code>untrace</code> returns a list of the symbols of those functions. Thus, in all situations, <code>untrace</code> returns a list of the symbols of those functions being untraced.
Description	<code>untrace</code> is used to cease the tracing of functions. If it is called with no arguments then the tracing of all currently traced functions is stopped. If it is called with one or more symbols then the tracing of those functions is stopped. A warning is given if <code>untrace</code> is called with a function that is not being traced.
Examples	<pre>USER 12 > (progn (untrace) (trace + - / *)) *</pre> <pre>USER 13 > (+ 2 3) 0 + > (2 3) 0 + < (5) 5</pre> <pre>USER 14 > (untrace + -) (* /)</pre> <pre>USER 15 > (+ 2 3) 5</pre> <p>To untrace a method:</p> <pre>(untrace (clos:method foo (t)))</pre>

See also **trace**
 untrace-new-instances-on-access
 untrace-on-access

with-output-to-string *Macro*

Summary	Creates a character output stream, performs a series of operations that may send results to this stream, and then closes the stream.
Package	common-lisp
Signature	with-output-to-string (var &optional string-form &key element-type) declaration form => result
Description	The macro with-output-to-string behaves as specified in the ANSI Common Lisp Standard with one exception: the default value of <i>element-type</i> is the value of *default-character-element-type* . Therefore for strict compliance you must call set-default-character-element-type to set the default string type to character .
See also	compile-file declare proclaim *default-character-element-type* set-default-character-element-type

4

The COMPILER Package

This chapter describes symbols available in the `COMPILER` package.

deftransform *Macro*

Summary	Defines a function that computes the expansion of a form.	
Package	<code>compiler</code>	
Signature	<code>deftransform name transform-name lambda-list &body body => list-of-transforms</code>	
Arguments	<i>name</i>	A symbol naming the function to which the transform is to be applied.
	<i>transform-name</i>	The symbol naming the transformation — it should be unique for the function being transformed — and provides a handle with which to redefine an existing transform.
	<i>lambda-list</i>	This must match against the form being expanded before expansion is allowed to take place, in the sense that it must be valid

	to call a function with such a lambda-list using the arguments supplied in the candidate-form for expansion.
<i>body</i>	The body of the expander function, the result of which replaces the original form (unless it evaluates to <code>compiler::%pass%</code> , in which case no transformation is applied).
Values	<i>list-of-transforms</i> A list of the names of transforms defined for the function, including the one just added.
Description	<code>deftransform</code> , like <code>defmacro</code> , defines a function that computes the expansion of a form. Transforms are only used by the compiler and not by the interpreter. <code>deftransform</code> allows you to add to the optimizations performed by the compiler.
Examples	<pre>(compiler:deftransform + +-of-2 (x y) '(system:: +2 ,x ,y)) (compiler:deftransform + +-of-many (x &rest y) '(system:: +2 ,x (+ ,@y))) ;; now an expression like (+ a b c 4 5 7 d e f) ;; compiles to use the binary version ;; of + (inlined by default), ;; rather than the full (slow) version of + (compiler:deftransform list list-of-1 (x) '(cons ,x '())) (compiler:deftransform list list-of-2 (x y) '(cons ,x (cons ,y '()))) ;; save having to call list - ;; cons is inlined by default (compiler:deftransform constant my-trans (x) (cond ((constantp x) x) ((consp x) '(quote ,(eval x))) (t 'compiler::%pass%))) ; give up if not a cons (compile (defun three-list () (constant (list 1 2 3))))</pre>

```
; ; the function three-list returns the  
; ; same list (1 2 3)  
; ; every time it is called...
```

The `list-of-2` example returns

```
(LIST-OF-2 LIST-OF-1 COMPILER::LIST-TRANSFORM)
```

as its result, since a similar transform already exists in the compiler, by the name `compiler::list*-transform`.

Notes	<p><code>deftransform</code> differs from <code>defmacro</code> in various ways:</p> <p>The evaluation of the body can return <code>compiler:%pass%</code> indicating that the form is not to be expanded (in other words, the transform method has elected to give up trying to improve the code).</p> <p>The compiler only calls the expander function if the arguments match the lambda list — macros are unconditionally expanded.</p> <p>There can be several <code>deftransforms</code> for the same symbol, each having a different name. (The compiler calls each one in turn until one succeeds. This repeats until they all pass, so that the replacement form may itself be transformed.)</p> <p>If a transform takes keyword arguments the compiler preserves the correct order of evaluation.</p> <p>A carelessly written <code>deftransform</code> may lead the compiler to transform valid Common Lisp into incorrect code — there is no semantic checking of the transform.</p>
-------	---

See also	<code>compile</code> <code>compile-file</code>
----------	---

5

The DBG Package

This chapter describes symbols available in the `DBG` package, used to configure the debugging information produced by LispWorks.

debug-print-length		<i>Variable</i>
Summary	Controls the number of object components printed in debugger output.	
Package	<code>dbg</code>	
Initial Value	<code>40</code>	
Description		This variable is used to control the number of components of an object which are printed during output from the debugger. If its value is a positive integer then the components up to that number are printed. If it is <code>nil</code> then all the parts of an object are shown.
Examples		<code>USER 83 > (setq dbg:*debug-print-length* 3)</code>

```

3
USER 84 > (aref
'(1 2 3 4 "Jenny" "cottage" "door")
2)

Error: (1 2 3 4 Jenny cottage door) must be
      an array
1 (abort) return to top loop level 0.

Type :c followed by a number to proceed

USER 85 : 1 > :v
Call to ARRAY-ACCESS :
Arg 0 (ARRAY): (1 2 3 ...)
Arg 1 (SUBSCRIPTS): (2)
Arg 2 (SET-P): NIL Arg 3 (VALUE): NIL

```

Notes ***debug-print-length*** is an extension to Common Lisp.

debug-print-level		<i>Variable</i>
Summary	Controls the depth to which nested objects are printed in debugger output.	
Package	dbg	
Initial Value	4	
Description		dbg: *debug-print-level* controls the depth to which nested objects are printed during output from the debugger. If its value is a positive integer then components at or above that level are printed. By definition an object to be printed is considered to be at level 0, its components are at level 1, their subcomponents are at level 2, and so on. If dbg:*debug-print-level* is nil then objects are printed to arbitrary depth.
Example		USER 89 > (setq dbg: *debug-print-level* 2)

```

2
USER 90 > (subseq 3 '(cat (dog) ((goldfish)
                                (((hamster)))))

Error: Illegal START argument (CAT (DOG)
                                ((GOLDFISH))
                                (((HAMSTER))))
1 (abort) return to top loop level 0.

Type :c followed by a number to proceed

USER 91 : 1 > :v
Call to CHECK-START-AND-END  :
Arg 0 (START): (CAT (DOG) (#) (#))
Arg 1 (END): NIL

```

Notes ***debug-print-level*** is an extension to Common Lisp.

hidden-packages		<i>Variable</i>
Summary	A list of packages whose symbols should not be displayed in debugger output.	
Package	dbg	
Initial Value	A list containing the dbg and conditions packages.	
Description	dbg: * hidden-packages * is used by the debugger. It should be bound to a list of packages. If a package is included in the list then any symbols in it are not shown by the debugger. Thus during backtraces the call frames corresponding to functions in these packages are not displayed. This can be useful in restricting the debugger to particular areas.	

Examples	<pre> USER 108 > (setq dbg:*hidden-packages* (cons (find-package 'Lisp) dbg:*hidden-packages*)) (#<The LISP package, 10/224 internal, 829/911 external> #<The DBG package, 865/905 internal, 0/11 external> #<The CONDITIONS package, 577/704 internal, 89/111 external>) USER 109 > (cons unbound-var '(u v)) Error: the variable UNBOUND-VAR is unbound. 1 (continue) Try evaluating it again 2 Return a value to use 3 Return a value to set it to 4 (abort) return to top loop level 0. Type :c followed by a number to proceed USER 110 : 1 > :b 3 Catch frame: (NIL) Catch frame: #: block-catcher-1748 Call to %EVAL : Call to %EVAL : USER 111 : 1 ></pre>
----------	--

Notes	* hidden-packages* is an extension to Common Lisp.
-------	---

print-binding-frames		<i>Variable</i>
Summary	Controls whether binding frames are printed in debugger output.	
Package	<code>dbg</code>	
Initial Value	<code>nil</code>	
Description		This variable is used by the debugger when it displays the stack frames. Binding frames are formed when special variables are bound, but are normally not shown by the debug-

ger. However if the value of `dbg:*print-binding-frames*` is true then the binding frames are shown.

Example

```

CL-USER 16 > (defun print-to-length (object length)
    (let ((*print-length* length))
      (prinnt object)))
PRINT-TO-LENGTH

CL-USER 17 > (setf dbg:*print-binding-frames* t)
T

CL-USER 18 > (print-to-length '(x y z) 2)

Error: Undefined operator PRINNT in form (PRINNT
OBJECT).
  1 (continue) Try invoking PRINNT again.
  2 Return some values from the form (PRINNT OBJECT).
  3 Try invoking something other than PRINNT with the
same arguments.
  4 Set the symbol-function of PRINNT to another
function.
  5 Set the macro-function of PRINNT to another
function.
  6 (abort) Return to level 0.
  7 Return to top loop level 0.

Type :b for backtrace, :c <option number> to proceed,
or :? for other options

CL-USER 19 : 1 > :n print-to-length
Interpreted call to PRINT-TO-LENGTH

CL-USER 20 : 1 > :b :verbose 5
Interpreted call to PRINT-TO-LENGTH:
  OBJECT          : (X Y Z)
  LENGTH          : 2
  *PRINT-LENGTH* : 2

  Block environment contour:
  Tag environment contour:
  Function environment contour
  Variable environment contour: ()

  Tag environment contour:
  Block environment contour:
  Function environment contour
  Variable environment contour: ()

Call to EVAL (offset 184)
  EXP : (PRINT-TO-LENGTH (QUOTE (X Y Z)) 2)

```

```

Binding frame:
  SYSTEM:::*TOP-LOOP-ACTIVE* : -1
  COMPILER:::*IN-COMPILER-HANDLER* : #<Unbound

Marker>
  * : NIL
  ** : NIL
  *** : NIL
  - : NIL
  + : NIL
  ++ : NIL
  +++ : NIL
  /// : NIL
  // : NIL
  / : NIL
  SYSTEM:::*TOP-LOOP-HOOK* : NIL
  SYSTEM:::*USER-COMMANDS* : NIL
  SYSTEM:::*IN-TOP-LEVEL-READ-A-COMMAND* : NIL

```

CL-USER 21 : 1 >

Notes *print-binding-frames* is an extension to Common Lisp.

print-catch-frames

Variable

Summary	Controls whether catch frames are printed in debugger output.
Package	dbg
Initial Value	t
Description	This variable is used by the debugger when it displays the stack frames. Catch frames are created when the special form <code>catch</code> is used. They are set up so that throws to the matching tag can be received. By default the debugger displays these frames, but if <code>*print-catch-frames*</code> is set to <code>nil</code> then the catch frames are no longer shown.

Examples	<pre> USER 17 > (setq dbg:*print-catch-frames* nil) NIL USER 18 > (defun catch-it () (catch 'tag (throw-it) (print "Not caught"))) CATCH-IT USER 19 > (defun throw-it () (throw 'tag (break))) THROW-IT USER 20 > (catch-it) break 1 (continue) return from break. 2 (abort) return to top loop level 0. Type :c followed by a number to proceed USER 21 : 1 > :b 5 Interpreted call to (DEFUN THROW-IT): Call to *%APPLY-INTERPRETED-FUNCTION : Interpreted call to (DEFUN CATCH-IT): Call to *%APPLY-INTERPRETED-FUNCTION : Call to %EVAL : </pre>
Notes	*print-catch-frames* is an extension to Common Lisp.

print-handler-frames <i>Variable</i>	
Summary	Controls whether handler frames are printed in debugger output.
Package	dbg
Initial Value	nil
Description	This variable is used by the debugger when it displays the stack frames. Handler frames are created by error handlers (see the <i>LispWorks User Guide</i>), and are normally not shown by the debugger. However if *print-handler-frames* is set to t then the handler frames are displayed.

Example

```

USER 162 > (setq lw:*print-handler-frames* t)

T
USER 163 > (defun test (n)
  (handler-case (fn-to-use n)
    (type-error () (format t "~%Type error~%") 0)))

TEST
USER 164 > (test #C(1 1))

Error: Undefined function: FN-TO-USE, with args
  (#C(1 1))

1 (continue) Call FN-TO-USE again
  2 (abort) return to top loop level 0.

Type :c followed by a number to proceed

USER 165 : 1 > :b 10
Catch frame: (NIL)
Catch frame: #:|block-catcher-1854|
Call to *%UNDEFINED-FUNCTION-FUNCTION  :
Call to %EVAL  :
Call to RETURN-FROM  :
Call to %EVAL  :
Call to EVAL-AS-PROGN  :
Handler frame: ((TYPE-ERROR %LEXICAL-CLOSURE%
  (LAMBDA
    (CONDITIONS::TEMP)
    (GO #:|lambda-633|))
  ((#:|lambda-632|) (N . #))
  NIL ((#:|lambda-631|) (TEST))
  ((#:|lambda-633| # #))))
Catch frame: "<* Catch All Object *>"
Call to LET  :
```

Notes *print-handler-frames* is an extension to Common Lisp.

print-open-frames

Variable

Summary	Controls whether open frames are printed in debugger output.
Package	dbg

Initial Value	<code>nil</code>
Description	This variable is used by the debugger when it displays the stack frames. Open frames are made by the system and are normally not shown by the debugger. However if <code>*print-open-frames*</code> is set to <code>t</code> then the open frames are displayed. It is unlikely that you need to examine open frames: their use is connected with implementation details.
Examples	<pre>USER 52 > (setq dbg:*print-open-frames* t) T USER 53 > (car 2) Error: Cannot take CAR of 2 1 (abort) return to top loop level 0. Type :c followed by a number to proceed USER 54 : 1 > :b 3 Open frame (5) Open frame (5) Call to CAR-FRAME :</pre>
Notes	<code>*print-open-frames*</code> is an extension to Common Lisp.

print-restart-frames <i>Variable</i>	
Summary	Controls whether restart frames are printed in debugger output.
Package	<code>dbg</code>
Initial Value	<code>nil</code>
Description	This variable is used by the debugger when it displays the stack frames. Restart frames are formed when restarts are established (see the <i>LispWorks User Guide</i>), but are normally not shown by the debugger. However if <code>*print-restart-frames*</code> is set to <code>t</code> then the restart frames are shown.

Example

```

USER 43 > (setq dbg:*print-restart-frames* t)

T
USER 44 > (truncate 12.5 0.0)

Error: Division-by-zero caused by TRUNCATE
      of (12.5 0.0)
1 (continue) Return a value to use
2 Supply new arguments to use
3 (abort) return to top loop level 0.

Type :c followed by a number to proceed

USER 45 : 1 > :b 5
Restart frame: (ABORT)
Catch frame: (NIL)
Catch frame: #:<|block-catcher-3223|
Call to DIVISION-BY-ZERO-ERROR  :
Call to TRUNCATEANY  :
USER 46 : 1 >

```

Notes ***print-restart-frames*** is an extension to Common Lisp.

terminal-debugger-block-multiprocessing *Variable*

Summary	Controls blocking of multiprocessing in the terminal debugger.
Package	dbg
Initial Value	t
Description	<p>When the debugger is entered on the terminal, multiprocessing is blocked if the value of *terminal-debugger-block-multiprocessing* is t. This is the default value.</p> <p>If you set this variable to nil then other processes, including timers, will continue to run in parallel to the process that entered the terminal debugger (as they did before the debugger was entered). Beware that this will make it more difficult to debug multiprocess activities.</p>

The other allowed value is `:maybe`. This means that multiprocessing is blocked in the terminal debugger unless the debugger was entered from the CAPI environment.

The value of `*terminal-debugger-block-multiprocessing*` affects the behavior of a REPL started by `start-tty-listener`.

Example

This listener session illustrates the effect of `*terminal-debugger-block-multiprocessing*`.

Firstly we see the default behavior whereby a call to `print` in another process is blocked by the debugger.

```
CL-USER 1 > dbg:*terminal-debugger-block-
multiprocessing*
T

CL-USER 2 > unbound

Error: The variable UNBOUND is unbound.
 1 (continue) Try evaluating UNBOUND again.
 2 Specify a value to use this time instead of
evaluating UNBOUND.
 3 Specify a value to set UNBOUND to.
 4 (abort) Return to level 0.
 5 Return to top-level loop.
 6 Return from multiprocessing.

Type :b for backtrace, :c <option number> to proceed,
or :? for other options

CL-USER 3 : 1 > (setq *timer* (mp:make-timer 'print
10))
Warning: Setting unbound variable *TIMER*
#<Time Event : PRINT>

CL-USER 4 : 1 > (mp:schedule-timer-relative *timer* 1)
#<Time Event : PRINT>

CL-USER 5 : 1 > :a
```

On leaving the debugger the output 10 from the call to `print` appears. Then we set `*terminal-debugger-block-multiprocessing*` to `nil` and repeat the commands:

```
CL-USER 6 >
10
(setf dbg:*terminal-debugger-block-multiprocessing*
nil)
NIL

CL-USER 7 > unbound

Error: The variable UNBOUND is unbound.
1 (continue) Try evaluating UNBOUND again.
2 Specify a value to use this time instead of
evaluating UNBOUND.
3 Specify a value to set UNBOUND to.
4 (abort) Return to level 0.
5 Return to top-level loop.
6 Return from multiprocessing.

Type :b for backtrace, :c <option number> to proceed,
or :? for other options

CL-USER 8 : 1 > (setq *timer* (mp:make-timer 'print
10))
#<Time Event : PRINT>

CL-USER 9 : 1 > (mp:schedule-timer-relative *timer* 1)
#<Time Event : PRINT>

CL-USER 10 : 1 >
10
```

Notice above that the output 10 from the call to `print` appears after 1 second, in the debugger. Multiprocessing was not blocked.

See also

`start-tty-listener`

6

The DSPEC Package

This chapter describes symbols available in the `DSPEC` package.

For an overview of the `dspec` system, see the *LispWorks User Guide*.

active-finders		<i>Variable</i>
Summary	Controls how source finding operates.	
Package	<code>dspec</code>	
Initial Value	<code>(:internal)</code>	
Description	The <code>*active-finders*</code> variable controls how the functions <code>find-name-locations</code> and <code>find-dspec-locations</code> operate. This in turn controls source the finding commands in the Common LispWorks development environment. You can switch between different sources of location information by setting this variable.	
The legal values for the elements of <code>*active-finders*</code> are:		

:internal	The internal database of definitions performed in this image.
:tags	Prompt for a tags file, when first used.
pathname	Either a tags file or a tags database. A tags database is a fasl file generated by <code>save-tags-database</code> .
	The order of this list determines the order that the results from the finders are combined in — you would usually want :internal to be the first item on this list, as it contains the up-to-date information about the state of the image. More than one pathname is allowed.

See also `discard-source-info`
 `find-dspec-locations`
 `find-name-locations`
 `save-tags-database`

		<i>Macro</i>				
Summary	Tells the dspec system of the source location.					
Package	<code>dspec</code>					
Signature	<code>at-location (location) &body body => result</code>					
Arguments	<table> <tr> <td><i>location</i></td> <td>A pathname or a keyword.</td> </tr> <tr> <td><i>body</i></td> <td>Forms, including defining forms.</td> </tr> </table>	<i>location</i>	A pathname or a keyword.	<i>body</i>	Forms, including defining forms.	
<i>location</i>	A pathname or a keyword.					
<i>body</i>	Forms, including defining forms.					
Values	<i>result</i>	The result of <i>body</i> .				
Description	The macro <code>at-location</code> informs the dspec system that the source for definitions done during the execution of <i>body</i> are at the location <i>location</i> .					

location is usually a pathname, for definitions occurring in a file or editor buffer with that pathname.

Other locations are reserved for internal use. These are:

An editor buffer Defined in an editor buffer with no pathname.

:listener Interactively defined.

:unknown Defined without dspec information being recorded.

:implicit An aggregate defined by the existence of a part.

(:inside dspec loc)

A subform of *dspec* at location *loc*.

canonicalize-dspec

Function

Summary Returns the canonical form for a dspec.

Package **dspec**

Signature **canonicalize-dspec dspec => canonical-dspec**

Arguments *dspec* A dspec.

Values *canonical-dspec* A canonical dspec.

Description The function **canonicalize-dspec** checks that *dspec* is syntactically correct and returns its canonical form if *dspec* is valid. Otherwise **canonicalize-dspec** returns **nil**.

canonicalize-dspec expands dspec aliases

Example	<pre>CL-USER 12 > (dspec:canonicalize-dspec 'foo) (FUNCTION FOO) CL-USER 13 > (dspec:canonicalize-dspec '(defmethod bar (list t))) (METHOD BAR (LIST T))</pre>
See also	define-dspec-alias

def *Macro*

Summary	Informs the system of a name for a definition.	
Package	dspec	
Signature	def <i>dspec</i> & <i>body</i> <i>body</i> => <i>result</i>	
Arguments	<i>dspec</i>	A dspec.
	<i>body</i>	Lisp forms, evaluated as an implicit progn .
Values	<i>result</i>	The result of <i>body</i> .
Description	<p>The macro def informs the system that any definitions within <i>body</i> should be recorded as being within the dspec <i>dspec</i>. This means that when something attempts to locate such a definition, it should look for a definition named <i>dspec</i>.</p> <p>Use def to wrap a group of definitions so that source location for one of the group causes the LispWorks Editor to look for the dspec in the def instead. Typically you will also need a define-form-parser definition for the macro that expands into the def.</p> <p><i>dspec</i> can be non-canonical.</p> <p>You can also use def to provide a dspec for a definition that has its own class that has been defined with define-dspec-</p>	

`class`. In this case, you arrange to call `record-definition` with the same dspec as in the example below.

It is also possible to mix these cases, recording a dspec and also grouping inner definitions. For example `defstruct` does this, recording itself and also grouping definitions such as the constructor function.

In all cases, to make source location work in the LispWorks editor you typically also need a `define-form-parser` definition for the macro that expands into the `def`.

Example

```
(defmacro define-wibble (x y)
  `(dspec:def (define-wibble ,x)
    (set-wibble-definition ',x ',y (dspec:location)))))

(defun set-wibble-definition (x y loc)
  (when (record-definition `(define-wibble ,x) loc)
    ;; defining code here
  ))
```

See also

`location`

define-dspec-alias

Macro

Summary Informs the dspec system that a definer expands into another definer.

Package `dspec`

Signature `define-dspec-alias name lambda-list &body body`

Arguments *name* A symbol naming a definer.

lambda-list A list representing the parameters of a *name* dspec.

body Forms evaluated to yield a dspec.

Description	The macro <code>define-dspec-alias</code> works rather like <code>deftype</code> . Dspecs whose <code>car</code> is <i>name</i> should have parameters that match <i>lambda-list</i> . They will be canonicalized into the dspec returned by <i>body</i> .
	<code>define-dspec-alias</code> is useful when you add a new way of making existing definitions with a new defining form that expands into a system-provided defining form. The dspec system should consider the new and system-provided definers as variant forms of the same dspec class. <code>define-dspec-alias</code> is used to convert one of them to the other during canonicalization by <code>canonicalize-dspec</code> .
Example	<code>defparameter</code> is pre-defined as an alias for <code>defvar</code> .

See also `canonicalize-dspec`

define-dspec-class		<i>Macro</i>										
Summary	Defines a dspec class.											
Package	<code>dspec</code>											
Signature	<code>define-dspec-class name superspace documentation &key pretty-name undefiner canonicalize prettify definedp object-dspec defined-parts aggregate-class</code>											
Arguments	<table> <tr> <td><i>name</i></td><td>A symbol naming the dspec class</td></tr> <tr> <td><i>superspace</i></td><td>A symbol naming the superspace</td></tr> <tr> <td><i>documentation</i></td><td>A string describing the dspec class</td></tr> <tr> <td><i>undefiner</i></td><td>A function that generates the undefining form for the class</td></tr> <tr> <td><i>canonicalize</i></td><td>A function to canonicalize a dspec if it belongs to the class</td></tr> </table>	<i>name</i>	A symbol naming the dspec class	<i>superspace</i>	A symbol naming the superspace	<i>documentation</i>	A string describing the dspec class	<i>undefiner</i>	A function that generates the undefining form for the class	<i>canonicalize</i>	A function to canonicalize a dspec if it belongs to the class	
<i>name</i>	A symbol naming the dspec class											
<i>superspace</i>	A symbol naming the superspace											
<i>documentation</i>	A string describing the dspec class											
<i>undefiner</i>	A function that generates the undefining form for the class											
<i>canonicalize</i>	A function to canonicalize a dspec if it belongs to the class											

<i>prettyf</i>	A function to return a prettier form of a dspec of the class
<i>definedp</i>	A function to decide if a dspec of the class currently has a definition
<i>object-dspec</i>	A function to return the dspec from an object if it was defined by the class
<i>defined-parts</i>	A function to return all the currently defined parts in the class for a given a primary-name
<i>aggregate-class</i>	The aggregate dspec class for a part dspec
Description	<p>The macro define-dspec-class defines a dspec class, providing handlers for definitions in that dspec class.</p> <p>define-dspec-class defines <i>name</i> as a dspec class, inheriting from the dspec class <i>superspace</i>. <i>superspace</i> should be <code>nil</code> to define a new top-level dspec class.</p> <p><i>documentation</i> should be a string documenting the dspec class. For example "<code>My Objects</code>".</p> <p>After evaluating a define-dspec-class form, <i>name</i> can be used by defining forms to record locations of definitions of that dspec class name by calling record-definition.</p> <p>All of the remaining arguments described below can be omitted if not needed. The most important arguments for the LispWorks IDE are <i>definedp</i> and <i>undefiner</i>.</p> <p>If <i>undefiner</i> is given, its value must be a function of one argument. When LispWorks wants to remove a definition, it will call the function with a canonical dspec of class <i>name</i>. The function should returns a form that removes the current definition of that dspec. For example, the undefining form for package dspecs might be delete-package. If <i>undefiner</i> is omitted, then definitions of this class cannot be undefined.</p> <p>If <i>canonicalize</i> is given, its value must be a function of one argument. The function will be called by canonicalize-dspec for a dspec of the given class. The value returned by</p>

the canonicalize function must be a fully canonical dspec of the given class. A typical use for the canonicalize function would be to remove extra options from the dspec which are not required to make the dspec unique. The canonicalize function should return `nil` for malformed dspecs and should take care not to signal an error. The default canonicalize function returns the dspec if it matches the form

`(dspec-class symbol)`

If `prettify` is given, its value must be a function of one argument. When LispWorks wants to print a dspec, for example in an error message, it will call the prettify function for the class of the dspec. The argument will be the canonical dspec and the function should return a dspec which is considered "prettier" for a user to see. The default prettify function returns the dspec unchanged.

If `definedp` is given, its value must be function of one argument. When LispWorks wants to discover if a given dspec is defined, it calls the function with the `dspec-primary-name` of the dspec. The `definedp` function should return true if the primary name is defined in this dspec class and `nil` otherwise. The default `definedp` function always returns `nil`.

If `object-dspec` is given, its value must be a function of one argument. When LispWorks wants to find the dspec that created a given object (for example a package object created by a `defpackage` form), it calls the `object-dspec` functions in all dspec classes. The function should return a dspec for the object if that object was defined by the dspec class or `nil` otherwise. For example, the `object-dspec` function for package dspecs might be:

```
#'(lambda (obj)
  (and (packagep obj)
       `(package ,(package-name obj))))
```

The `object-dspec` function is used by the "Find Source" menu option in the Inspector in the LispWorks IDE to find where the current object was defined.

If *defined-parts* is given, its value must be a function of one argument. When LispWorks wants to find all the definitions that are parts of a given aggregate dspec class, it calls the *defined-parts* functions with the **dspec-primary-name** of the dspec in each class that aggregates with it. The function should return a list of dspecs which are defined parts of the primary name in the class *name*. If this keyword is given, *aggregate-class* must also be given.

If *aggregate-class* is given, its value must be a symbol naming a dspec class that is the aggregate class of the parts defined by *name* dspecs. For example, the aggregate class of **method** is **defgeneric** because methods are the defined parts of a particular generic function. If this keyword is given, the *defined-parts* must also be given.

To make **c1:documentation** work for your dspec class, add a suitable method as described for **documentation**.

Example See the section "Dspec classes" in the *LispWorks User Guide*.

See also **canonicalize-dspec**
def
dspec-primary-name
record-definition

define-form-parser *Macro*

Summary Establishes a parser for top level forms with the given definier.

Package **dspec**

Signature **define-form-parser definer-and-options &optional parameters**
&body body => parser

Arguments *definer-and-options*

		A symbol <i>definer</i> naming a definer of functions, macros, variables and so on, or a list (<i>definer options</i>) where <i>options</i> is a plist of keys and values.
	<i>parameters</i>	<i>nil</i> , or list of parameters <i>params</i> in the top level form, optionally ending with &rest param-getter .
	<i>body</i>	The body of a parser function.
Values	<i>parser</i>	A form parser function.
Description		<p>The macro define-form-parser defines a form parser for forms beginning with <i>definer</i>. <i>options</i> is a property list with the following keys allowed:</p> <ul style="list-style-type: none"> :parser A parser function <i>parser-function</i>. :alias A dspec class or alias <i>alias</i>. :anonymous A boolean. <p>The parser function defined is named by <i>parser-function</i>. If the :parser option is omitted then the name defaults to a symbol in the current package whose symbol name is the symbol name of <i>definer</i> with "-FORM-PARSER" appended.</p> <p>If <i>parameters</i> and <i>body</i> are given, then <i>parser-function</i> is defined as a global function that is expected to return a dspec for the defining form or <i>nil</i> if this is not possible. Within <i>body</i>, <i>definer</i> is bound to the car of the actual form being parsed. In simple cases, this is just <i>definer</i>, but if the form parser is used as in the :alias option of another form parser then the symbol will be bound to the car of that form instead.</p> <p>The <i>params</i> are bound to subsequent subforms of the defining form. If &rest param-getter is supplied, then it is bound to a function of no arguments that returns two values: the next subform if there is one and a boolean to indicate if a subform was found.</p>

If *parameters* and *body* are omitted, then *parser-function* is expected to be a form parser defined by a different `define-form-parser` form, or you can specify as an alias a definer with an existing form parser via the value *alias* of the `:alias` key in *options*.

If the `:anonymous` option is non-`#nil` then *definer* is not associated with the form parser. This is useful in conjunction with *parameters* and *body* for defining generic form parsers that can be used in other `define-form-parser` forms.

LispWorks contains pre-defined form parser functions for the Common Lisp definers `defun`, `defmethod`, `defgeneric`, `defvar`, `defparameter`, `defconstant`, `defstruct`, `defclass`, `defmacro` and `deftype` and for LispWorks definers such as `fli:define-foreign-type` and `dspec:define-form-parser` itself.

When a defining symbol *definer* has an associated form parser, this parser function is used by the source location commands such as **Expression > Find Source** in the Common LispWorks development environment. Having identified the file where the definition was recorded, LispWorks parses the top level forms in the file looking for the one which matches the definition spec. When found, this match is displayed.

Example

Define a parser for `def-foo` forms which have a single name as the second element in the form:

```
(dspec:define-form-parser def-foo (name)
  `(&,def-foo ,name))
```

Define a parser for `def-other-foo` forms which are like `def-foo` forms:

```
(dspec:define-form-parser
  (def-other-foo (:parser def-foo-form-parser)))
```

Define a parser for `def-bar` forms whose name is made from the second element of the form and any subsequent keywords:

```
(dspec:define-form-parser def-bar (name &rest details)
  `,(def-bar ,(name
    `,@(loop for detail = (funcall details)
      while (keywordp detail)
      collect detail))))
```

Define a parser for forms which have another name as the second element in the form:

```
(dspec:define-form-parser (two-names
  (:anonymous t)) (name1 name2)
  `,(two-names ,name1 ,name2))
```

Define a new way to define CLOS methods, and tell the dspec system to treat them the same. Note the use of `define-dspec-alias` to inform the dspec system that `my-defmethod` is another way of naming `defmethod` dspecs:

```
(defmacro my-defmethod (name args &body body)
  `,(defmethod ,(name ,args
    ,@body)))

(dsdp:define-dspec-alias my-defmethod
  (name &rest args)
  `,(defmethod ,(name ,@args)))

(my-defmethod foo ((x number))
  42)

(dsdp:define-form-parser
  (my-defmethod
    (:parser
      #.(dsdp:get-form-parser 'defmethod))))
```

A simpler way to write the last form is:

```
(dsdp:define-form-parser
  (my-defmethod
    (:alias defmethod)))
```

See also

`get-form-parser`
`parse-form-dspec`

dspec-class	<i>Function</i>
Summary	Returns the dspec class of a dspec.
Package	<code>dspec</code>
Signature	<code>dspec-class dspec => class</code>
Arguments	<code>dspec</code> A dspec.
Values	<code>class</code> A dspec class name.
Description	The function <code>dspec-class</code> returns the dspec class name for <i>dspec</i> .
Example	<pre>CL-USER 14 > dspec:dspec-class 'foo FUNCTION CL-USER 15 > dspec:dspec-class '(defmacro foo) DEFMACRO CL-USER 16 > dspec:dspec-class '(defmethod foo) DEFMETHOD</pre>
See also	<code>dspec-name</code>

dspec-classes	<i>Variable</i>
Summary	Lists all the dspec classes.
Package	<code>dspec</code>
Signature	<code>*dspec-classes*</code>
Description	The variable <code>*dspec-classes*</code> contains a list of the names of all the dspec classes.

		<i>Function</i>
dspec-defined-p		
Summary	The predicate for whether a dspec has a definition.	
Package	<code>dspec</code>	
Signature	<code>dspec-defined-p dspec => definedp</code>	
Arguments	<code>dspec</code>	A dspec.
Values	<code>definedp</code>	The canonical form of <i>dspec</i> if <i>dspec</i> is defined, or <code>nil</code> otherwise.
Description	The function <code>dspec-defined-p</code> determines whether the dspec <i>dspec</i> has a definition. If so, it returns the canonical form of <i>dspec</i> . If <i>dspec</i> has no definitions, <code>dspec-defined-p</code> returns <code>nil</code> .	
Example	<code>CL-USER 23 > (dspec:dspec-defined-p '(function list)) (DEFUN LIST)</code>	

		<i>Function</i>
dspec-definition-locations		
Summary	Returns the locations of the known definitions.	
Package	<code>dspec</code>	
Signature	<code>dspec-definition-locations dspec => locations</code>	
Arguments	<code>dspec</code>	A dspec.
Values	<code>locations</code>	A list of pairs (<i>recorded-dspec location</i>).
Description	The function <code>dspec-definition-locations</code> returns the locations of the definitions recorded for the dspec <i>dspec</i> .	

For each known definition *recorded-dspec* names the definition that defined *dspec* in *location*, and *location* is a pathname or keyword as described in [at-location](#).

Note that non-file locations, such as `:unknown`, can occur in the list. The locations in *locations* are all basic locations: that is, there are no `(:inside ...)` locations.

If *dspec* is a local dspec, the parent function is located.

Example `CL-USER 6 > (dspec:dspec-definition-locations
 '(defun foo-bar))
 (((DEFSTRUCT FOO) #P"C:/temp/hack.lisp"))`

See also [name-definition-locations](#)

dspec-equal *Function*

Summary Tests two dspecs for equality as dspecs.

Package `dspec`

Signature `dspec-equal dspec1 dspec2 => result`

Arguments `dspec1, dspec2` Dspecs.

Values `result` A boolean.

Description The function `dspec-equal` compares *dspec1* and *dspec2* for equality as dspecs.

Both arguments are canonicalized before the comparison.

Dspecs in different subclasses of the same namespace are `dspec-equal` if their names match.

Unknown dspecs are compared simply by `equal`.

Example CL-USER 44 > (dspec:dspec-equal '(deftype foo)
 '(defclass foo))
 T

	<i>Function</i>
dspec-name	
Summary	Extracts the name from a canonical dspec.
Package	dspec
Signature	dspec-name <i>dspec</i> => <i>name</i>
Arguments	<i>dspec</i> A canonical dspec.
Values	<i>name</i> A dspec name.
Description	The function dspec-name extracts the name from the canonical dspec <i>dspec</i> . Note that for part classes this is a list starting with the primary name. If <i>dspec</i> is not canonicalized, dspec-name signals an error.
See also	dspec-class

	<i>Function</i>
dspec-primary-name	
Summary	Extracts the primary name from a canonical dspec.
Package	dspec
Signature	dspec-primary-name <i>dspec</i> => <i>name</i>
Arguments	<i>dspec</i> A canonical dspec.
Values	<i>name</i> A dspec name.

Description	The function <code>dspec-primary-name</code> extracts the primary name from the canonical dspec <i>dspec</i> . Note that for part classes this is the name of the aggregate definition, for example for methods it returns the name of the generic function.
See also	<code>dspec-class</code>

dspec-progenitor *Function*

Summary	Returns the ultimate parent of a <code>subfunction</code> dspec.	
Signature	<code>dspec-progenitor dspec => result</code>	
Package	<code>dspec</code>	
Arguments	<code>dspec</code>	A dspec.
Values	<code>result</code>	A dspec.
Description	The function <code>dspec-progenitor</code> returns a dspec <i>result</i> which is the ultimate parent of a <code>subfunction</code> dspec argument <i>dspec</i> . If the argument <i>dspec</i> is not a local dspec, it is simply returned. Note that <i>result</i> is not necessarily a canonical dspec.	
Example	<pre>(dspec-progenitor '(subfunction 1 (subfunction (flet a) (defun foo)))) => (defun foo)</pre>	
See also	<code>local-dspec-p</code>	

		<i>Function</i>
	dspec-subclass-p	
Summary		Tests whether one dspec class is a subclass of another.
Package		<code>dspec</code>
Signature		<code>dspec-subclass-p <i>class1</i> <i>class2</i> => <i>result</i></code>
Arguments	<i>class1, class2</i>	Symbols naming dspec classes.
Values	<i>result</i>	A boolean.
Description		The function <code>dspec-subclass-p</code> determines whether the dspec class denoted by <i>class1</i> is a subclass of that denoted by <i>class2</i> .
Example		<pre>CL-USER 55 > (dspec:dspec-subclass-p 'defmacro 'type) NIL CL-USER 56 > (dspec:dspec-subclass-p 'defmacro 'function) T</pre>

		<i>Function</i>
	dspec-undefiner	
Summary		Returns an undefining expression for a dspec.
Package		<code>dspec</code>
Signature		<code>dspec-undefiner <i>dspec</i> => <i>form</i></code>
Arguments	<i>dspec</i>	A dspec.
Values	<i>form</i>	A Lisp form.
Description		The function <code>dspec-undefiner</code> returns a form which would undefine dspec, whether or not <i>dspec</i> is currently defined.

If no such form can be constructed, `nil` is returned.

Example	<pre>CL-USER 66 > (dspec:dspec-undefiner '(defun foo)) (PROGN (FMAKUNBOUND (QUOTE FOO)) (SETF (DOCUMENTATION (QUOTE FOO) (QUOTE FUNCTION)) NIL))</pre>
---------	---

discard-source-info

Function

Summary	Clears the internal dspec database.
Package	<code>dspec</code>
Signature	<code>discard-source-info => nil</code>
Arguments	None.
Values	Returns <code>nil</code> .
Description	The function <code>discard-source-info</code> removes all source location information from the internal dspec database.
Example	To build <code>my-image</code> which does not contain source locations for the definitions loaded, but retaining a tags database of those definitions: <pre>(load-all-patches) (load "my-code") (dsdp:save-tags-database #P"my-tags-database.ofas1") (dsdp:discard-source-info) (save-image "my-image")</pre>
See also	<code>save-tags-database</code>

find-dspec-locations

Function

Summary	Returns the locations of the definitions of a dspec.
---------	--

Package	<code>dspec</code>	
Signature	<code>find-dspec-locations dspec => locations</code>	
Arguments	<code>dspec</code>	A dspec.
Values	<code>locations</code>	A list of pairs (<i>recorded-dspec location</i>).
Description	<p>The function <code>find-dspec-locations</code> returns the locations of the relevant definitions for the dspec <i>dspec</i>.</p> <p>For each known definition <i>recorded-dspec</i> names the definition that defined <i>dspec</i> in <i>location</i>, and <i>location</i> is a pathname or keyword as described in <code>at-location</code>.</p> <p>If <i>dspec</i> is a local dspec, the parent function is located.</p> <p>The location information is collected from all finders on <code>*active-finders*</code>, that is, the relevant definitions are those known to at least one of these finders.</p> <p>If two or more finders return the same pair (<i>recorded-dspec location</i>), as compared by <code>dspec-equal</code> and location equality, then only the first occurrence of the pair (in the order of <code>*active-finders*</code>) appears in <i>locations</i>.</p>	
See also	<code>*active-finders*</code> <code>dspec-definition-locations</code> <code>dspec-equal</code>	

	Function
Summary	Returns the locations of the definitions of a name.
Package	<code>dspec</code>
Signature	<code>find-name-locations classes name => locations</code>

Arguments	<i>classes</i>	A list of dspec class names.
	<i>name</i>	A name.
Values	<i>locations</i>	A list of pairs (<i>recorded-dspec location</i>).
Description		<p>The function <code>find-name-locations</code> returns the locations of the relevant definitions for <i>name</i> in the classes listed in <i>classes</i>.</p> <p>For each known definition <i>recorded-dspec</i> names the definition that defined <i>name</i> in <i>location</i>, and <i>location</i> is a pathname or keyword as described in <code>at-location</code>.</p> <p>The location information is collected from all finders on <code>*active-finders*</code>, that is, the relevant definitions are those known to at least one of these finders.</p> <p>If two or more finders return the same pair (<i>recorded-dspec location</i>), as compared by <code>dspec-equal</code> and location equality, then only the first occurrence of the pair (in the order of <code>*active-finders*</code>) appears in <i>locations</i>.</p>
See also		<code>*active-finders*</code> <code>name-definition-locations</code> <code>dspec-equal</code>

get-form-parser		<i>Function</i>
Summary		Returns the form parser associated with a definer.
Package		<code>dspec</code>
Signature		<code>get-form-parser definer => parser</code>
Arguments	<i>definer</i>	A symbol naming a definer.
Values	<i>parser</i>	A form parser function, or <code>nil</code> .

Description	The function <code>get-form-parser</code> returns a form parser function if there is one associated with <code>definer</code> . This is the case for predefined definers and for those for which you have established a form parser using <code>define-form-parser</code> . If there is no associated form parser, <code>nil</code> is returned.
Example	<pre>CL-USER 1 > dspec:get-form-parser 'defun DSPEC:NAME-ONLY-FORM-PARSER</pre>
See also	<code>define-form-parser</code> <code>parse-form-dspec</code>

	<i>Function</i>
Summary	The predicate for local dspecs.
Package	<code>dspec</code>
Signature	<code>local-dspec-p dspec => localp</code>
Arguments	<code>dspec</code> A dspec.
Values	<code>localp</code> A boolean.
Description	The function <code>local-dspec-p</code> determines whether the dspec <code>dspec</code> is a local dspec. Local dspecs name local definitions, such as local functions. Currently a local dspec is a list whose <code>car</code> is <code>subfunction</code> .
See also	<code>dspec-progenitor</code>

location	<i>Macro</i>
Summary	Returns the source location.
Package	<code>dspec</code>
Signature	<code>location => location</code>
Values	<code>location</code> A pathname or a keyword.
Description	The macro <code>location</code> returns a location suitable for passing to <code>record-definition</code> . This is usually done via a separate defining function. You will need to use <code>location</code> only if you create your own ways of making definitions (and not if your definers call only system-provided definers).
Example	<pre>(defmacro define-wibble (x y) `(dspec:def (define-wibble ,x) (set-wibble-definition ',x ',y (dspec:location))))) (defun set-wibble-definition (x y loc) (when (record-definition `(define-wibble ,x) loc) ;; defining code here))</pre>
See also	<code>at-location</code> <code>def</code>

name-defined-dspecs	<i>Function</i>
Summary	Returns defined dspecs matching a name.
Package	<code>dspec</code>
Signature	<code>name-defined-dspecs classes name => dspecs</code>
Arguments	<code>classes</code> A list of dspec class names.

	<i>name</i>	A name.
Values	<i>dspecs</i>	A list of canonical dspecs.
Description		The function <code>name-defined-dspecs</code> looks in each of the dspec classes <i>classes</i> for definitions of <i>name</i> . For each definition found (as if by <code>dspec-defined-p</code>), the result <i>dspecs</i> contains the canonical dspec.
See also		<code>dspec-defined-p</code>

name-definition-locations

Function

Summary	Returns the locations of the known definitions.	
Package	<code>dspec</code>	
Signature	<code>name-definition-locations classes name => locations</code>	
Arguments	<i>classes</i>	A list of dspec class names.
	<i>name</i>	A name.
Values	<i>locations</i>	A list of pairs (<i>recorded-dspec location</i>).
Description		The function <code>name-definition-locations</code> returns the locations of the definitions recorded for the name <i>name</i> in any of the dspec classes in <i>classes</i> . For each known definition <i>recorded-dspec</i> names the definition that defined <i>name</i> in <i>location</i> , and <i>location</i> is a pathname or keyword as described in <code>at-location</code> .
Example	<pre>CL-USER 7 > (dspec:name-definition-locations '(function) 'foo-bar) (((DEFSTRUCT FOO) #P"C:/temp/hack.lisp"))</pre>	

See also [dspec-definition-locations](#)

name-only-form-parser *Function*

Summary A pre-defined form parser.

Package `dspec`

Signature `name-only-form-parser top-level-form getter => dspec`

Arguments `top-level-form` A top level defining form.

`getter` The subform getter function.

Values `dspec` A `dspec`.

Description The function `name-only-form-parser` is a predefined form parser for use with `define-form-parser`. The parser consumes one subform and returns it.

`name-only-form-parser` can be used for function definitions where the function name is an abbreviation for the full `dspec`. It is the predefined parser for `defun`, `defmacro` and `defgeneric` forms.

You can define it to be the parser for your defining forms. using `define-form-parser`.

Example

```
(defmacro my-definer (name &body body)
  `(#(defun ,name (x)
    ,@body))

(dsdp:define-form-parser
  (my-definer (:parser
    dsdp:name-only-form-parser)))
```

See also [define-form-parser](#)

		<i>Function</i>
	parse-form-dspec	
Summary		Parses the dspec from a defining form.
Package		<code>dspec</code>
Signature		<code>parse-form-dspec <i>form</i> => <i>result</i></code>
Arguments	<i>form</i>	A form.
Values	<i>result</i>	A dspec or <code>nil</code> .
Description		The function <code>parse-form-dspec</code> invokes the defined form parser for <i>form</i> and returns the resulting dspec.
Example		<pre>(parse-form-dspec '(def-foo my-foo (arg) (foo-it arg))) => (def-foo my-foo)</pre>
See also		<code>define-form-parser</code> <code>get-form-parser</code>
	record-definition	
Summary		Checks for existing definitions and records a new definition.
Package		<code>dspec</code>
Signature		<code>record-definition <i>dspec</i> <i>location</i> &key <i>check-redefinition-p</i> => <i>result</i></code>
Arguments	<i>dspec</i>	A dspec.
	<i>location</i>	A pathname or keyword.
	<i>check-redefinition-p</i>	A boolean.

Values	<i>result</i>	A generalised boolean.
Description	The function record-definition tells the system that <i>dspec</i> is defined at <i>location</i> . The system-provided definer macros call the function record-definition with the current location. <i>location</i> should be a pathname or keyword as returned by location .	When <i>check-redefinition-p</i> is true, it checks for existing definitions according to the value of *redefinition-action* . The default value of <i>check-redefinition-p</i> is t. If the definition is made, then <i>result</i> is true. If the definition is not made then <i>result</i> is nil. This can happen if you choose the "Don't redefine ..." restart at a redefinition error. Note: You should not usually call record-definition , since all the system-provided definers call it. However, for new classes of definition which you add with define-dspec-class , you should call record-definition for dspecs in their new classes.
Compatibility note	record-definition was documented in the lispworks package in LispWorks 4.3 and earlier. Although it is currently still available there, this may change in future releases and you should now reference it via the dspec package.	
See also	define-dspec-class *redefinition-action* location	

record-source-files

Variable

Summary	Controls whether the locations of definitions are recorded.
Package	dspec

Initial value	<code>t</code>
Description	The variable <code>*record-source-files*</code> controls whether locations of definitions are recorded in the internal tags database.
Compatibility note	<code>*record-source-files*</code> was documented in the <code>lispworks</code> package in LispWorks 4.3 and earlier. Although it is currently still available there, this may change in future releases and you should now reference it via the <code>dspec</code> package.
See also	<code>*active-finders*</code>

redefinition-action *Variable*

Summary	Specifies the action on some redefinitions.
Package	<code>dspec</code>
Initial value	<code>:warn</code>
Description	<code>*redefinition-action*</code> specifies what action should be taken when methods and other forms are redefined. If <code>*redefinition-action*</code> is set to <code>:warn</code> then you are warned. If it is set to <code>:quiet</code> or <code>nil</code> , the redefinition is done quietly. If, however, it is set to <code>:error</code> , then LispWorks signals an error.
Compatibility note	<code>*redefinition-action*</code> was documented in the <code>lispworks</code> package in LispWorks 4.3 and earlier. Although it is currently still available there, this may change in future releases and you should now reference it via the <code>dspec</code> package.
See also	<code>*handle-warn-on-redefinition*</code>

		<i>Function</i>
Summary	Saves the current internal dspec database to a given file.	
Package	<code>dspec</code>	
Signature	<code>save-tags-database pathname => pathname</code>	
Arguments	<i>pathname</i>	A filename.
Values	<i>pathname</i>	The filename that was supplied.
Description		The <code>save-tags-database</code> function saves the current internal dspec database into the file given by <i>pathname</i> . The file can then be used in the variable <code>*active-finders*</code> .
See also		<code>*active-finders*</code>

		<i>Function</i>
Summary		A pre-defined form parser.
Package		<code>dspec</code>
Signature		<code>single-form-form-parser top-level-form getter => dspec</code>
Arguments	<i>top-level-form</i>	A top level defining form.
	<i>getter</i>	The subform getter function.
Values	<i>dspec</i>	A dspec.
Description		The function <code>single-form-form-parser</code> is a predefined form parser for use with <code>define-form-parser</code> . The parser consumes one subform and returns a dspec made from the defining form and the subform. This can be used in the common case where a defining form has a name that follows the defin-

ing macro and the dspec class is the same as the defining macro, for example `defclass`.

`single-form-form-parser` is the predefined parser for `defvar`, `defparameter`, `defconstant`, `define-symbol-macro`, `define-compiler-macro`, `deftype`, `defsetf`, `define-setf-expander`, `defpackage`, `defclass`, `define-condition` and `define-method-combination` top level forms. It is also the parser for various LispWorks extensions such as `defsystem`.

You can define it to be the parser for your defining forms. using `define-form-parser`.

See also `define-form-parser`

	<i>Function</i>
<code>single-form-with-options-form-parser</code>	
Summary	A pre-defined form parser.
Package	<code>dspec</code>
Signature	<code>single-form-with-options-form-parser top-level-form getter => dspec</code>
Arguments	<code>top-level-form</code> A top level defining form. <code>getter</code> The subform getter function.
Values	<code>dspec</code> A dspec.
Description	The function <code>single-form-with-options-form-parser</code> is a predefined form parser for use with <code>define-form-parser</code> . The parser consumes one subform and returns a dspec made from the defining form and either the first element of the subform if it is a cons or the subform itself otherwise. This can be used in the common case where a defining form has a name with options that follows the defining macro and the

`dspec` class is the same as the defining macro, for example `defstruct`.

`single-form-with-options-form-parser` is the predefined parser for `defstruct`, `fli:define-foreign-function`, `fli:define-foreign-variable`, `fli:define-c-struct`, `fli:define-c-union`, `fli:define-c-enum` and `fli:define-c-typedef` forms.

You can define it to be the parser for your defining forms. using `define-form-parser`.

See also `define-form-parser`

traceable-dspec-p *Function*

Summary Tests whether definition can be traced.

Package `dspec`

Signature `traceable-dspec-p dspec => result`

Arguments `dspec` A `dspec`.

Values `result` A generalised boolean.

Description The function `traceable-dspec-p` determines whether the `dspec` `dspec` denotes a definition that can be traced using the Common Lisp macro `trace`.

`dspec` must not be a local `dspec`, and must be defined, according to `dspec-defined-p`. The result does not depend on whether `dspec` is currently traced.

Example	<pre>CL-USER 67 > (dspec:traceable-dspec-p '(subfunction foo bar)) NIL</pre> <pre>CL-USER 68 > (dspec:traceable-dspec-p '(defun open)) OPEN</pre>
---------	---

tracing-enabled-p *Function*

Summary	Gets and sets the global tracing state..
Package	dspec
Signature	$\text{tracing-enabled-p} \Rightarrow \text{enabledp}$ $(\text{setf tracing-enabled-p}) \text{ enabledp} \Rightarrow \text{enabledp}$
Values	<i>enabledp</i> A generalized boolean.
Description	<p>The function tracing-enabled-p determines whether tracing (by the Common Lisp macro trace) is currently on. This is independent of whether any functions are currently traced.</p> <p>The function (setf tracing-enabled-p) switches tracing on or off according to the value of <i>enabledp</i>. This does not affect the list of functions that are currently traced.</p>
See also	trace tracing-state

tracing-state *Function*

Summary	Gets the current trace details.
Package	dspec
Signature	tracing-state &optional dspec => state

Signature	<code>(setf tracing-state) state &optional dspec => state</code>	
Arguments	<i>dspec</i>	A dspec.
Values	<i>state</i>	A list.
Description	<p>The function <code>tracing-state</code> returns a listing describing the current state of the tracing system. It shows the current tracing state for the dspec <i>dspec</i>, or for all traced definitions if <i>dspec</i> is not supplied.</p> <p>The result <i>state</i> is a list each element of which is a list whose car is a dspec naming the traced definition and whose cdr is the additional trace options. Note that <code>tracing-state</code> returns more information than is returned by <code>trace</code>. It is useful for preserving a complex set of traces.</p>	
	<p>The function <code>(setf tracing-state)</code> sets the state of the tracing system. It changes the current tracing state for the dspec <i>dspec</i>, or for all traced definitions if <i>dspec</i> is not supplied.</p> <p><code>(setf tracing-state)</code> can be used to switch between different sets of traces. Note however that turning tracing on or off is better done using <code>tracing-enabled-p</code>.</p>	
See also	<p><code>trace</code> <code>tracing-enabled-p</code></p>	

The EXTERNAL-FORMAT Package

This chapter describes symbols available in the `EXTERNAL-FORMAT` package.

char-external-code	<i>Function</i>				
Summary	Returns the code of a character in the specified character set.				
Package	<code>external-format</code>				
Signature	<code>char-external-code char set => code</code>				
Arguments	<table><tr><td><i>char</i></td><td>The character whose code you wish to return.</td></tr><tr><td><i>set</i></td><td>A character set. Legal values for <i>set</i> are <code>:unicode</code>, <code>:latin-1</code>, <code>:ascii</code>, <code>:macos-roman</code>, <code>:jis-x-208</code>, <code>:jis-x-212</code>, <code>:euc-jp</code> and <code>:sjis</code>. Additionally, on Windows, <i>set</i> can be a valid Windows code page identifier.</td></tr></table>	<i>char</i>	The character whose code you wish to return.	<i>set</i>	A character set. Legal values for <i>set</i> are <code>:unicode</code> , <code>:latin-1</code> , <code>:ascii</code> , <code>:macos-roman</code> , <code>:jis-x-208</code> , <code>:jis-x-212</code> , <code>:euc-jp</code> and <code>:sjis</code> . Additionally, on Windows, <i>set</i> can be a valid Windows code page identifier.
<i>char</i>	The character whose code you wish to return.				
<i>set</i>	A character set. Legal values for <i>set</i> are <code>:unicode</code> , <code>:latin-1</code> , <code>:ascii</code> , <code>:macos-roman</code> , <code>:jis-x-208</code> , <code>:jis-x-212</code> , <code>:euc-jp</code> and <code>:sjis</code> . Additionally, on Windows, <i>set</i> can be a valid Windows code page identifier.				
Values	<i>code</i> The code of <i>char</i> in the character set <i>set</i> . An integer.				

Description Returns the code of the character *char* in the coded character set specified by *set*, or `nil`, if there is no encoding. Note that a coded character set is not the same thing as an external format.

For the *set* parameter, the `:jis-*` codes are KUTEN indexes (from the 1990 version of these standards) encoded as

```
(+ (* 100 row) column)
```

`:euc-jp` is the complete two-byte format encoded as

```
(+ (* 256 first-byte) second-byte)
```

`:sjis` is Shift-JIS encoded in the same way. Strictly speaking, EUC and Shift-JIS are not coded character sets, but encodings of the JIS sets, but the encoding is easily expressed as an integer, so the same interface to it is used.

See also `find-external-char`

decode-external-string

Function

Summary Decodes a binary vector to make a string.

Package `external-format`

Signature `decode-external-string vector external-format &key start end => string`

Arguments `vector` A binary vector.

`external-format` An external format spec.

`start, end` Bounding index designators of `vector`.

Values `string` A string.

Description	The function <code>decode-lisp-string</code> decodes the integers in the part of the vector <code>vector</code> bounded by <code>start</code> and <code>end</code> using encoding <code>external-format</code> to make a string <code>string</code> . The element type of <code>vector</code> does not need to match the <code>external-format-foreign-type</code> of <code>external-format</code> .
Compatibility note	This function exists in LispWorks 5.0 but is not documented and does not take the <code>:start</code> and <code>:end</code> arguments. Also, it was inefficient prior to LispWorks 5.0.1.
See also	<code>encode-lisp-string</code>

	<i>Function</i>						
encode-lisp-string	<i>Function</i>						
Summary	Converts a string to an encoded binary vector.						
Package	<code>external-format</code>						
Signature	<code>encode-lisp-string string external-format &key start end => vector</code>						
Arguments	<table> <tr> <td><code>string</code></td> <td>A string.</td> </tr> <tr> <td><code>external-format</code></td> <td>An external format spec.</td> </tr> <tr> <td><code>start, end</code></td> <td>Bounding index designators of <code>string</code>.</td> </tr> </table>	<code>string</code>	A string.	<code>external-format</code>	An external format spec.	<code>start, end</code>	Bounding index designators of <code>string</code> .
<code>string</code>	A string.						
<code>external-format</code>	An external format spec.						
<code>start, end</code>	Bounding index designators of <code>string</code> .						
Values	<code>vector</code> A binary vector.						
Description	<p>The function <code>encode-lisp-string</code> converts the part of <code>string</code> bounded by <code>start</code> and <code>end</code> to a binary vector <code>vector</code> encoded in encoding <code>external-format</code>.</p> <p>The element type of <code>vector</code> matches the <code>external-format-foreign-type</code> of <code>external-format</code>.</p>						

Compatibility note This function exists in LispWorks 5.0 but is not documented and does not take the `:start` and `:end` arguments. Also, it was inefficient prior to LispWorks 5.0.1.

See also [decode-external-string](#)

external-format-error *Condition*

Summary The condition class `external-format-error` is the superclass of all errors relating to external formats.

Package `external-format`

Superclasses `error`

Initargs `:name` The name of the external format involved.

Description The class `external-format-error` provides a slot for the name of external format involved: this is the fully expanded form of the specification with all the parameters filled in. It is also useful for users who want to set up a handler for encoding errors.

external-format-foreign-type *Function*

Summary Returns a type specifier for the integers handled by a specified external format.

Package `external-format`

Signature `external-format-foreign-type external-format => type-specifier`

Arguments `external-format` An external character format.

Values	<i>type-specifier</i>	A type specifier describing the integer types handled by <i>external-format</i> .
Description		Takes the name of an external format, and returns a Lisp type specifier for the type of integers that the external format handles on the foreign side.
See also		<code>external-format-type</code>

external-format-type *Function*

Summary	Returns a type specifier for the characters handled by a specified external format.	
Package	<code>external-format</code>	
Signature	<code>external-format-type external-format => type-specifier</code>	
Arguments	<i>external-format</i> An external character format.	
Values	<i>type-specifier</i> A type specifier describing the character types handled by <i>external-format</i> .	
Description	Takes the name of an external format, and returns a type specifier for the type of characters that the external format handles on the Lisp side.	
See also	<code>external-format-foreign-type</code>	

find-external-char *Function*

Summary	Returns the character of a given code in a specified character set.	
Package	<code>external-format</code>	

Signature	<code>find-external-char code set => char</code>	
Arguments	<code>code</code>	A character code. This is an integer.
	<code>set</code>	A character set. Legal values for <code>set</code> are <code>:unicode</code> , <code>:latin-1</code> , <code>:ascii</code> , <code>:macos-roman</code> , <code>:jis-x-208</code> , <code>:jis-x-212</code> , <code>:euc-jp</code> and <code>:sjis</code> . Additionally, on Windows, <code>set</code> can be a valid Windows code page identifier.
Values	<code>char</code>	The character represented by <code>code</code> . If <code>code</code> is not a legal code in the specified set, the return value is undefined.
Description	<p>Returns the character that has the code <code>code</code> (an integer) in the coded character set specified by <code>set</code>, or <code>nil</code>, if that character is not represented in the Lisp character set. Note that a coded character set is not the same thing as an external format.</p> <p>For the <code>set</code> parameter, the <code>:jis-*</code> codes are KUTEN indexes (from the 1990 version of these standards) encoded as</p> <pre>(+ (* 100 row) column)</pre> <p><code>:euc-jp</code> is the complete two-byte format encoded as</p> <pre>(+ (* 256 first-byte) second-byte)</pre> <p><code>:sjis</code> is Shift-JIS encoded in the same way. Strictly speaking, EUC and Shift-JIS are not coded character sets, but encodings of the JIS sets, but the encoding is easily expressed as an integer, so the same interface to it is used.</p>	
See also	<code>char-external-code</code>	

valid-external-format-p	<i>Function</i>
Summary	Tests whether an external format spec is valid.

Package	<code>external-format</code>	
Signature	<code>valid-external-format-p <i>ef-spec</i> &optional <i>env</i> => <i>bool</i></code>	
Arguments	<i>ef-spec</i>	An external format spec.
	<i>env</i>	An environment across which the spec should apply.
Values	<i>bool</i>	<code>t</code> if <i>ef-spec</i> is a valid spec; <code>nil</code> otherwise.
Description	This predicate tests whether the external format spec given in <i>ef-spec</i> is valid (in the environment <i>env</i>).	
Example	<code>(valid-external-format-p '(:Unicode :eol-style :lf))</code>	

8

The HCL Package

This chapter describes symbols available in the `HCL` package. This package is used by default. Its symbols are visible in the `CL-USER` package.

		<i>Function</i>
Summary	Adds a function to perform a special action during garbage collection.	
Package	<code>hcl</code>	
Signature	<code>add-special-free-action <i>function</i> => <i>function-list</i></code>	
Arguments	<i>function</i>	A symbol naming a function of one argument.
Values	<i>function-list</i>	A list of the functions currently called to perform special actions, including the one just added.
Description		When some objects are garbage collected, you may require a “special action” to be performed as well. <code>add-special-free-</code>

`action` adds the function `function` to perform the special action. Note that the function is applied to all objects flagged for special-free-action, so the function `function` should check for the object's type, so that it only affects relevant objects.

The functions `flag-special-free-action` and `flag-not-special-free-action` flag and unflag objects for action.

Example `(add-special-free-action 'free-my-app)`

See also `remove-special-free-action`
`flag-special-free-action`
`flag-not-special-free-action`

add-symbol-profiler *Function*

Summary Adds a symbol to the list of profiled symbols.

Package `hcl`

Signature `add-symbol-profiler symbol => nil`

Arguments `symbol` A symbol to be added to the `*profile-symbol-list*`.

Values Returns `nil`.

Description `add-symbol-profiler` adds a symbol to `*profile-symbol-list*`, the list of profiled symbols.

See also `*profile-symbol-list*`
`remove-symbol-profiler`

allocation-in-gen-num Macro

Summary	Allocates objects from a specified generation within the scope of evaluating a number of forms in 32-bit LispWorks.	
Package	<code>hcl</code>	
Signature	<code>allocation-in-gen-num <i>gen-num</i> &body <i>body</i> => <i>result</i></code>	
Arguments	<i>gen-num</i>	An integer, which if out of range for a valid generation number is rounded either to the youngest or oldest generation. If <i>gen-num</i> is negative, the specified generation is: the highest generation number + 1 - <i>gen-num</i> , so that an argument of -1 specifies the highest generation number.
	<i>body</i>	The forms to be evaluated while the allocation generation has been temporarily set to <i>gen-num</i> .
Values	<i>result</i>	The result of evaluating body.
Description	<p>Allocates objects from a specified generation during the extent of the evaluation of the body forms.</p> <p>Normally objects are allocated from the first (youngest) generation, which assumes that they are short-lived. The storage allocator and garbage collector perform better if allocation of large numbers of non-ephemeral objects is done explicitly into a generation other than the youngest.</p> <p>Note: this macro is implemented only in 32-bit LispWorks. In 64-bit implementations, use <code>apply-with-allocation-in-gen-num</code> or the <code>:allocation</code> argument to <code>make-array</code> instead.</p>	

Examples	<pre>(allocation-in-gen-num 1 (setq tab (make-hash-table :size 1200 :test 'eq) arr (make-array 20)))</pre>
See also	<code>apply-with-allocation-in-gen-num</code> <code>make-array</code> <code>set-default-generation</code> <code>get-default-generation</code> <code>*symbol-alloc-gen-num*</code>

	<i>Function</i>
avoid-gc	
Summary	Avoids garbage collection if possible in 32-bit LispWorks.
Package	<code>hcl</code>
Signature	<code>avoid-gc => previous-results</code>
Arguments	None.
Values	The function returns the previous settings of <code>minimum-for-sweep</code> , <code>maximum-overflow</code> and <code>minimum-overflow</code> . (See <code>set-gc-parameters</code> for details of these.)
Description	<p><code>avoid-gc</code> sets various internal parameters so that garbage collection is avoided as far as possible.</p> <p>This can be useful with non-interactive programs.</p> <p>If you use <code>avoid-gc</code>, use <code>normal-gc</code> later to reset the parameters to their default settings.</p> <p>Note: <code>avoid-gc</code> is implemented only in 32-bit LispWorks. It is not relevant to the Memory Management API in 64-bit implementations. In 64-bit implementations, you can use <code>set-default-segment-size</code> to increase the default size of seg-</p>

ments in the lower generations (typically generations 0 and 1). This will lead to less frequent garbage collections.

See also

gc-if-needed
normal-gc
set-gc-parameters
set-default-segment-size
without-interrupts

binds-who *Function*

Summary Lists special variables bound by a definition.

Package **hcl**

Signature **binds-who** *function => result*

Arguments *function* A symbol or a function dspec.

Values *result* A list.

Description The function **binds-who** returns a list of the special variables bound by the definition named by *function*.

Note: The cross-referencing information used by **binds-who** is generated when code is compiled with source-level debugging switched on.

See also **toggle-source-debugging**
who-binds

block-promotion *Macro*

Summary Prevents promotion of objects into generation 2 during the execution of *body*.

Package	<code>hcl</code>
Signature	<code>block-promotion &body body => result</code>
Arguments	<code>body</code> Forms executed as an implicit <code>progn</code> .
Values	<code>result</code> The result of evaluating the final form in <code>body</code> .
Description	<p>The macro <code>block-promotion</code> executes <code>body</code> and prevents promotion of objects into generation 2 during this execution. After <code>body</code> is executed, generations 0 and 1 are collected.</p> <p>This is useful when a significant number of transient objects actually survive all the garbage collections on generation 1. These would normally then be promoted and, by default, never get collected. In such a situation, (<code>mark-and-sweep 2</code>) will free a large amount of space in generation 2.</p> <p><code>block-promotion</code> can be thought of as doing <code>set-promotion-count</code> on generation 1 with an infinite <code>count</code>, for the duration of <code>body</code>.</p> <p><code>block-promotion</code> is suitable only for use in particular operations that are known to create such relatively long-lived, but transient, objects. In typical uses these are objects that live for a few seconds to several hours. An example usage is Lisp-Works <code>compile-file</code>, to ensure the transient compile-time data gets collected.</p> <p><code>block-promotion</code> has global scope and hence may not be useful in an application such as a multi-threaded server. During the execution of <code>body</code>, generation 1 grows to accomodate all the allocated data, which may have some negative effects on the behaviour of the system, in particular on its interactive response.</p> <p>Note: symbols and process stacks are allocated in generation 2 or 3 (see <code>*symbol-alloc-gen-num*</code>) hence <code>block-promotion</code> cannot prevent these getting into that generation.</p>

`allocation-in-gen-num` can also cause allocation in higher generations.

Note: in 64-bit LispWorks, `block-promotion` is implemented using `set-blocking-gen-num`.

See also [allocation-in-gen-num](#)
[mark-and-sweep](#)
[set-promotion-count](#)

building-universal-intermediate-p *Function*

Summary	Used in a build script to determine if LispWorks is building an intermediate image when making a universal binary.
Package	hcl
Signature	<code>building-universal-intermediate-p => intermediatep</code>
Arguments	None
Values	<i>intermediatep</i> A boolean.
Description	<p>The function <code>building-universal-intermediate-p</code> can be used in a build script to determine if it is being executed to build one of the architectures of a universal binary.</p> <p>The return value <i>intermediatep</i> is <code>nil</code> in most cases. It will be <code>t</code> only when building an intermediate image for the purpose of building a universal binary, either by <code>save-universal-from-script</code> or the Application Builder (see the <i>Common LispWorks User Guide</i>).</p> <p>This is useful if there are some configuration that should be done only in a universal binary image but not in a mono-architecture ("thin") image. Whether the intermediate image will be the Intel or the PowerPC part of the universal binary can be determined by checking <code>*features*</code>.</p>

On architectures that do not have universal binaries, this function always returns `nil`.

See also `save-universal-from-script`
`save-argument-real-p`

calls-who	<i>Function</i>
Summary	Lists functions called by a function.
Package	<code>hcl</code>
Signature	<code>calls-who dspec => callees</code>
Arguments	<code>dspec</code> A <code>dspec</code> .
Values	<code>callees</code> A list.
Description	The function <code>calls-who</code> returns a list of the <code>dspecs</code> naming the functions called by the function named by <code>dspec</code> . See also the editor commands <code>List Callees</code> , and <code>Show Paths From</code> .
	Note: The cross-referencing information used by <code>calls-who</code> is generated when code is compiled with source-level debugging switched on.
Example	<code>(calls-who '(method foo (string)))</code>
See also	<code>toggle-source-debugging</code> <code>who-calls</code>

cd	<i>Macro</i>
Summary	Changes the current directory.

Package	<code>hcl</code>	
Signature	<code>cd &optional directory => current-dir</code>	
Arguments	<code>directory</code>	A pathname designator specifying the new directory.
Values	<code>current-dir</code>	A physical pathname.
Description		The macro <code>cd</code> changes the current directory to that specified by <code>directory</code> . <code>directory</code> may be an absolute or relative pathname, and defaults to the string " <code>~/</code> ".
See also		<code>change-directory</code> <code>get-working-directory</code>

change-directory *Function*

Summary	Changes the current directory.	
Package	<code>hcl</code>	
Signature	<code>change-directory directory => current-dir</code>	
Arguments	<code>directory</code>	A pathname designator specifying the new directory.
Values	<code>current-dir</code>	A physical pathname.
Description		<code>change-directory</code> changes the current directory to that specified by <code>directory</code> . <code>directory</code> may be an absolute or relative pathname. Use <code>get-working-directory</code> to find the current directory.
See also		<code>cd</code> <code>get-working-directory</code>

		<i>Function</i>
Summary		Provides information about the fragmentation in a generation in 32-bit LispWorks.
Package		<code>hcl</code>
Signature		<code>check-fragmentation gen-num => total-free, total-small-blocks, total-large-blocks</code>
Arguments	<i>gen-num</i>	0 for the most recent generation, 1 for the most recent two generations, and so on up to a maximum (usually 3). Numbers outside this range signal an error.
Values	<i>total-free</i>	Total free space in the generation.
	<i>total-small-blocks</i>	Amount of free space in the generation which is available in blocks of 512 bytes or larger.
	<i>total-large-blocks</i>	Amount of free space in the generation which is available in blocks of 4096 bytes or larger.
Description		The latter two values give indication of the level of fragmentation in the generation. This information can be used, for example, to decide whether to call <code>try-move-in-generation</code> . Note: <code>check-fragmentation</code> is implemented only in 32-bit LispWorks. It is not relevant to the Memory Management API in 64-bit implementations, where <code>gen-num-segments-fragmentation-state</code> is available instead.
See also		<code>try-compact-in-generation</code> <code>try-move-in-generation</code>

	<i>Function</i>
Summary	Frees memory and reduces the size of the image, if possible.
Package	<code>hcl</code>
Signature	<code>clean-down &optional full => new-size</code>
Arguments	<i>full</i> controls whether to operate on the highest generation. The default is <code>t</code> .
Values	<i>new-size</i> The new size of the image, after reduction.
Description	<p>Tries to free as much memory as possible and then reduce the size of the image as much as possible, and also move all the allocated objects to an old generation.</p> <p>If <i>full</i> is <code>t</code>, <code>clean-down</code> does a mark and sweep on generation 3, promotes all the objects into generation 3, deletes the empty segments and tries to reduce the image size. This is called by default before saving an image.</p> <p>If <i>full</i> is <code>nil</code>, <code>clean-down</code> does a mark and sweep on generation 2, promotes all the objects to generation 2 and tries to reduce the size of all generations up to 2, but does not touch generation 3.</p> <p><code>clean-down</code> may fail to delete empty segments if there are static segments in high address space.</p> <p>Note: in 64-bit LispWorks, <code>clean-down</code> is implemented as if by</p> <pre>(gc-generation 7 :coalesce t)</pre> <p>though you can use <code>gc-generation</code> directly for better control.</p>
See also	<code>gc-generation</code> <code>save-image</code>

	<i>Function</i>
clean-generation-0	
Summary	Attempts to promote all objects from generation zero into generation one, thereby clearing generation zero, in 32-bit LispWorks.
Package	<code>hcl</code>
Signature	<code>clean-generation-0 => 1</code>
Arguments	None
Values	Returns the value 1.
Description	<p>This is useful when passing from a phase of creating long-lived data to a phase of mostly ephemeral data, for example, the end of loading an application and the start of its use.</p> <p>Note: The function may not be very useful, as it may be more efficient to directly allocate the objects in a particular generation in the first place, using <code>allocation-in-gen-num</code> or <code>set-default-generation</code>.</p> <p>Note: <code>clean-generation-0</code> is implemented only in 32-bit LispWorks. It is not relevant to the Memory Management API in 64-bit implementations, where the same effect can be obtained by a call (<code>gc-generation 0</code>).</p>
Example	<pre>; allocate lots of non-ephemeral objects ; (clean-generation-0)</pre>
See also	<code>allocation-in-gen-num</code> <code>collect-generation-2</code> <code>collect-highest-generation</code> <code>expand-generation-1</code> <code>gc-generation</code> <code>set-promotion-count</code>

	<i>Function</i>
Summary	Controls whether generation 2 is garbage collected in 32-bit LispWorks.
Package	<code>hcl</code>
Signature	<code>collect-generation-2 on => size</code>
Arguments	<p><code>on</code> If <code>on</code> is <code>nil</code>, generation 2 is not garbage collected. If <code>on</code> is <code>t</code>, the generation is garbage collected.</p>
Values	<code>size</code> The current size of the image.
Description	Controls whether generation 2 is garbage collected. (Generation 2 normally holds long-lived objects created dynamically.)
	Note: <code>collect-generation-2</code> is implemented only in 32-bit LispWorks. It is not relevant to the Memory Management API in 64-bit implementations, where you can use <code>set-blocking-gen-num</code> instead.
See also	<code>clean-generation-0</code> <code>collect-highest-generation</code> <code>expand-generation-1</code> <code>set-blocking-gen-num</code> <code>set-promotion-count</code>

	<i>Function</i>
Summary	Controls whether the top generation is garbage-collected in 32-bit LispWorks.
Package	<code>hcl</code>

Signature	<code>collect-highest-generation flag</code>	
Arguments	<code>flag</code>	If <code>flag</code> is non- <code>nil</code> , the top generation is collected; if <code>flag</code> is any other value, the top generation is not collected. The default is <code>nil</code> .
Values	<code>collect-highest-generation</code> returns no values.	
Description	<p>Note: <code>collect-highest-generation</code> is implemented only in 32-bit LispWorks. It is not relevant to the Memory Management API in 64-bit implementations.</p>	
See also	avoid-gc clean-generation-0 collect-generation-2 expand-generation-1 normal-gc	

compiler-break-on-error *Variable*

Summary	Controls whether <code>compile-file</code> handles compilation errors.
Package	<code>hcl</code>
Initial Value	<code>nil</code>
Description	<p>If an error occurs during compilation of a form by <code>compile-file</code>, an error handler normally causes the compilation of that form to be skipped, and the error is reported later.</p> <p>When <code>*compiler-break-on-error*</code> is non-<code>nil</code>, an error during compilation by <code>compile-file</code> is signaled and the debugger is entered.</p>
See also	<code>compile-file</code>

	<i>Function</i>
Summary	Compiles a Lisp source file if it is newer than the corresponding fasl file.
Package	<code>hcl</code>
Signature	<code>compile-file-if-needed <i>input-pathname</i> &key <i>output-file</i> <i>load</i> &<i>allow-other-keys</i> => <i>output-truename</i>, <i>warnings-p</i>, <i>failure-p</i></code>
Arguments	<p><i>input-pathname</i> A pathname designator.</p> <p><i>output-file</i> A pathname designator.</p> <p><i>load</i> A generalized boolean.</p>
Values	<p><i>output-truename</i> A pathname or <code>nil</code>.</p> <p><i>warnings-p</i> A generalized boolean.</p> <p><i>failure-p</i> A generalized boolean.</p>
Description	<p>The function <code>compile-file-if-needed</code> compares the <code>file-write-date</code> of the source file named by <i>input-pathname</i> with the <code>file-write-date</code> of the appropriate fasl file (as computed by <code>compile-file-pathname</code> from <i>input-pathname</i> and <i>output-file</i>).</p> <p>If the fasl file does not exist or is older than <i>input-pathname</i>, then <code>compile-file</code> is called with <i>input-pathname</i>, <i>output-file</i>, <i>load</i> and any other arguments passed., and the values returned are those returned from <code>compile-file</code>.</p> <p>Otherwise, if <i>load</i> is true <code>compile-file-if-needed</code> loads the fasl file and returns <code>nil</code>, and if <i>load</i> is <code>nil</code> it simply returns <code>nil</code>.</p>

Example	<pre> CL-USER 19 > (compile-file-if-needed "H:/tmp/foo.lisp" :output-file "C:/temp/") ;; Compiling file H:/tmp/foo.lisp ... ;; Safety = 3, Speed = 1, Space = 1, Float = 1, Interruptible = 0 ;; Compilation speed = 1, Debug = 2, Fixnum safety = 3 ;; Source level debugging is off ;; Source file recording is on ;; Cross referencing is off ; (TOP-LEVEL-FORM 1) ; (TOP-LEVEL-FORM 2) ; (TOP-LEVEL-FORM 3) ; FOO ; BAR #P"C:/temp/foo.ofasl" NIL NIL CL-USER 20 > (compile-file-if-needed "H:/tmp/foo.lisp" :output-file "C:/temp/" :load t) ; Loading fasl file C:\temp\foo.ofasl NIL </pre>
---------	---

See also [compile-file](#)

copy-to-weak-simple-vector

Function

Summary	Creates a weak vector with the same contents as the supplied vector.
Package	<code>hcl</code>
Signature	<code>copy-to-weak-simple-vector vector-t => weak-vector</code>
Arguments	<code>vector-t</code> An array of type (<code>vector t</code>).
Values	<code>weak-vector</code> A weak array of type (<code>vector t</code>).

Description	The function <code>copy-to-weak-simple-vector</code> creates and returns a weak vector with the same contents as the argument <i>vector-t</i> . Apart from the checking of arguments, this is equivalent to:
	<pre>(replace (make-array (length vector-t) :weak t) vector-t)</pre>
See also	<code>make-array</code> <code>set-array-weak</code>

create-universal-binary *Function*

Summary	Creates a universal binary from two mono-architecture Lisp-Works images.	
Package	<code>hcl</code>	
Signature	<code>create-universal-binary target-image src-image1 src-image2 => target-image</code>	
Arguments	<p><i>target-image</i> A pathname designator.</p> <p><i>src-image1</i> A pathname designator.</p> <p><i>src-image2</i> A pathname designator.</p>	
Values	<i>target-image</i> A pathname designator.	
Description	<p>This function is intended for advanced use. See the function <code>save-universal-from-script</code> for a simpler way to create a universal binary.</p> <p>The function <code>create-universal-binary</code> writes a universal binary to the file <i>target-image</i> from the saved image files <i>src-image1</i> and <i>src-image2</i>. The value of <i>target-image</i> is returned.</p>	

The source images *src-image1* and *src-image2* must both be LispWorks for Macintosh mono-architecture ("thin") images and one should be for the Intel architecture and the other for the PowerPC architecture (the order is immaterial). For example, they could have been created by `save-image` or `deliver`.

Note: The function `create-universal-binary` checks that *src-image1* and *src-image2* are LispWorks images of different architectures, but it does not check how they were saved or how similar they are. You need to ensure that both images contain the same functionality.

Note: The function `create-universal-binary` can only be called from a LispWorks for Macintosh image that is itself a universal binary, such as the distributed image.

Example	Suppose that you have saved two images, <code>my-application-intel</code> and <code>my-application-powerpc</code> , which contains the same application code loaded on an Intel Macintosh and a PowerPC Macintosh. The following command will combine them into a universal binary <code>my-application</code> that will run on both kinds of Macintosh:
	<pre>(create-universal-binary "my-application" "my-application-intel" "my-application-powerpc")</pre>

See also	<code>save-image</code> <code>save-universal-from-script</code>
----------	--

current-stack-length

Function

Summary Returns the size of the current stack.

Package `hcl`

Signature `current-stack-length => stack-size`

Arguments	None
Values	<p><code>stack-size</code> The current size of the stack, in 32 bit words (in 32-bit implementations) or 64-bit words (in 64-bit implementations).</p>
Example	<code>(current-stack-length) => 16000</code>
Compatibility Note	In LispWorks 4.4 and previous on Windows and Linux platforms, <code>current-stack-length</code> was not implemented. This is fixed in LispWorks 5.0 and later.
See also	<code>extend-current-stack</code> <code>*sg-default-size*</code>

default-package-use-list *Variable*

Summary	List of packages that newly created packages use by default.
Package	<code>hcl</code>
Initial Value	<code>("CL" "LW" "HCL")</code>
Description	This variable is the default value of the <code>:use</code> keyword to <code>defpackage</code> , which specifies which existing packages the package being defined inherits from.

default-profiler-collapse *Variable*

Summary	Controls collapsing of the profile tree.
Package	<code>hcl</code>
Initial Value	<code>nil</code>

Description	The variable <code>*default-profiler-collapse*</code> is a boolean indicating whether the profile tree should collapse functions with only one child function. The default value is <code>nil</code> .
See also	<code>print-profile-list</code> <code>set-up-profiler</code>

default-profiler-cutoff *Variable*

Summary	The minimum percentage that the profiler will display in the output tree.
Package	<code>hcl</code>
Initial Value	<code>0</code>
Description	The variable <code>*default-profiler-cutoff*</code> is the minimum percentage (0 to 100) that the profiler will display in its output tree. Functions below this percentage will not be displayed. The initial value is 0, meaning display everything.
See also	<code>print-profile-list</code> <code>set-up-profiler</code>

default-profiler-limit *Variable*

Summary	The maximum number of lines of output that are printed during profiling.
Package	<code>hcl</code>
Initial Value	<code>100,000,000</code>
Description	<code>*default-profiler-limit*</code> is the maximum number of lines of output in profile results. The default value is large to

ensure that you receive all possible output requested.

default-profiler-limit only counts output lines for functions that are actually called during profiling. Therefore, if ***default-profiler-limit*** is 19, and 20 functions were profiled, you would receive full output if one or more of the functions were not actually called during profiling.

See also [print-profile-list](#)
[set-up-profiler](#)

default-profiler-sort

Variable

Summary The default sorting style for the profiler.

Package hcl

Initial Value :profile

Description The variable `*default-profiler-limit*` controls which column of the profiler's columnar report is used for sorting.

The value can be one of :profile, :call or :top.

See also [print-profile-list](#)
[set-up-profiler](#)

delete-advice

Macro

Summary Removes a piece of advice.

Package hcl

Signature **delete-advice** *dspec name => nil*

dspec ::= *fn-name* | *macro-name* |
 (*clos::method* *generic-fn-name* [(*class**)])

Arguments	<code>dspec</code>	Specifies the functional definition to which the piece of advice belongs. The specification contains the name of the associated function. In the case of a method the list of classes is used to identify from which particular method the advice should come. This list must correspond exactly with the classes corresponding to the specialized parameters for some method belonging to the generic function.
	<code>name</code>	A symbol naming the piece of advice to be removed. Since several pieces of advice may be attached to a single functional definition, the name is necessary to indicate which one is to be removed.
Values	<code>delete-advice</code> returns <code>nil</code> .	
Description	<p><code>delete-advice</code> is used to remove a piece of advice. Advice is a way of altering the behavior of functions. Pieces of advice are associated with a function using <code>defadvice</code>. They define additional actions to be performed when the function is invoked, or alternative code to be performed instead of the function, which may or may not access the original definition. As well as being attached to ordinary functions, advice may be attached to methods and to macros (in this case it is in fact associated with the macro's expansion function).</p> <p><code>remove-advice</code> is a function, identical in effect to <code>delete-advice</code>, except that you need to quote the arguments.</p>	
Notes	<code>delete-advice</code> is an extension to Common Lisp.	
See also	<code>defadvice</code> <code>remove-advice</code>	

disable-trace	<i>Variable</i>
Summary	Controls tracing.
Package	<code>hcl</code>
Initial Value	<code>nil</code>
Description	<code>*disable-trace*</code> controls tracing without affecting the tracing state. If it is set to <code>t</code> then tracing is switched off, but this does not call <code>untrace</code> . When the value of <code>*disable-trace*</code> is restored to <code>nil</code> , tracing continues as before.
Notes	<code>*disable-trace*</code> is an extension to Common Lisp.
See also	<code>trace</code>

dump-form	<i>Function</i>				
Summary	Dumps selected forms to a stream.				
Package	<code>hcl</code>				
Signature	<code>dump-form form stream => nil</code>				
Arguments	<table> <tr> <td><i>form</i></td> <td>Form to be dumped.</td> </tr> <tr> <td><i>stream</i></td> <td>Stream form is to be dumped to.</td> </tr> </table>	<i>form</i>	Form to be dumped.	<i>stream</i>	Stream form is to be dumped to.
<i>form</i>	Form to be dumped.				
<i>stream</i>	Stream form is to be dumped to.				
Values	Returns <code>nil</code> .				
Description	<p><code>dump-form</code> is used in conjunction with <code>with-output-to-fasl-file</code> to dump selected forms. A dumped form is evaluated when loaded using <code>load-data-file</code>.</p> <p>See <code>with-output-to-fasl-file</code> for more details.</p>				

See also **dump-forms-to-file**
with-output-to-fasl-file

dump-forms-to-file *Function*

Summary Dumps specified forms to a fasl file.

Package **hcl**

Signature **dump-forms-to-file pathname forms => nil**

Arguments *pathname* Name of the fasl file to be created.
forms Forms to be dumped.

Values Returns **nil**.

Description **dump-forms-to-file** dumps specified forms to a fasl file. Use the Common Lisp functions **make-load-form** and **make-load-form-saving-slots** to control the dumping of forms.

The best way to specify the file type of the output file is to use **compile-file-pathname** as in the example below. The file types currently used by LispWorks for fasl files are listed in **compile-file**.

If the file *pathname* already exists, it is superseded.

A fasl file created using **dump-forms-to-file** must be loaded only by **load-data-file**, and not by **load**.

Example

```
(defclass my-class () ((a :initarg :a :accessor my-a)))  
  
(defmethod make-load-form ((self my-class) &optional  
environment)  
  (declare (ignore environment))  
  `(make-instance ',(class-name (class-of self))  
    :a ',(my-a self)))  
  
(setq *my-instance* (make-instance 'my-class :a 42))
```

```
(dump-forms-to-file
  (compile-file-pathname "my-instance")
  (list `(setq *my-instance* ,*my-instance*)))
```

In another session, with the same definition of `my-class`, loading the file "my-instance" using `load-data-file` will create an equivalent instance of `my-class`:

```
(sys:load-data-file
  (compile-file-pathname "my-instance"))
```

See also [with-output-to-fasl-file](#)

enlarge-generation *Function*

Summary Enlarges a generation in 32-bit LispWorks.

Package `hcl`

Signature `enlarge-generation gen-num size => result`

Arguments `gen-num` A generation number.

`size` The amount (in bytes) by which the generation is to be enlarged.

Values `result` A boolean.

Description The function `enlarge-generation` enlarges generation `gen-num` by `size` bytes. If possible, an existing segment in generation `gen-num` is enlarged, otherwise a new segment of size `size` is added to the generation.

`result` is `t` on success and `nil` on failure.

This function is useful when it is known that a generation will need to grow. After `enlarge-generation` is called, the Garbage Collector is saved the work of deducing that the generation must grow.

`enlarge-generation` is most useful in non-interactive applications, where relatively long GC delays are not a problem. In this case, enlarging generations 0 and 1 by several Mb may improve the overall performance of the GC.

Note: `enlarge-generation` is implemented only in 32-bit LispWorks. It is not relevant to the Memory Management API in 64-bit implementations. In 64-bit implementations you can use `set-default-segment-size`.

See also `set-default-segment-size`

enlarge-static *Function*

Summary Enlarges the size of the first static segment in 32-bit LispWorks.

Package `hcl`

Signature `enlarge-static size => result`

Arguments `size` A non-negative `fixnum`.

Values `result` A boolean.

Description This function can be used when the system would otherwise allocate additional static segments. Such additional segments would cause the application to grow irreversibly.

`size` is the amount (in bytes) by which the static segment is to be enlarged. It is rounded up to a multiple of 64K.

`result` is `t` if the static segment was successfully enlarged, and `nil` otherwise.

Use `room`, with argument `t`, to find the size of the static segments, and thus the size by which to enlarge the first static segment.

Note: `enlarge-static` is implemented only in 32-bit LispWorks. It is not relevant to the Memory Management API in 64-bit implementations, where the irreversible growth problem described above does not exist.

See also `in-static-area`
`room`
`set-default-segment-size`
`switch-static-allocation`

expand-generation-1 *Function*

Summary Controls expansion of generation 1 in 32-bit LispWorks.

Package `hcl`

Signature `expand-generation-1 on`

Arguments `on` `t`, `nil` or `1`.

Description The function `expand-generation-1` controls the subsequent behavior of the garbage collector when insufficient space is freed by a `mark-and-sweep`. When this occurs, either generation 1 is expanded, or the objects in it are promoted.

If `on` is `nil`, generation 1 is never expanded.

If `on` is `t`, generation 1 is always expanded (rather than promotion) when needed.

If `on` is `1`, generation 1 is only expanded if its current size is less than 500000 bytes. This is the initial setting.

Note: `expand-generation-1` is implemented only in 32-bit LispWorks. It is not relevant to the Memory Management API in 64-bit implementations, where you can use `set-default-segment-size`.

See also `clean-generation-0`
`collect-generation-2`
`collect-highest-generation`
`mark-and-sweep`
`set-default-segment-size`
`set-gc-parameters`

extend-current-stack *Function*

Summary	Extends the current stack.	
Package	<code>hcl</code>	
Signature	<code>extend-current-stack &optional how-much => size</code>	
Arguments	<code>how-much</code>	What percentage the stack should be extended by. The default is 50.
Values	<code>size</code>	The new size of the stack, after extending.
Description	Extend the current stack by the given percentage.	
Example	To double the size of the current stack: <code>(hcl:extend-current-stack 100)</code>	
Compatibility Note	In LispWorks 4.4 and previous on Windows and Linux platforms, <code>extend-current-stack</code> is not implemented. This is fixed in LispWorks 5.0 and later.	
See also	<code>current-stack-length</code> <code>*stack-overflow-behaviour*</code>	

extended-time *Macro*

Summary	Prints useful timing information, including information on garbage collection (GC) activity.			
Package	<code>hcl</code>			
Signature	<code>extended-time &body body</code>			
Arguments	<code>body</code>	The forms to be timed.		
Description	<p>The macro <code>extended-time</code> runs the forms in <code>body</code>. It then prints a summary of the time taken followed by a breakdown of time spent in the GC.</p> <p>The three columns of the GC breakdown show, respectively, total time, user time, and system time. The rows of the GC breakdown indicate the type of activity.</p>			
In 32-bit LispWorks these rows begin:				
<code>main promote</code> indicates promotions from generation 0.				
<code>internal promote</code>				
indicates when an attempt to promote from one generation to the next causes promotion of the higher generation, to make room for the objects from the lower generation.				
<code>fixup</code> is a part of the compaction and promotion process.				
In 64-bit LispWorks these rows begin:				
<code>Standard gen-num (n calls)</code>				
indicates <code>n</code> Standard GCs (includes automatic GCs and calls to <code>gc-generation</code>) in which the highest generation collected was <code>gen-num</code> .				
<code>Marking gen-num (n calls)</code>				

indicates *n* Marking GCs (includes calls to `marking-gc`) in which the highest generation collected was *gen-num*.

Thus in the example below

```
Standard 1 (6 calls) ...
```

indicates that there were 6 Standard GCs in which the highest generation collected was 1.

Example

This example illustrates output in 32-bit LispWorks:

```
CL-USER 2 > (extended-time (foo))
Timing the evaluation of (FOO)

User time      =      7.203
System time    =      0.046
Elapsed time   =      7.265
Allocation     = 84011236 bytes
0 Page faults
Calls to %EVAL      23000075

                                total / user / system
total gc activity           =2.125000/ 2.078125/ 0.046875
main promote (9 calls)      =1.640625/ 1.593750/ 0.046875
mark and sweep (12 calls)   =0.484375/ 0.484375/ 0.000000
internal promote (3 calls)=0.437500/ 0.421875/ 0.015625
promote (0 calls)          =0.000000/ 0.000000/ 0.000000
fixup (21 calls)           =0.562500/ 0.562500/ 0.000000
compact (0 calls)          =0.000000/ 0.000000/ 0.000000
537870911
```

This example illustrates output in 64-bit LispWorks:

```

CL-USER 2 > (extended-time (foo))
Timing the evaluation of (FOO)

User time      =        4.468
System time    =        0.208
Elapsed time   =        4.716
Allocation     = 96030696 bytes
0 Page faults

                                total      /  user      /  system
total gc activity      = 1.148826 /  0.959855 /  0.188971
Standard 1 (6 calls) = 0.761885 /  0.632905 /  0.128980
Standard 2 (1 calls) = 0.386941 /  0.326950 /  0.059991
1152921504607846975

```

See also [time](#)

file-string *Function*

Summary Returns the contents of a file as a string.

Package [hcl](#)

Signature `file-string file &key length external-format => string`

Arguments `file` A pathname, string or file-stream, designating a file.

`length` The number of characters to return in `string`, or `nil` (the default).

`external-format` An external format specification, default value `:default`.

Values `string` A string containing characters from `file`.

Description Returns the entire contents of `file` (if `length` is `nil`), or the first `length` characters, as a string.

Example `CL-USER 26 > file-string "configure.lisp" :length 18
";; -*- Mode: Lisp;"`

See also `guess-external-format`

file-writable-p *Function*

Summary Tests whether a file is writable.

Package `hcl`

Signature `file-writable-p file => result`

Arguments `file` A pathname, string or file-stream, designating a file.

Values `result` `t` or `nil`

Description Checks if *file* is writable. Note that this checks the properties of the file, so trying to write to the file may still fail if the file is non-writable for other reasons, for example if it is opened for writing by another program.

Example `CL-USER 44 > file-writable-p (sys:lispworks-file
"private-patches/load.lisp")
T`

find-object-size *Function*

Summary Returns the size in bytes of the representation of any Lisp object.

Package `hcl`

Signature `find-object-size object => size`

Arguments	<i>object</i>	Any Common Lisp form.
Values		<p>The result is an integer which is the number of bytes of heap storage currently used to represent the object. If the object takes up no heap storage (fixnum or character), then 0 is returned. Such objects are represented by an immediate value held in a single machine “word”.</p> <p>The size of a heap object includes hidden space required to hold type and other information; for instance, a string of 10 characters occupies more than 10 bytes of storage.</p>
Description		Certain Common Lisp objects are not represented by a single heap object; for instance, using <code>find-object-size</code> on a hash-table is misleading as the function returns the size of the hash-table descriptor, rather than the total of the descriptor and the hash-table-array. General vectors and arrays also have this property. All symbols are of the same size, since the print name is not part of a symbol object.
Example		<pre>USER 37 > (hcl:find-object-size (make-string 1000 :initial-element #\A)) 1012</pre>
See also		room total-allocation

finish-heavy-allocation *Function*

Summary	Tells the system that allocation of many long-lived objects is over.
Package	<code>hcl</code>
Signature	<code>finish-heavy-allocation</code>

Description	The function <code>finish-heavy-allocation</code> tells the system that the application finished doing 'heavy' allocation, and from that point onwards allocation is 'normal'. The main distinction between heavy and normal allocation is the typical lifetime of objects: normal allocation means most of new objects are ephemeral, while heavy allocation a large proportion of the new objects are long-lived.
	Heavy allocation normally happens when loading, either the application itself or large amount of data. Operations that do not involve loading will almost always be normal. Hence the time that is useful to call <code>finish-heavy-allocation</code> is after loading something.
See also	<code>with-heavy-allocation</code>

	<i>Function</i>		
Summary	Unflags an object for special action on garbage collection.		
Package	<code>hcl</code>		
Signature	<code>flag-not-special-free-action <i>object</i> => nil</code>		
Arguments	<table> <tr> <td><i>object</i></td> <td>The object on which the special actions are to be removed.</td> </tr> </table>	<i>object</i>	The object on which the special actions are to be removed.
<i>object</i>	The object on which the special actions are to be removed.		
Values	Returns <code>nil</code> .		
Example	<pre>CL-USER 29 : 1 > (make-instance 'capi:title-pane) #<CAPI:TITLE-PANE "" 20F9898C> CL-USER 30 : 1 > (flag-not-special-free-action *) NIL</pre>		
See also	<code>add-special-free-action</code> <code>flag-special-free-action</code> <code>remove-special-free-action</code>		

	<i>Function</i>
Summary	Flags an object for special action on garbage collection.
Package	<code>hcl</code>
Signature	<code>flag-special-free-action <i>object</i> => t</code>
Arguments	<i>object</i> The object on which the special actions are to be performed. This cannot be a symbol.
Values	Returns <code>t</code> .
Description	Note that all the current special-free-action functions are performed on the object. Use <code>flag-not-special-free-action</code> to unflag an object.
Example	<pre>CL-USER 29 > (make-instance 'capi:title-pane) #<CAPI:TITLE-PANE "" 20F9898C> CL-USER 30 > (flag-special-free-action *) T</pre>
See also	<code>add-special-free-action</code> <code>flag-not-special-free-action</code> <code>remove-special-free-action</code>

	<i>Function</i>
Summary	Does a Copying GC.
Package	<code>hcl</code>
Signature	<code>gc-generation <i>gen-num</i> &key coalesce promote <i>block</i> => <i>allocation</i></code>
Arguments	<i>gen-num</i> An integer between 0 and 7 inclusive, or <code>t</code> .

	<i>coalesce</i>	A generalized boolean.
	<i>promote</i>	A generalized boolean.
	<i>block</i>	An integer between 0 and 7, inclusive, or one of the keywords <code>:blocking-gen-num</code> and <code>:all</code> .
Values	<i>allocation</i>	The total allocation in generation <i>gen-num</i> and younger generations.
Description	<p>The function <code>gc-generation</code> does a Garbage Collection of a specific generation. The actual operation is different between 64-bit LispWorks and 32-bit LispWorks.</p> <p><i>gen-num</i> should be a valid generation number, or <code>t</code>. The value <code>t</code> is mapped to the blocking generation number in 64-bit LispWorks, and to <code>2</code> in 32-bit LispWorks. For backwards compatibility the keyword <code>:blocking-gen-num</code> is also accepted, with the same meaning as <code>t</code>.</p> <p>It is especially helpful to GC the blocking generation (or other higher generations) when large, long-lived data structures become garbage. This is because higher generations are rarely collected by default. For the higher generations, the GC takes longer but recovers more space.</p> <p>Another situation which may require <code>gc-generation</code> is when objects are marked for special free action (by <code>flag-special-free-action</code>). If such objects live long enough to be promoted to higher generation, they may not be GCed long after there are no pointers to them. If the free action is important, you may need to periodically GC higher generation (typically the blocking generation, by passing <i>gen-num t</i>).</p>	

Operation in 64-bit LispWorks

By default `gc-generation` operates on the live objects in generation *gen-num* and all lower generations at or above the generation specified by *block* by copying them inside their current generation, and it operates on the live objects in

generations lower than *block* by copying them to the next higher generation.

If *promote* is non-`nil`, the live objects in generation *gen-num* are also promoted to the next generation. That is the same operation that happens when the GC is invoked automatically. The default value of *promote* is `nil`.

If *coalesce* is non-`nil`, all non-static live objects in lower generations are promoted to generation *gen-num*. That is what `clean-down` does (with *gen-num* being the highest generation). It may be useful directly in some cases. The default value of *coalesce* is `nil`.

block specifies a generation number up to which to promote. An integer value specifies the generation number. If *block* is `:blocking-gen-num`, then `gc-generation` promotes up to the blocking generation. If *block* is `:all`, then `gc-generation` promotes nothing. The default value of *block* is `:blocking-gen-num`.

`gc-generation` is useful when you know points in your application where many objects tend to die, or when you know that that application is less heavily loaded at some time. Typically many objects die in the end (or beginning) of an iteration in a top level loop of the application, and that is normally a useful place to put a call to `gc-generation` of generation 2 or generation 3. If you know a time when the application can spend time GCing, a call to `gc-generation` with a higher value of *gen-num* may be useful. It is probably never really useful to use `gc-generation` on generation 0 or 1.

To decide on which *gen-num* to call `gc-generation`, check which generation gets full by making periodic calls to `room`.

`gc-generation` with *promote* or *coalesce* may also be useful to move objects from the blocking generation to higher generations, which does not happen automatically (except when saving the image). For example, after loading a large amount

of code, and before generating any data that may die shortly, assuming the blocking generation is 3, it may be useful to do:

```
(gc-generation 4 :coalesce t)
```

to move all (non-static) objects to generation 4, where they will not be touched by the GC any more (except following pointers to younger generations).

Operation in 32-bit LispWorks

`gc-generation` marks and sweeps the generation *gen-num* and all generations below, and then does some additional cleanups. *coalesce*, *promote* and *block* are ignored.

Compatibility	In 32-bit LispWorks, <code>gc-generation</code> simply calls <code>mark-and-sweep</code> . This has a similar effect, but two significant differences must be noted:
Note	<ol style="list-style-type: none"> 1. by default, <code>gc-generation</code> promotes the young generations, so repeated calls to <code>gc-generation</code> will promote everything to generation <i>gen-num</i> or generation <i>block</i> (whichever is lower). In contrast <code>mark-and-sweep</code> never promotes. 2. In 32-bit LispWorks, generation 2 is the blocking generation. In 64-bit LispWorks, the default blocking generation is generation 3. That is because the 64-bit implementation promotes faster and so needs more generations before the block.

Compatibility	<code>(gc-generation t)</code>
Note	is intended as the replacement of <code>(mark-and-sweep 2)</code>

See also	<code>clean-down</code> <code>mark-and-sweep</code> <code>marking-gc</code> <code>set-blocking-gen-num</code>
----------	--

	<i>Function</i>
Summary	Garbage collects if the previous call requires more space than is actually available in 32-bit LispWorks.
Package	<code>hcl</code>
Signature	<code>gc-if-needed => nil</code>
Arguments	None.
Values	Returns <code>nil</code> .
Description	This function checks to see if the amount of allocation from the previous call is more than <code>system:*allocation-interval*</code> , and if it is, performs a mark and sweep and promotion on generation 0. It also tries to reduce the big-chunk area. This is a fairly brief operation, and can be used whenever some operation is finished and may have left some garbage. The system itself uses it after compiling and loading files, when waiting for input, etc.
	Note: This function does nothing in 64-bit LispWorks.
See also	<code>avoid-gc</code> <code>get-gc-parameters</code> <code>mark-and-sweep</code> <code>normal-gc</code> <code>set-gc-parameters</code> <code>without-interrupts</code> <code>with-heavy-allocation</code>

	<i>Function</i>
Summary	Returns the current default generation.
Package	<code>hcl</code>

Signature	<code>get-default-generation => default-gen</code>
Arguments	None.
Values	Returns the current default.
Description	By default, all new objects are allocated to a specific generation. This function returns the current value of this default generation.
	Note: in 64-bit LispWorks, <code>get-default-generation</code> returns 0.
See also	allocation-in-gen-num clean-generation-0 collect-generation-2 collect-highest-generation expand-generation-1 set-default-generation *symbol-alloc-gen-num*

get-gc-parameters *Function*

Summary	Returns the current values of various garbage collector parameters in 32-bit LispWorks.	
Package	<code>hcl</code>	
Signature	<code>get-gc-parameters parameters => values</code>	
Arguments	<code>parameters</code>	A keyword representing a single GC parameter. Any other value means all parameters.
Values	<code>values</code>	If <code>parameters</code> specifies a single GC parameter, the value of that parameter is returned. Otherwise <code>values</code> is an alist containing every GC parameter, together with its current value.

Description	See <code>set-gc-parameters</code> for a full description of these parameters. With keyword argument, of one of the parameters, the corresponding value is returned.
	Note: <code>get-gc-parameters</code> is implemented only in 32-bit LispWorks. It is not relevant to the Memory Management API in 64-bit implementations.
Example	<pre>CL-USER 1 > (get-gc-parameters :minimum-overflow) 500000 CL-USER 2 > (pprint (get-gc-parameters t)) ((:ENLARGE-BY-SEGMENTS . 10) (:MINIMUM-FOR-PROMOTE . 1000) (:MAXIMUM-OVERFLOW . 1000000) (:MINIMUM-OVERFLOW . 500000) (:MINIMUM-BUFFER-SIZE . 200) (:NEW-GENERATION-SIZE . 262144) (:PROMOTE-MAX-BUFFER . 100000) (:PROMOTE-MIN-BUFFER . 200) (:MAXIMUM-BUFFER-SIZE . 131072) (:MINIMUM-FOR-SWEEP . 8000) (:BIG-OBJECT . 131072))</pre>
See also	<code>set-gc-parameters</code>

	<i>Function</i>
get-working-directory	
Summary	Finds the current working directory.
Package	<code>hcl</code>
Signature	<code>get-working-directory => cwd</code>
Arguments	None.

Values	<code>cwd</code>	The current working directory, as a pathname.
Description		This function is used to find the current working directory. It returns a pathname, the directory component of which is the current working directory.
Example		<pre>CL-USER 1 > (get-working-directory) #P"/u/dubya/"</pre>
See also		<code>cd</code> <code>change-directory</code>

handle-existing-defpackage *Variable*

Summary	Controls LispWorks' response when <code>defpackage</code> is used on an existing package that is different from the definition given.		
Package	<code>hcl</code>		
Initial value	<code>(:warn :modify)</code>		
Description	<p>The standard explicitly declines to define what <code>defpackage</code> does if the named package already exists and is in a different state to that described by the <code>defpackage</code> form. The variable <code>*handle-existing-defpackage*</code> is an extension to Common Lisp which allows you to select between alternative behaviors that are known to be useful.</p> <p>The two alternatives are to modify the package to conform exactly to the definition, removing features if necessary, or to merely add features specified in the <code>defpackage</code> but missing from the package. You can also control whether a condition is signalled.</p> <p>The variable consists of a list of any of the following:</p> <table> <tr> <td><code>:error</code></td> <td>Signal an error.</td> </tr> </table>	<code>:error</code>	Signal an error.
<code>:error</code>	Signal an error.		

<code>:warn</code>	Signal a warning.
<code>:add</code>	Add the new symbols to the externals, imports, and so on.
<code>:modify</code>	Modify the package to have only these externals.
<code>:verbose</code>	The signalled errors or warnings also contain details of the differences.

The options `:error` and `:warn` cannot be specified at the same time. One of `:add` and `:modify` must be specified. Undistinguished internals (that is, internal symbols that are not imported or shadowed), `:intern` options and sizes are ignored when deciding whether to signal.

Note that when you use `:modify` some symbols can be uninterned if `defpackage` imports another symbol with the same name from another package through `:import-from`, `:shadowing-import-from` or `:export`. This happens whether the symbol has a definition as a function, a variable, or nay other Lisp construct, so after making such a change in the package, you should re-execute the definitions that were (presumably erroneously) attached to the uninterned symbols.

Notes	<code>*handle-existing-defpackage*</code> is an extension to Common Lisp.
See also	<code>defpackage</code>

handle-old-in-package *Variable*

Summary	Controls the handling of CLtL1-style <code>in-package</code> forms.
Package	<code>hcl</code>
Initial Value	<code>:warn</code>

Description The variable `*handle-old-in-package*` controls what happens when a CLtL1-style `in-package` form is processed. This refers to the specification in Common Lisp the Language, first Edition, which preceded ANSI Common Lisp and specified `in-package` as a function with keyword arguments.

The allowed values are as follows:

<code>:quiet</code>	Quietly use the CLtL1 definition of the <code>in-package</code> function.
<code>:warn</code>	Signal a warning and use the old definition.
<code>:error</code>	Signal a continuable error.

See also `*handle-old-in-package-used-as-make-package*`

`*handle-old-in-package-used-as-make-package*` *Variable*

Summary Controls the handling of CLtL1-style `in-package` forms.

Package `hcl`

Initial Value `:quiet`

Description The variable `*handle-old-in-package-used-as-make-package*` controls what happens when a CLtL1-style `in-package` form which attempts to create a package is processed. This refers to the specification in Common Lisp the Language, first Edition, which preceded ANSI Common Lisp and specified `in-package` as a function with keyword arguments.

The allowed values are as follows:

<code>:quiet</code>	Handle according to the value of <code>*handle-old-in-package*</code> .
<code>:warn</code>	Signal a warning and create the package.
<code>:error</code>	Signal a continuable error.

See also ***handle-old-in-package***

load-fasl-or-lisp-file *Variable*

Summary Controls the behavior of `load` for untyped pathnames.

Package `hcl`

Description The variable `*load-fasl-or-lisp-file*` determines whether `(load "foo")` should load the binary file `(foo.ofasl, foo.ufasl, foo.xfasl` etc, depending on platform) or `foo.lisp`, when both exist. It may take the following values:

`:load-newer` If the fasl is out-of-date, the lisp file is loaded, and a warning message is output in verbose mode.

`:load-newer-no-warn` Like `:load-newer`, but without the warning.

`:load-fasl` Always choose fasl files in preference to lisp files, but when verbose, warn if the lisp file is newer.

`:load-fasl-no-warn` Like `:load-fasl`, but without the warning.

`:load-lisp` Always choose lisp files in preference to fasl.

`:recompile` If the fasl file is out-of-date or there is none, compile and load the new fasl.

`:maybe-recompile` If the fasl is out-of-date, queries whether to load it, recompile and then load it, or load the lisp file.

Initial Value `:load-fasl`

	mark-and-sweep	<i>Function</i>
Summary	Garbage collects a specified generation in 32-bit LispWorks.	
Package	<code>hcl</code>	
Signature	<code>mark-and-sweep gen-number => bytes</code>	
Arguments	<code>gen-number</code>	0 for the most recent generation, 1 for the most recent two generations, and so on up to a maximum (usually 3). Numbers outside this range signal an error.
Values	<code>bytes</code>	The number of bytes allocated in that generation.
Description	<p><code>mark-and-sweep</code> is used to garbage-collect a specified generation of storage (and all lower generations). A call to this function forces the garbage collector to scan the specified generations. This can be of use in obtaining consistent timings of programs that require memory allocation. Alternatively, performance can sometimes be improved by forcing a garbage collection, when it is known that little memory has been allocated since a previous collection, rather than waiting for a later, more extensive collection. For example, the function could be called outside a loop that allocates a small amount of memory.</p> <p>It is specially helpful to mark and sweep generation 2 when large, long-lived data structures become garbage, because by default it is never marked and swept. The higher the generation number the more time the <code>mark-and-sweep</code> takes, but also the more space recovered.</p> <p>Note: <code>mark-and-sweep</code> is implemented only in 32-bit LispWorks. It is not relevant to the Memory Management API in 64-bit implementations. In 64-bit implementations you can use <code>gc-generation</code> or <code>marking-gc</code>.</p>	

Examples `(mark-and-sweep 0) ; collect most recent generation`
`(mark-and-sweep 3) ; collect all generations`

See also `avoid-gc`
`block-promotion`
`get-gc-parameters`
`gc-if-needed`
`normal-gc`
`set-array-weak`
`set-gc-parameters`
`set-hash-table-weak`
`without-interrupts`
`with-heavy-allocation`

max-trace-indent *Variable*

Summary The maximum level of indentation used in trace output.

Package `hcl`

Initial value `50`

Description `*max-trace-indent*` is the maximum indentation that is used during output from tracing. Typically each successive invocation of tracing causes the output to be further indented, making it easier to see how the calls are nested. The value of `*max-trace-indent*` should be an integer.

Example `USER 8 > (setq hcl:*max-trace-indent* 4)`
`4`
`USER 9 > (defun sum (n res) (if (= n 0)`
`res`
`(+ n (sum (1- n) res))))`
`SUM`

`USER 10 > (trace sum)`
`SUM`

```

USER 11 > (sum 3 0)
0 SUM > (3 0)
  1 SUM > (2 0)
    2 SUM > (1 0)
      3 SUM > (0 0)
      3 SUM < (0)
      2 SUM < (1)
        1 SUM < (3)
0 SUM < (6)
6

```

Notes ***max-trace-indent*** is an extension to Common Lisp.

See also **trace**

normal-gc *Function*

Summary Returns the image to normal garbage collection activity in 32-bit LispWorks.

Package **hcl**

Signature **normal-gc => t**

Arguments None.

Values The function returns the single result **t**.

Description **normal-gc** resets various internal parameters that determine the frequency and extent of garbage collection to their default settings.

normal-gc is generally used in conjunction with **avoid-gc**, to cancel the effects of the latter.

Note: **normal-gc** is useful only in 32-bit LispWorks. In 64-bit implementations it does nothing and simply returns nil.

See also [avoid-gc](#)
[get-gc-parameters](#)
[gc-if-needed](#)
[mark-and-sweep](#)
[set-gc-parameters](#)
[without-interrupts](#)
[with-heavy-allocation](#)

packages-for-warn-on-redefinition *Variable*

Summary List of packages whose symbols should be checked for definitions.

Package `hcl`

Initial Value A list containing "COMMON-LISP" and other package names

Description LispWorks detects attempts to define external symbols in certain packages. The packages are those on the list `*packages-for-warn-on-redefinition*`.
LispWorks is configured to protect the COMMON-LISP package and other system packages.
The action taken by LispWorks on such attempted definitions depends on the value of `*handle-warn-on-redefinition*`.

See also [*handle-warn-on-redefinition*](#)

parse-float *Function*

Summary Parses a float from a string and returns it as float.

Package `hcl`

Signature	<code>parse-float string &key start end default-format => float</code>	
Arguments	<code>string</code>	A string
	<code>start, end</code>	Bounding index designators for <code>string</code>
	<code>default-format</code>	One of the atomic type specifiers <code>short-float</code> , <code>single-float</code> , <code>double-float</code> , or <code>long-float</code> .
Values	<code>float</code>	A float
Description	<p>The function <code>parse-float</code> parses a float from the substring of <code>string</code> delimited by <code>start</code> and <code>end</code> and returns it as <code>float</code>.</p> <p>If the substring represents an integer or the exponent marker is E or is omitted, then <code>float</code> will be of type <code>default-format</code>, which defaults to the value of <code>*read-default-float-format*</code>. Otherwise, its type will match the exponent marker as specified by 2.3.2.2 "Syntax of a Float" in the Common Lisp standard.</p> <p>If the substring does not represent an integer or a float, then an error of type <code>parse-error</code> is signalled.</p>	
Examples	<pre>(parse-float "10") => 10.0f0 (parse-float "10" :default-format 'double-float) => 10.0d0 (parse-float "10d0") => 10.0d0 (parse-float "10.5") => 10.5f0 (parse-float "10.5d0") => 10.5d0</pre>	

print-profile-list *Function*

Summary Prints a report of symbols that have been profiled.

Package `hcl`

Signature	<code>print-profile-list &key sort limit cutoff collapse => nil</code>
Arguments	<p><code>sort</code> <code>:call</code>, <code>:profile</code> or <code>:top</code></p> <p><code>limit</code> An integer.</p> <p><code>collapse</code> A generalized boolean.</p> <p><code>cutoff</code> A real number.</p>
Values	<code>print-profile-list</code> returns <code>nil</code> .
Description	<p>The function <code>print-profile-list</code> prints a report of symbols, after profiling using <code>profile</code>, or <code>start-profiling</code> followed by <code>stop-profiling</code>.</p> <p>If the profiler was set up with <code>style :tree</code>, then a tree of calls is printed first, according to <code>limit</code>, <code>cutoff</code> and <code>collapse</code>. Then a columnar report is printed showing how often each function was called, profiled and found on the top of the stack. This report is sorted by the column indicated by the value of <code>sort</code>.</p> <p>If the profiler was set up with <code>style :list</code>, then only the columnar report is printed.</p> <p><code>sort</code> can take these values:</p> <ul style="list-style-type: none"> <code>:call</code> Sort by the number of times the function was called. <code>:profile</code> Sort by the number of times the function was found on the stack. <code>:top</code> Sort by the number of times the function was found at the top of the stack. <p>If <code>sort</code> is not passed then the results are printed as after the profiling run. The default is the value of the variable <code>*default-profiler-sort*</code>.</p> <p><code>limit</code> is the maximum number of lines printed in the columnar report as described for <code>*default-profiler-limit*</code>. The</p>

default is the value of the variable ***default-profiler-limit***.

cutoff is the minimum percentage that the profiler will display in the output tree as described for ***default-profiler-cutoff***. The default is the value of the variable ***default-profiler-cutoff***.

collapse controls collapsing of the output tree as described for ***default-profiler-collapse***. The default is the value of the variable ***default-profiler-collapse***.

Example First set up the profiler :

```
CL-USER 1 > (set-up-profiler
               :symbols
               '(cadr car eql fixnump + 1+ caaddr caddr))

CL-USER 2 > (profile (dotimes (a 1000000 nil)
                         (+ a a)
                         (car '(foo))))
```

Then call **print-profile-list**:

```
CL-USER 3 > (print-profile-list :sort :call)
```

```
profile-stacks called 327 times
```

```
Cumulative profile summary
```

Symbol	called	profile
(%)	top (%)	
CADR	5000012	13
(4)	13 (4)	
CDDR	3000000	3
(1)	3 (1)	
EQL	2000202	4
(1)	4 (1)	
FIXNUMP	2000003	2
(1)	2 (1)	
CAR	1000000	1
(0)	1 (0)	
+	1000000	3
(1)	3 (1)	
CAADR	1000000	2
(1)	2 (1)	
1+	1000000	2
(1)	2 (1)	

```
Top of stack not monitored 91% of the time
```

```
NIL
```

Notes You can suppress printing of those symbols that are currently profiled but which were not called in the profiling run by setting `system:*profiler-print-out-all*` to `nil`.

`system:*profiler-print-out-all*` is a variable defined when the profiler is loaded by `set-up-profiler`. Its initial value is `nil`.

See also `*default-profiler-collapse*`
`*default-profiler-cutoff*`
`*default-profiler-limit*`
`*default-profiler-sort*`

profile	<i>Macro</i>
Summary	Runs the specified forms, and prints a performance profile.
Package	<code>hcl</code>
Signature	<code>profile &body forms => final</code>
Arguments	<p><i>forms</i> The forms making up the program being profiled.</p>
Values	<i>final</i> The result of evaluating the final form.
Description	<p>This macro starts up the LispWorks program profiler. This tool is useful for determining the time critical elements of a program.</p> <p>At a regular time interval the Lisp process is halted and the execution stack is scanned for the presence of any symbols in the list <code>*profile-symbol-list*</code>. Counters are maintained for the number of calls to each symbol, the total number of times the symbol is found on the stack, and the number of times the profiler finds the symbol on the top of the stack.</p> <p>This information is then presented as absolute numbers and as a percentage of the total number of calls to the profiler. These figures taken together give useful information about which functions the program spends most of its time executing.</p>

Examples

```

USER 22 > (set-up-profiler
             :symbols '(* gethash typep maphash))
NIL
USER 23 > (profile (let ((x 1))
                      (loop for a from 1 to 50 by 1
                            do (setq x (* a x)))
                            finally (return x))))
profile-stacks called 12 times
Symbol           called   profile (%)  top
(%)
MAPHASH          1        0        (0)
0 (0)*           50       1        (8)
0 (0)
SYSTEM::DUMMY-STRUCTURE-ACCESSOR 6        0        (0)
0 (0)
SYSTEM::DUMMY-STRUCTURE-SETTER   9        0        (0)
0 (0)
TYPEP            19       1        (8)
0 (0)
GETHASH          78       3        (25)
3 (25)
Top of stack not monitored 75% of the time
3041409320171337804361260816606476884437764156896051200
0000000000

```

See also

- `print-profile-list`
- `*profile-symbol-list*`
- `set-up-profiler`

profiler-threshold *Variable*

Summary	Controls which symbols are profiled on repeated profiling runs.
Package	<code>hcl</code>
Description	<code>*profiler-threshold*</code> is used with repeated profiling runs, to control which symbols are profiled. It is set by <code>set-profiler-threshold</code> .

See also [set-profiler-threshold](#)

profile-symbol-list *Variable*

Summary The list of symbols to be profiled.

Package [hcl](#)

Description ***profile-symbol-list*** is the list of symbols that are profiled if `profile` is called. Symbols in this list are monitored by the profiler to see if their function objects are on the stack when the profiler interrupts the Lisp process. The length of this list does not affect the speed of the profiling run.

Initial Value `nil`

Notes ***profile-symbol-list*** should normally be set by one of the above functions which check that the symbol is suitable for profiling before adding them to the list.

See also [add-symbol-profiler](#)
[remove-symbol-profiler](#)
[set-up-profiler](#)

profiler-tree-from-function *Function*

Summary Prints a call tree of profiled code below a given function.

Package [hcl](#)

Signature `profiler-tree-from-function function-name &optional max-depth`

Arguments `function-name` A symbol naming a function.
`max-depth` A number or `nil`.

Description	<p>The function <code>profiler-tree-from-function</code> prints a tree with root <i>function-name</i> whose children are the callees of <i>function-name</i> and their callees.</p> <p><code>profiler-tree-from-function</code> uses the data from the previous 'profile session' with style <code>:tree</code>. A profile session ends at the end of <code>profile</code> or when <code>stop-profiling</code> is called, or when the Profiler tool finishes profiling.</p> <p>In both cases the counts of profile calls is the total counts of the calls to <i>function-name</i>. Note that the percentages (the number in parentheses) are percentages from the total number of profile calls, rather than from the numbers of calls to <i>function-name</i>.</p> <p>If <i>max-depth</i> is a number it limits the depth of tree that is printed to that value. The default value of <i>max-depth</i> is <code>nil</code>, meaning no limit on the depth that is printed.</p>
See also	<code>profile</code> <code>start-profiling</code> <code>stop-profiling</code>

	Function				
Summary	Prints a reversed call tree of profiled code below a given function.				
Package	<code>hcl</code>				
Signature	<code>profiler-tree-to-function <i>function-name</i> &optional <i>max-depth</i></code>				
Arguments	<table> <tr> <td><i>function-name</i></td><td>A symbol naming a function.</td></tr> <tr> <td><i>max-depth</i></td><td>A number or <code>nil</code>.</td></tr> </table>	<i>function-name</i>	A symbol naming a function.	<i>max-depth</i>	A number or <code>nil</code> .
<i>function-name</i>	A symbol naming a function.				
<i>max-depth</i>	A number or <code>nil</code> .				
Description	The function <code>profiler-tree-to-function</code> prints a tree with root <i>function-name</i> whose children are the callers of <i>function-</i>				

name and their callers. Note that the tree is reversed, that is, callers appear under their callees.

`profiler-tree-to-function` uses the data from the previous 'profile session' with style `:tree`. A profile session ends at the end of `profile` or when `stop-profiling` is called, or when the Profiler tool finishes profiling.

In both cases the counts of profile calls is the total counts of the calls to *function-name*. Note that the percentages (the number in parentheses) are percentages from the total number of profile calls, rather than from the numbers of calls to *function-name*.

max-depth limits the depth of tree that is printed. If *max-depth* is `nil` there is no limit on the depth that is printed. The default value of *max-depth* is 7.

See also

`profile`
`profiler-tree-from-function`
`stop-profiling`

		<i>Function</i>
Summary		Lists special variables referenced by a definition.
Package		<code>hcl</code>
Signature		<code>references-who function => result</code>
Arguments	<i>function</i>	A symbol or a function dspec.
Values	<i>result</i>	A list.
Description		The function <code>references-who</code> returns a list of the special variables referenced by the definition named by <i>function</i> .

Note: The cross-referencing information used by `references-who` is generated when code is compiled with source-level debugging switched on.

See also `toggle-source-debugging`
 `who-references`

remove-special-free-action *Function*

Summary Removes the specified function from the special actions performed when flagged objects are garbage collected.

Package `hcl`

Signature `remove-special-free-action function => function-list`

Arguments `function` The function to be removed.

Values `function-list` A list of the functions currently called to perform special actions, not including the one just removed.

Description Removes the specified function from the special actions performed when flagged objects are garbage-collected. (The special actions are added by `add-special-free-action`.)

See also `add-special-free-action`
 `flag-special-free-action`
 `flag-not-special-free-action`

remove-symbol-profiler *Function*

Summary Removes a symbol from the list of profiled symbols.

Package `hcl`

Signature	<code>remove-symbol-profiler symbol => nil</code>	
Arguments	<code>symbol</code>	A symbol to be removed from the <code>*profile-symbol-list*</code> .
Values	Returns <code>nil</code> .	
Description	<code>remove-symbol-profiler</code> removes a symbol from <code>*profile-symbol-list*</code> , the list of profiled symbols.	
See also	<code>add-symbol-profiler</code> <code>*profile-symbol-list*</code>	

reset-profiler *Function*

Summary	Resets the profiler so that symbols below a given threshold are no longer profiled.	
Package	<code>hcl</code>	
Signature	<code>reset-profiler &key according-to => nil</code>	
Arguments	<code>according-to</code>	One of two values — <code>:profile</code> or <code>:top</code> . This refers to which column of the profiling results <code>reset-profiler</code> uses to determine which symbols to delete from <code>*profile-symbol-list*</code> . The default is <code>:profile</code> .
Values	<code>reset-profiler</code> returns <code>nil</code> .	
Description	This function updates the list of symbols being profiled according to the results of the previous profiling run. <code>reset-profiler</code> runs down the list of symbols being profiled and removes any symbols whose appearance in the previous profiling run falls below the value <code>*profiler-threshold*</code> . In	

this way the number of symbols being considered by the profiler can be reduced to just those which are important.

Example `(reset-profiler :according-to :top)`

Notes Reducing the number of symbols in `profile-symbol-list` does not actually speed up the execution of the form being profiled, but does reduce the setting up time of the profiler and the size of the list of results.

See also `profile`
 `*profiler-threshold*`
 `print-profile-list`
 `set-profiler-threshold`

save-argument-real-p *Function*

Summary Used to determine if a build script knows the real name of the image being saved.

Package `hcl`

Signature `save-argument-real-p => realp`

Arguments None

Values `realp` A boolean.

Description The function `save-argument-real-p` can be used in a build script to determine if the argument passed to a subsequent call to `save-image` or `deliver` is the real filename of the application.

The return value `realp` is `t` in most cases. It is `nil` only when building an intermediate image for the purpose of building a universal binary, either by `save-universal-from-script` or

the Application Builder (see the *Common LispWorks User Guide*).

Operations in a build script that are related to the path of the saved image, such as building an application bundle, should be executed only when this function returns `t`. When using `save-universal-from-script`, any required application bundle should be created before calling that function (see the `save-macos-application.lisp` example below). When using the Application Builder, any required application bundle should be created in the build script only when `save-argument-real-p` returns `t`.

On architectures that do not have universal binaries, this function always returns `t`.

Example `examples/configuration/save-macos-application.lisp`

See also `save-universal-from-script`
`building-universal-intermediate-p`
`deliver`
`save-image`

save-image	<i>Function</i>
Summary	Saves the image to a new file.
Package	<code>hcl</code>
Signature	<code>save-image filename &key dll-exports dll-added-files automatic-init</code> <code>gc type normal-gc restart-function multiprocessing console</code> <code>environment remarks clean-down image-type => nil</code>

The `console` argument is available only in LispWorks for Windows and LispWorks for Macintosh.

Arguments	<i>filename</i>	A string. It is the name of the file that the image is saved as. This name should not be the same as the original name of the image.
	<i>dll-exports</i>	A list of strings, or the keyword : default .
	<i>dll-added-files</i>	A list of strings.
	<i>automatic-init</i>	A generalized boolean.
	<i>gc</i>	If non- nil , there is a garbage collection before the image is saved. The default value is t .
	<i>type</i>	Determines if some global variables are cleared before the image is saved. You can generally use the default value, which is : user .
	<i>normal-gc</i>	If this is t the function normal-gc is called before the image is saved. The default is t .
	<i>restart-function</i>	A function to be called on restart.
	<i>multiprocessing</i>	Controls whether multiprocessing is enabled on restart.
	<i>console</i>	On Windows <i>console</i> controls whether the new image will be a Console or GUI application and when, if ever, to make a console window in the latter case. On the Macintosh <i>console</i> controls when, if ever, to make a console window. Possible values are discussed below.
	<i>environment</i>	<i>environment</i> controls whether the LispWorks environment is started on restart. Possible values are discussed below.
	<i>remarks</i>	<i>remarks</i> adds a comment to the save history. The value should be a string.
	<i>clean-down</i>	When t , calls (clean-down t).

	<i>image-type</i> One of :exe, :dll or :bundle.
Values	Returns nil.
Description	<p>The function <code>save-image</code> saves the LispWorks image to a new executable or dynamic library containing any modifications you have made to the supplied image.</p> <p>For information about the sort of changes you might want to save in a new image, see "Customization of LispWorks" in the <i>LispWorks User Guide</i>.</p> <p>Do not use <code>save-image</code> when the graphical IDE is running. Instead create a build script similar to the examples below, or run LispWorks in a subprocess using the Application Builder tool.</p> <p>You cannot use <code>save-image</code> on Windows, Linux and Mac OS X when multiprocessing is running. It signals an error in this case.</p> <p>On Cocoa you can combine a call to <code>save-image</code> with the creation of an application bundle containing your new LispWorks image, as in the example shown below.</p> <p><i>dll-exports</i> is implemented only on Windows/Linux/Macintosh/FreeBSD. It controls whether the image saved is an executable or a dynamic library (DLL). The default value is :default and this value means an executable is saved. Otherwise <i>dll-exports</i> should be list (potentially nil) of strings. In this case a dynamic library is saved, and each string in <i>dll-exports</i> names a function which becomes an export of the dynamic library and should be defined as a Lisp function using <code>fli:define-foreign-callable</code>. Each exported name can be found by <code>GetProcAddress</code> (on Windows) or <code>dlsym</code> (on other platforms). The exported symbol is actually a stub which ensures that the LispWorks dynamic library has finished initializing, and then enters the Lisp code.</p>

On Mac OS X the default behavior is to generate an object of type "Mach-O dynamically linked shared library" with file type `dylib`. See *image-type* below for information about creating another type of library on Mac OS X.

On Linux/Macintosh/FreeBSD, to save a dynamic library image the computer needs to have a C compiler installed. This is typically `gcc` (which is available by installing Xcode on the Macintosh).

An image saved as a dynamic library (DLL):

- always runs multiprocessing, and
- may need to be shut down by `quitLispWorks` or by a callback which uses `dll-quit`.

automatic-init specifies whether a LispWorks dynamic library should initialize inside the call to `LoadLibrary` (on Microsoft Windows) or `dlopen` (on other platforms), or wait for further calls. Automatic initialization is useful when the dynamic library does not communicate by function calls. On Microsoft Windows it also allows `LoadLibrary` to succeed or fail according to whether the LispWorks dynamic library initializes successfully or not. Not using automatic initialization allows you to relocate the library if necessary using `InitLispWorks`, and do any other initialization that may be required. The default value of *automatic-init* is `t` on Windows, `nil` on other platforms. For more information about automatic initialization in LispWorks dynamic libraries, see "LispWorks as a dynamic library" in the *LispWorks User Guide*.

dll-added-files should be a list of filenames. It is ignored on Microsoft Windows. On other platforms if *dll-added-files* is non-`nil` then a dynamic library containing each named file is saved. Each file must be of a format that the default C compiler (`(scm:*c-default-compiler*)`) knows about and can incorporate into a shared library. Typically they will be C source files, but can also be assembler or object files. They must not contain exports that clash with names in the Lisp-

Works shared library (see “Dynamic library C functions” for the predefined exports). The added files are useful to write wrappers around calls into the LispWorks dynamic library. Such wrappers are useful for:

- Calling `InitLispWorks` when required, for example to relocate the LispWorks dynamic library to avoid memory clashes with other software, as described under "Startup Relocation" in the *LispWorks User Guide*.
- Calling `QuitLispWorks` when required.
- Changing calls that involve complex C structs or even C++ objects into plain calls, because accessing C structures in Lisp requires defining the structure, while in C it only needs to include the header.
- Creating 'stub' functions that can be called from Lisp, for example for calling a C++ method. The address of the stub function can be passed to Lisp which can call it using a function defined by `fli:define-foreign-funcallable`.
- Adding code that runs automatically inside the call to `dlopen`, by using `__attribute__ ((constructor))`

image-type defaults to `:exe` or `:dll` according to the value of `dll-exports` and therefore you do not normally need to supply *image-type*.

image-type :bundle is used only when saving a dynamic library. On Mac OS X it generates an object of type "Mach-O bundle" and is used for creating shared libraries that will be used by applications that cannot load dylibs (FileMaker for example). It also does not force the filename extension to be `dylib`. On other Unix-like systems *image-type* merely has the effect of not forcing the file type of the saved image, and the format of the saved image is the same as the default. On Microsoft Windows *image-type :bundle* is ignored.

Note: *image-type :bundle* is completely unrelated to the Mac OS X notion of an application bundle.

restart-function, if non-`nil`, specifies a function (with no arguments) to be called when the image is started. If *multiprocessing* is true, *restart-function* is called in a new process. *restart-function* is called after the initialization file is loaded. The default value of *restart-function* is `nil`.

Note: *restart-function* is not called if the command line argument `-no-restart-function` is present

When *multiprocessing* is `nil`, the executable image will start without multiprocessing enabled. When *multiprocessing* is true or the image is a DLL, the image will start with multiprocessing enabled. The default value of *multiprocessing* is `nil`.

console is implemented only in Lispworks for Windows and LispWorks for Macintosh. The possible values for *console* are as follows:

- | | |
|-----------------------|---|
| <code>:default</code> | Unchanged since previous save. |
| <code>t</code> | On the Macintosh, the value <code>t</code> has the same effect as the value <code>:always</code> .
On Windows, a Console application is saved, else a Windows application is saved which creates its own console according to the other possible values. |

`:input, :output, :io`

Whenever input, output or any I/O is attempted on `*terminal-io*`.

- | | |
|----------------------|--|
| <code>:init</code> | At startup, if input and output are not redirected |
| <code>:always</code> | At startup, even if input and output are redirected. |

The LispWorks for Windows and LispWorks for Macintosh images shipped have *console* set to `:input`.

The possible values for *environment* are as follows:

<code>:default</code>	Unchanged since previous save.
<code>nil</code>	Start with just the TTY listener.
<code>t</code>	Start the environment automatically, no TTY listener.
 <code>:with-tty-listener</code>	
	Start the environment automatically, but still have a TTY listener.

The LispWorks image shipped is saved with `:environment t` on all platforms except for the Motif images on Mac OS X, Solaris, HP-UX and DEC Tru64 UNIX.

You should not try to save a new image over an existing one. Always save images using a unique image name, and then, if necessary, replace the new image with the old one after the call to `save-image` has returned.

Example Here is an example build script. Save this to a file such as `c:/build-my-image.lisp`:

```
(load-all-patches)
(load "my-code")
(save-image "my-image")
```

Then run LispWorks with the command line argument
`-build c:/build-my-image.lisp` to save the image
`my-image.exe`.

This example shows a portable build script which, on Cocoa, saves your new LispWorks image in a Mac OS X application bundle. This allows your new LispWorks for Macintosh image to be launchable from the Finder or Dock and to have its own icon or other resources:

```

(load-all-patches)
(load "my-code")
#+:cocoa
(compile-file-if-needed
  (example-file
    "configuration/macos-application-bundle")
  :load t)
(save-image
  #+:cocoa
  (write-macos-application-bundle
    "/Applications/LispWorks 5.1/My LispWorks.app")
  #-:cocoa
  "my-lispworks")

```

Compatibility note LispWorks 5.0 and previous versions documented `-init` as the way to run LispWorks with a build script. This method is deprecated.

Note that LispWorks quits automatically after processing a build script via `-build`, whereas with `-init` you need to call `quit` explicitly at the end of the build script.

Compatibility note In LispWorks 5.0 and previous versions `dll-exports` is supported only on Windows.

`dll-added-files` and `automatic-init` are new in LispWorks 5.1.

See also

- `deliver`
- `dll-quit`
- `InitLispWorks`
- `LispWorksDlsym`
- `load-all-patches`
- `quit`
- `QuitLispWorks`

save-universal-from-script

Function

Summary Saves a universal binary LispWorks image using a script designed for saving a mono-architecture image.

Package	<code>hcl</code>	
Signature	<code>save-universal-from-script <i>target-image</i> <i>script-name</i> &key <i>output-stream</i> => <i>target-image</i></code>	
Arguments	<i>target-image</i>	A pathname designator.
	<i>script-name</i>	A pathname designator.
	<i>output-stream</i>	A stream or <code>nil</code> .
Values	<i>target-image</i>	A pathname designator.
Description	<p>The function <code>save-universal-from-script</code> provides a convenient way to create a universal binary on an Intel Macintosh, using a script designed for saving a mono-architecture image.</p> <p>The <i>script-name</i> is the name of a Common Lisp build script for saving or delivering an image, as would be used to create a mono-architecture image. It should load the application and then call either <code>deliver</code> or <code>save-image</code> as appropriate.</p> <p>The function <code>save-universal-from-script</code> runs the current LispWorks image in two subprocesses, once for the PowerPC architecture (under Rosetta) and once for the native Intel architecture, passing <code>-build <i>script-name</i></code> on the command line. The script is evaluated as normal, except that the filename that is passed to any call to <code>save-image</code> or <code>deliver</code> is ignored and a temporary filename is used instead. If these two subprocesses are successful, then the temporary images are combined to make a universal binary <i>target-name</i> in the same way as <code>create-universal-binary</code>.</p> <p>The command line arguments of the images run by the subprocesses will include the command line arguments that were passed to the current image. In addition, various undocumented command line arguments will be prepended, which control how <code>deliver</code> or <code>save-image</code> work in the script.</p>	

Any output generated by the subprocesses is written to *output-stream*. If this is `nil`, then the output is discarded. If this is `t` (the default), then the output is written to the standard output.

Note: The function `save-universal-from-script` can only be called from a LispWorks for Macintosh image that is itself a universal binary, such as the distributed image.

Example Suppose the file `my-build-script.lisp` contains

```
(load-all-patches)
(load "my-application-defsys")
(compile-system 'my-application-system :load t)
(deliver 'my-application-function "my-application" 5)
```

Then, the following call creates a universal binary `my-application` using this script:

```
(save-universal-from-script "my-application"
                            "my-build-script.lisp")
```

See also `save-image`
`create-universal-binary`
`building-universal-intermediate-p`
`save-argument-real-p`

set-array-weak

Function

Summary	Sets the weakness state of an array.				
Package	<code>hcl</code>				
Signature	<code>set-array-weak array weakp => weakp</code>				
Arguments	<table><tr><td><code>array</code></td><td>A non-displaced array, with <code>array-element-type t</code>.</td></tr><tr><td><code>weakp</code></td><td>If <code>weakp</code> is non-<code>nil</code>, the array is made weak. If <code>weakp</code> is <code>nil</code>, the array is made non-weak.</td></tr></table>	<code>array</code>	A non-displaced array, with <code>array-element-type t</code> .	<code>weakp</code>	If <code>weakp</code> is non- <code>nil</code> , the array is made weak. If <code>weakp</code> is <code>nil</code> , the array is made non-weak.
<code>array</code>	A non-displaced array, with <code>array-element-type t</code> .				
<code>weakp</code>	If <code>weakp</code> is non- <code>nil</code> , the array is made weak. If <code>weakp</code> is <code>nil</code> , the array is made non-weak.				

Values	Returns <code>weakp</code> .
Description	<p>By default, arrays are non-weak, and they keep alive all the objects that are stored in them. A weak array may remove a pointer if the object that it points to is not pointed to from somewhere else. When a pointer is removed like this, it is replaced in <code>array</code> with <code>nil</code>.</p> <p>Pointers are replaced by <code>nil</code> after a garbage collector operation that identifies that they can be replaced. This means that if the object that is pointed to has been promoted to a higher generation, a garbage collection of the higher generation is required to remove the pointer. Note that by default the system does not automatically GC the blocking generation or higher.</p>
	The weakness state of an array can be changed many times.
	In all implementations, <code>array</code> must not be a displaced array, and the <code>array-element-type</code> of <code>array</code> must be <code>t</code> .
	In 64-bit LispWorks, an additional requirement is that <code>array</code> must be an adjustable array.
	<code>set-array-weak</code> can be called at any moment.
	Note: An array can be made weak at creation time using the <code>:weak</code> argument to <code>make-array</code> .
See also	<code>copy-to-weak-simple-vector</code> <code>set-hash-table-weak</code> <code>make-array</code> <code>mark-and-sweep</code>

	Function
Summary	Set the current generation for storage allocation in 32-bit LispWorks.

Package	<code>hcl</code>
Signature	<code>set-default-generation num => num</code>
Arguments	<p><code>num</code> The number of the generation from which to do future allocation.</p>
Values	Returns <code>num</code> .
Description	<p>Set the current generation for storage allocation. By default the system allocates memory from the youngest generation (generation 0).</p> <p>Note: <code>set-default-generation</code> is useful only in 32-bit Lisp-Works. In 64-bit implementations it does nothing and returns 0.</p>
Examples	<pre>(set-default-generation 1) ;; allocate from an ;; older generation (set-default-generation 0) ;; return to normal</pre>
See also	allocation-in-gen-num clean-generation-0 collect-generation-2 collect-highest-generation expand-generation-1 get-default-generation set-promotion-count *symbol-alloc-gen-num*

set-hash-table-weak

Function

Summary	Sets the weakness state of a hash-table.
Package	<code>hcl</code>

Signature	<code>set-hash-table-weak hash-table weak => weakness-state</code>	
Arguments	<i>hash-table</i>	A hash-table.
	<i>weak</i>	Sets the weakness state of <i>hash-table</i> . Value may be:
		<code>:value</code> or <code>t</code> — An entry is kept if there is a pointer to the value from another object.
		<code>:key</code> — An entry is kept if there is a pointer to the key from another object.
		<code>:both</code> — An entry is kept if there are pointers to both the key and the value.
		<code>:one</code> or <code>:either</code> — An entry is kept if there is a pointer to either the key or the value.
		<code>nil</code> — Make the hash-table non-weak. All entries are kept.
Values	Returns <i>weak</i> , unless <code>t</code> was passed, when <code>:value</code> is returned.	
Description	<p>By default, hash-tables are not weak, which means that they keep alive all the keys and the values in the table.</p> <p>A weak hash-table allows entries to be removed if there are no other pointers to them. The <i>weakness-state</i> tells the system which entries may be removed like this.</p> <p>Entries that can be removed are removed after a garbage collector operation which identifies that they can be removed. This means that if the relevant object(s) (the key or the value) have been promoted to a higher generation, a garbage collection (GC) of the higher generation is required to remove them from the table. Note that by default the system does not automatically GC the blocking generation or higher.</p>	

The *weakness-state* of a hash-table can be changed repeatedly, at any time, at any point using any of the *weak* values listed above. It can also be set by `make-hash-table`.

See also

`make-hash-table`
`mark-and-sweep`
`set-array-weak`

set-gc-parameters

Function

Summary Sets the parameters from the garbage collector in 32-bit Lisp-Works.

Package `hcl`

Signature `set-gc-parameters &key maximum-buffer-size minimum-buffer-size big-object promote-min-buffer promote-max-buffer new-generation-size minimum-overflow maximum-overflow minimum-for-sweep minimum-for-promote enlarge-by-segments => <no values>`

Arguments *maximum-buffer-size*

 Maximum size of the small objects buffer.

minimum-buffer-size

 Minimum size of the small objects buffer.

big-object An object that is bigger than this value is “big”. That is, it is not allocated from the small objects buffer, but from the big-chunk area (if it is allocated in generation 0 in the normal way).

promote-min-buffer

 During promotion, a buffer is allocated in the generation being promoted into, and the objects promoted are moved into it. *promote-min-buffer* controls the minimum size of this buffer.

promote-max-buffer

Controls the maximum size of the promotion buffer.

new-generation-size

Controls the minimum enlargement of generation *gen-num*, for *gen-num* > 0. Value 0 means the generation is not expanded. Otherwise, *new-generation-size* must be a fixnum in the exclusive range (10000, 100000000) and the minimum expansion is then *new-generation-size* * *gen-num* words.
new-generation-size has no effect on the enlargement of generation 0.

maximum-overflow

Maximum size of the small-objects buffer in the big-chunk area.

minimum-overflow

Minimum size of the small-objects buffer in the big-chunk area.

minimum-for-promote

Controls the frequency of promotions. Setting *minimum-for-promote* to a high value causes the system to promote less frequently. This may improve performance for programs that allocate a lot of data for a short term and then delete it.

minimum-for-sweep

Controls when a mark-and-sweep takes place. Setting *minimum-for-sweep* to a high value causes the system to mark and sweep less often, which means it has to grow. The

CPU time spent in garbage collection is mostly smaller, but the process is bigger and may cause more disk access.

enlarge-by-segments

A minimum for how much the image grows each time a segment is enlarged, as a multiple of 64K. This parameter is ignored when adding a static segment.

Values	None.
Description	This function sets the parameters of the garbage collector, using the keywords described above. Note: <code>set-gc-parameters</code> is implemented only in 32-bit LispWorks. It is not relevant to the Memory Management API in 64-bit implementations.
See also	<code>get-gc-parameters</code>

set-minimum-free-space *Function*

Summary	Sets the minimum free space for a segment of the specified generation in 32-bit LispWorks.	
Package	<code>hcl</code>	
Signature	<code>set-minimum-free-space gen-num size &optional segment => generation-size</code>	
Arguments	<code>gen-num</code>	The generation to be affected.
	<code>size</code>	The size (in bytes) to set the segment to.
	<code>segment</code>	An integer specifying the segment to be affected. The default value is 0, meaning the first segment of the generation.

Values	<code>generation-size</code>	A list showing information for the generation just specified in the call.
Description	<code>set-minimum-free-space</code>	Sets the minimum free space for a segment of the specified generation. By default, affects the first segment — pass <code>segment</code> to affect a different segment of the generation. The minimum free space is shown by <code>room</code> . Note: <code>set-minimum-free-space</code> is implemented only in 32-bit LispWorks. It is not relevant to the Memory Management API in 64-bit implementations.
See also		<code>clean-generation-0</code> <code>collect-generation-2</code> <code>collect-highest-generation</code> <code>expand-generation-1</code> <code>room</code> <code>set-promotion-count</code>

		<i>Function</i>
Summary	<code>Controls the set of processes that are profiled.</code>	
Package	<code>hcl</code>	
Signature	<code>set-process-profiling <i>flag processes</i></code>	
Arguments	<code>flag</code>	<code>:add, :remove or :set.</code>
	<code>processes</code>	One of <code>:current, :all, a mp:process object, or a list of mp:process objects which may also contain :current.</code>

Description	<p>The function <code>set-process-profiling</code> modifies the set of processes for which profiling information is (or will be) collected.</p> <p>If <code>set-process-profiling</code> is called while profiling (that is after a call to <code>start-profiling</code> and before the next call to <code>stop-profiling</code> with <code>print non-nil</code>) the system immediately starts collecting profile information for the new set of processes.</p> <p>When <code>start-profiling</code> is called without passing <i>processes</i>, it sets the processes to profile according to the last call to <code>set-process-profiling</code>.</p> <p><i>flag</i> determines how the set of processes to profile is modified:</p> <ul style="list-style-type: none"> <code>:add</code> The given processes are added to the set. <code>:remove</code> The given processes are removed from the set. <code>:set</code> The given processes are used as the set. <p><i>processes</i> controls which processes are added to the set, removed from the set or are contained in the set, as follows:</p> <ul style="list-style-type: none"> <code>:current</code> Means the current process. When <code>start-profiling</code> is called it interprets <code>:current</code> to mean the current process at the time it is called. If <code>set-process-profiling</code> is called while profiling, <code>:current</code> is interpreted as the current process when <code>set-process-profiling</code> is called. <code>:all</code> Means all processes, including those which are created after profiling started. <p>A <code>mp:process</code> object</p> <p>Means that process.</p>
-------------	---

	A list	Means the processes in that list. The list can contain the symbol <code>:current</code> , which is interpreted as described above.
		<code>set-process-profiling</code> can be called whether or not the profiler is collecting information. See <code>start-profiling</code> and <code>stop-profiling</code> .
Examples	Add <code>process1</code> to the set:	<pre>(set-process-profiling :add process1)</pre>
	Turn off profiling for the current process:	<pre>(set-process-profiling :remove :current)</pre>
	Turn off all profiling:	<pre>(set-process-profiling :remove :all)</pre>
	Set all processes for later profiling:	<pre>(set-process-profiling :set :all)</pre>
See also	<code>profile</code> <code>start-profiling</code> <code>stop-profiling</code>	

set-profiler-threshold *Function*

Summary	Sets the percentage threshold for symbols to be profiled in a subsequent run.
Package	<code>hcl</code>
Signature	<code>set-profiler-threshold value => value</code>
Arguments	<code>value</code> must be a fixnum between 0 and 100.
Values	<code>set-profiler-threshold</code> returns <code>value</code> .

Description	This function sets the value of <code>*profiler-threshold*</code> below which symbols are not profiled in a repeated profiling run. After a profiling run, all the symbols being profiled have a percentage value for the amount of time they were on the top of the stack. If <code>*profiler-threshold*</code> is set to 40 then by running <code>reset-profiler</code> with argument <code>:top</code> all symbols which are found on the top of the stack less than forty percent of the time are removed from the list of those symbols considered for profiling.
Example	<code>(set-profiler-threshold 40)</code>
See also	<code>reset-profiler</code> <code>profile</code> <code>*profiler-threshold*</code>

set-promotion-count

Function

Summary	Controls when objects can be promoted to the next generation in 32-bit LispWorks.	
Package	<code>hcl</code>	
Signature	<code>set-promotion-count <i>gen-num count</i> &optional <i>segment</i> => <i>count</i></code>	
Arguments	<i>gen-num</i>	The generation number affected.
	<i>count</i>	The number of garbage collections survived by objects in that generation, before promotion. If <i>count</i> is <code>nil</code> , the function returns the current promotion count setting.
	<i>segment</i>	An integer specifying which segment of the generation is to be affected. The default is 0, meaning the lowest segment of the generation.

Values	Returns <i>count</i> .
Description	Controls how many garbage collections an object in a segment must survive before promotion to the next generation. Note: <code>set-promotion-count</code> is implemented only in 32-bit LispWorks. It is not relevant to the Memory Management API in 64-bit implementations, wherein you may be able to achieve the effect with <code>set-delay-promotion</code> .
See also	<code>block-promotion</code> <code>clean-generation-0</code> <code>collect-generation-2</code> <code>collect-highest-generation</code> <code>expand-generation-1</code>

set-up-profiler *Function*

Summary	Declares the parameter values of the profiling function.	
Package	<code>hcl</code>	
Signature	<code>set-up-profiler &key symbols packages kind interval limit cutoff collapse style gc call-counter show-unknown-frames</code>	
Arguments	<code>symbols</code>	A symbol or a list of symbols.
	<code>packages</code>	A valid package name, or a list of package names, or <code>:all</code> .
	<code>kind</code>	<code>:profile</code> , <code>:virtual</code> OR <code>:real</code> .
	<code>interval</code>	An integer greater than or equal to 10000.
	<code>limit</code>	An integer or <code>nil</code> .
	<code>cutoff</code>	An integer or <code>nil</code> .
	<code>collapse</code>	A generalized boolean.
	<code>style</code>	<code>:tree</code> , <code>:list</code> OR <code>nil</code> .

<i>gc</i>	A generalized boolean.
<i>call-counter</i>	A generalized boolean.
<i>show-unknown-frames</i>	
	A generalized boolean.
Values	The time interval is returned.
Description	<p><code>set-up-profiler</code> is used to declare the values of the parameters of the profiling function. Three values are required, as follows.</p> <p><i>symbols</i>, if non-<code>nil</code>, specifies which symbols are to be monitored by the profiler. Each symbol in <i>symbols</i> is checked to see if it is suitable for profiling and if so it is added to the list <code>*profile-symbol-list*</code>.</p> <p>If <i>symbols</i> is not passed then <i>packages</i> specifies which symbols are to be monitored. If <i>packages</i> is <code>:all</code>, then all packages are monitored. All the symbols in the packages are checked as above. If a <i>symbols</i> argument is present then <i>packages</i> is ignored.</p> <p><i>kind</i> specifies the way that the time between samples is measured on Unix-like platforms:</p> <ul style="list-style-type: none"> <code>:profile</code> Process time only. <code>:virtual</code> Process time and system time for the process. <code>:real</code> Real time. <p>The default value of <i>kind</i> is <code>:profile</code>.</p> <p>Note: <i>kind</i> is ignored on Microsoft Windows platforms.</p> <p><i>interval</i> specifies the interval in microseconds between profile samples. The minimum value of <i>interval</i> is 10000, that is 10 ms. The default value of <i>interval</i> is 10000.</p>

limit, when non-`nil`, sets `*default-profiler-limit*`. This limits the maximum number of lines printed in the profile output (not including the tree). The default value is 100.

cutoff, when non-`nil`, sets `*default-profiler-cutoff*`. This is the default minimum percentage that the profiler will display in the output tree. Functions below this percentage will not be displayed. The default is `nil`, that is there is no cutoff.

collapse specifies whether functions with only one callee in the profile tree should be collapsed, that is, only the child is printed. When passed, sets `*default-profiler-collapse*`. The default value of *collapse* is `nil`.

style controls the format of output. If *style* is not passed or passed as `nil`, the format does not change. If *style* is passed, it can take these values:

- | | |
|--------------------|--|
| <code>:list</code> | The profiler will show the functions seen on the stack. |
| <code>:tree</code> | The profiler will generate a tree of calls seen in the profiler, as well as the output shown by <code>:list</code> . |

The default value of *style* is `:tree`.

gc specifies whether to profile functions inside the memory management code (more accurately, functions that are called on the GC stack) in addition to any other profiling. The default value of *gc* is `nil`.

call-counter whether to add extra code to count calls. The counting is done dynamically. If *call-counter* is `nil`, call counters are not added, and the call counter of all functions is displayed as 0. The default value of *call-counter* is `t`.

Note: Call counting can affect performance significantly on some platforms. To get accurate timing (in scales of a few percentage points), pass *call-counter* `nil`. However, in most cases the profiler is used to find bottlenecks where the slowdown is

hundreds of percentage points and so the effect of call counting is less significant.

Note: *call-counter* is effective only on x86 platforms or in 64-bit LispWorks. On non-x86 platforms 32-bit LispWorks decides whether to do call counting for each function when it is compiled, depending on the debug level, and *call-counter* has no effect.

show-unknown-frames controls whether the profile tree shows nodes where the name of the function is unknown. The default value of *show-unknown-frames* is `nil`.

Example

```
(set-up-profiler :symbols '(car cdr)
                 :interval 50000)
```

On Unix/Linux/Mac OS X:

```
(set-up-profiler :symbols '(car cdr)
                 :kind :profile :interval 50000)
```

See also

```
add-symbol-profiler
*default-profiler-collapse*
*default-profiler-cutoff*
*default-profiler-limit*
profile
*profile-symbol-list*
remove-symbol-profiler
```

sets-who

Function

Summary

Lists special variables set by a definition.

Package

`hcl`

Signature

```
sets-who function => result
```

Arguments

function

A symbol or a function dspec.

Values	<code>result</code>	A list.
Description	The function <code>sets-who</code> returns a list of the special variables set by the definition named by <i>function</i> .	 Note: The cross-referencing information used by <code>sets-who</code> is generated when code is compiled with source-level debugging switched on.
See also	<code>who-sets</code> <code>toggle-source-debugging</code>	

source-debugging-on-p *Function*

Summary	Tests if source level debugging is on for compiled code.	
Package	<code>hcl</code>	
Signature	<code>source-debugging-on-p => bool</code>	
Arguments	None.	
Values	<code>bool</code>	If <code>t</code> , source level debugging is on.
Description	Returns <code>t</code> if source level debugging is on for compiled code; otherwise returns <code>nil</code> .	
See also	<code>toggle-source-debugging</code>	

start-profiling *Function*

Summary	Starts collecting profiling information.
Package	<code>hcl</code>

Signature	<code>start-profiling &key <i>initialize processes</i></code>	
Arguments	<i>initialize</i>	A boolean.
	<i>processes</i>	One of <code>:current</code> , <code>:all</code> , a <code>mp:process</code> or a list of <code>mp:process</code> objects.
Description	<p>The function <code>start-profiling</code> starts collecting profiling information.</p> <p>If <code>initialize</code> is non-<code>nil</code> any profiling information collected so far is discarded. The default value of <code>initialize</code> is <code>t</code>.</p> <p>If <code>processes</code> is supplied, the set of processes that will be profiled is set as if by calling:</p> <pre>(set-process-profiling :set :processes <i>processes</i>)</pre> <p>Otherwise, the set of processes remains unchanged, so is controlled by any previous calls to <code>set-process-profiling</code>.</p> <p>Only processes that are active are profiled.</p> <p><code>start-profiling</code> can be repeatedly called without intervening calls to <code>stop-profiling</code>, for example to change the profiled processes.</p> <p><code>start-profiling</code> cannot be used while <code>profile</code> is used or while the Profiler tool is profiling (on any thread). Between the call to <code>start-profiling</code> and the next call to <code>stop-profiling</code> with <code>print t</code> (or omitted), <code>profile</code> and the Profiler tool cannot be used.</p> <p>Various parameters which are set by <code>set-up-profiler</code> control the behavior of the profiler. See the documentation for <code>set-up-profiler</code>.</p>	
Examples	<p>The following sequence of calls to <code>start-profiling</code> and <code>stop-profiling</code> can be used to profile only interesting work and print the results:</p> <p>Start profiling the current process:</p>	

```
(start-profiling :processes :current)
(do-interesting-work)
```

Temporarily suspend profiling:

```
(stop-profiling :print nil)
(do-uninteresting-work)
```

Resume profiling:

```
(start-profiling :initialize nil)
(do-more-interesting-work)
(stop-profiling)
```

See also

`profile`
`set-process-profiling`
`stop-profiling`

stop-profiling *Function*

Summary Stops collecting profiling information.

Package `hcl`

Signature `stop-profiling &key print stream`

Arguments `print` A generalized boolean.

`stream` An output stream.

Description The function `stop-profiling` stops collecting profiling information, and optionally prints the results.

If `print` is `non-nil`, the information collected so far is printed and the next call to `start-profiling` must pass `initialize t` or omit the `initialize` argument. If `print` is `nil`, then the profiler is put into a suspended state where no profiling information is collected, but can be restarted by calling

```
(start-profiling :initialize nil)
```

The default value of *print* is `t`.

stream specifies the stream for output when *print* is non-`nil`. It is ignored when *print* is `nil`. The default value of *stream* is the value of `*trace-output*`.

Note: parameters set by `set-up-profiler` control the format of the output.

See also

`profile`
`set-process-profiling`
`start-profiling`

sweep-all-objects

Function

Summary

Applies a function to all the live objects in the image.

Package

`hcl`

Signature

`sweep-all-objects function &optional gen-0`

Arguments

function A function of one argument, the object.

gen-0 A generalized boolean, default value `nil`

Values

`sweep-all-objects` returns `nil`.

Description

Applies *function* to all the live objects in the image. Normally it is not useful to sweep objects in generation 0 because they are ephemeral, so by default `sweep-all-objects` does not sweep generation 0. This can be changed by passing a non-`nil` value as *gen-0*.

function should take one argument, the object. It can allocate, but if it allocates heavily the sweeping becomes unreliable. Small amounts of allocation will normally happen only in generation 0, and so will not affect sweeping of other generations.

Note: in 64-bit LispWorks there is a more specific alternative:
 function `sweep-gen-num-objects` can be used to call a function on all live objects in a particular generation.

See also `sweep-gen-num-objects`

switch-static-allocation *Function*

Summary	Controls whether objects are allocated in the static area.
Package	<code>hcl</code>
Signature	<code>switch-static-allocation flag => previous-flag</code>
Arguments	<p><i>flag</i> If <i>flag</i> is <code>non-nil</code>, subsequent objects are allocated in the static area; if <i>flag</i> has any other value, objects are allocated conventionally.</p>
Values	<code>switch-static-allocation</code> returns the previous setting of <i>flag</i> .
Description	<p>Objects in the static area are garbage-collected, but not moved.</p> <p>You should avoid using this function.</p>
See also	<code>enlarge-static</code> <code>in-static-area</code>

symbol-alloc-gen-num *Variable*

Summary	Specifies the generation in which interned symbols and their symbol names are allocated.
---------	--

Package	<code>hcl</code>
Initial Value	2 in 32-bit LispWorks, 3 in 64-bit LispWorks
See also	<code>allocation-in-gen-num</code> <code>get-default-generation</code> <code>set-default-generation</code>

	<i>Function</i>
toggle-source-debugging	<i>Function</i>
Summary	Changes compiler settings affecting production of source level debugging information.
Package	<code>hcl</code>
Signature	<code>toggle-source-debugging &optional <i>on</i> => <i>bool</i></code>
Arguments	<code>on</code> Flag (<code>t</code> or <code>nil</code>) to control the resulting setting of the variables. The default is <code>t</code> .
Values	<code>bool</code> The current state of source level debugging: <code>t</code> if source level debugging is on.
Description	<p><code>toggle-source-debugging</code> sets certain compiler parameters, and also turns leaf case optimizations on (when called with <code>nil</code>) or off (when called with <code>t</code>). For all these parameters, the value <code>nil</code> reduces compilation speed.</p> <p><code>toggle-source-debugging</code> is called in the configuration file <code>a-dot-lispworks.lisp</code>, and the initial state of LispWorks such that source level debugging is on.</p> <p>The parameters relate to information required for source level debugging, cross-referencing and finding all changed definitions.</p> <p>The parameters (all in the <code>compiler</code> package) are:</p>

produce-xref-info

When true, the compiler produces information for the Cross Referencer.

load-xref-info

When true, the cross-referencing information produced by the compiler is loaded when the corresponding file is loaded.

notice-changed-definitions

When true, the Cross Referencer notices when a function is redefined, including an interpreted redefinition..

source-level-debugging

When true, the compiler generates information used by the debugger.

`toggle-source-debugging` modifies the status of the variables, and then returns the new value. To check whether all the variables are set to true, without modifying them, use `source-debugging-on-p`.

Cross-referencing information is used by the functions `who-calls`, `who-binds`, `who-references`, `who-sets`, and friends.

Compatibility Note In LispWorks 4.2 and earlier, `toggle-source-debugging` controlled source file recording information. In LispWorks 4.3 and later, this is controlled independently by `*record-source-files*`.

See also `source-debugging-on-p`

total-allocation*Function*

Summary Calculate memory consumed since the image was started.

Package	<code>hcl</code>
Signature	<code>total-allocation</code>
Arguments	None.
Values	Returns the amount allocated
Description	This function calculates the total amount of memory consumed since the current image was created. Use at the start and end of a piece of code, to see how much it allocates.
See also	<code>find-object-size</code> <code>room</code>

traced-arglist *Variable*

Summary	The list of arguments given to the function being traced.
Package	<code>hcl</code>
Initial Value	<code>nil</code>
Description	Upon entering a function that is being traced, <code>*traced-arglist*</code> is bound to the list of arguments given to the function. <code>*traced-arglist*</code> is then printed after the function name in the output from tracing. It is accessible in the <code>:before</code> and <code>:after</code> forms to trace. However care should be used when manipulating this variable, since it is the value of <code>*traced-arglist*</code> itself that is used when calling the traced function. Thus if this value is altered by the <code>:before</code> forms then the function receives the altered argument list.

Example `USER 14 > (trace (+ :before
 ((setq *traced-arglist*
 (mapcar #'1+
 traced-arglist)))))`

```

+
USER 15 > (+ 1 2 3)

0 + > (1 2 3)
(2 3 4)
0 + < (9)
9

```

Notes *traced-arglist* is an extension to Common Lisp.

See also **trace**

traced-results *Variable*

Summary The list of results from the function being traced.

Package **hcl**

Initial Value **nil**

Description Upon leaving a function that is being traced, *traced-results* is bound to the list of results from the function. *traced-results* is then printed after the function name in the output from tracing. It is accessible in the :after forms to trace. However care should be used when manipulating this variable, since it is the value of *traced-results* itself that is used when returning from the traced function. Thus if this value is altered by the :after forms then the caller of the traced function receives the altered results.

Example

```
USER 5 > (trace (ceiling
                      :after
                      ((setq *traced-results*
                            (mapcar #'1- *traced-results*)))))
```


CEILING

```
USER 6 > (multiple-value-call #'+ (ceiling 4 3))
```

```
0 CEILING > (4 3)
0 CEILING < (2 -2)
(1 -3)
-2
```

Notes *traced-results* is an extension to Common Lisp.

See also `trace`

trace-indent-width

Variable

Summary The amount of extra indentation in the trace output for each level of nesting.

Package `hcl`

Initial Value 2

Description *trace-indent-width* is the extra amount by which the traced output for function calls is indented upon entering a deeper level of nesting (i.e. a traced call from a function that is itself traced). If it is 0 then no indentation occurs.

Example

```
USER 7 > (setq *trace-indent-width* 4
                  *max-trace-indent* 50)
```

```
50
USER 8 > (defun quad (a b c) (- (* b b) (* 4 a c)))
QUAD
USER 9 > (trace quad *)
*
USER 10 > (quad 4 3 14)

0 QUAD > (4 3 14)
  1 * > (3 3)
  1 * < (9)
  1 * > (4 4 14)
  1 * < (224)
0 QUAD < (-215)
-215
```

Notes ***trace-indent-width*** is an extension to Common Lisp.

See also **trace**

trace-level *Variable*

Summary The current depth of tracing.

Package **hcl**

Initial Value 0

Description ***trace-level*** is a special variable whose value is the current depth of tracing. The current value of ***trace-level*** is printed before the function name during the output from tracing.

Example

```

USER 8 > (defun fac (n) (if (<= n 1)
                               1
                               (* n (fac (1- n)))))
FAC
USER 9 > (trace fac)

FAC
USER 10 > (fac 3)

0 FAC > (3)
  1 FAC > (2)
    2 FAC > (1)
    2 FAC < (1)
    1 FAC < (2)
  0 FAC < (6)
  6

```

Notes ***trace-level*** is an extension to Common Lisp.

See also **trace**

trace-print-circle		<i>Variable</i>
Summary	Controls how circular structure are printed in trace output.	
Package	<code>hcl</code>	
Initial Value	<code>nil</code>	
Description		<code>*trace-print-circle*</code> controls how circular structures are printed during output from tracing. It allows the printing of circular structures by the tracer to be controlled independently of the usual printing mechanism, which is governed by <code>*print-circle*</code> . <code>*print-circle*</code> is bound to the value of <code>*trace-print-circle*</code> while printing tracing information.
Example		<pre> USER 19 > (setq *trace-print-circle* t) T USER 20 > (defun circ (l) (rplacd (last l) l) l) CIRC USER 21 > (trace second) SECOND USER 22 > (second (circ '(1 2 3 4))) 0 SECOND > (#1=(1 2 3 4 . #1#)) 0 SECOND < (2) 2 </pre>
Notes		<code>*trace-print-circle*</code> is an extension to Common Lisp.
See also		<code>trace</code>

trace-print-length		<i>Variable</i>
Summary	The number of components of an object that are printed in trace output.	
Package	<code>hcl</code>	

Initial Value	100
Description	<p>*trace-print-length* controls the number of components of an object which are printed during output from tracing. If its value is a positive integer then the first *trace-print-length* components are printed.</p> <p>*print-length* is bound to the value of *trace-print-length* while printing tracing information. If *trace-print-length* is nil then all the components of the object are printed.</p>
Example	<pre>USER 5 > (trace append) APPEND USER 6 > (setq *trace-print-length* 3) 3 USER 7 > (dotimes (i 10) (setq li (if (zerop i) nil (cons i li))))) NIL USER 8 > (append li '(a b)) 0 APPEND > ((9 8 7 ...) (A B)) 0 APPEND < ((9 8 7 ...)) (9 8 7 6 5 4 3 2 1 A B)</pre>
Notes	*trace-print-length* is an extension to Common Lisp.
See also	<code>trace</code>

trace-print-level		<i>Variable</i>
Summary		The depth to which nested objects are printed in trace output.
Package		<code>hcl</code>
Initial value		5

Description	<p><code>*trace-print-level*</code> controls the depth to which nested objects are printed during output from tracing. If its value is a positive integer then components at or above that level are suppressed. By definition an object to be printed is considered to be at level 0, its components are at level 1, their sub-components are at level 2, and so on.</p> <p><code>*print-level*</code> is bound to the value of <code>*trace-print-level*</code> while printing tracing information. If <code>*trace-print-level*</code> is <code>nil</code> then objects are printed without regard to depth.</p>
Examples	<pre>USER 8 > (trace append) APPEND USER 9 > (dotimes (i 10) (setq li (if (zerop i) nil (list i li))))) NIL USER 10 > (append li '(a b)) 0 APPEND > ((9 (8 (7 (6 #)))) (A B)) 0 APPEND < ((9 (8 (7 (6 #)))) A B)) (9 (8 (7 (6 (5 (4 (3 (2 (1 NIL))))))) A B)</pre>
Notes	<code>*trace-print-level*</code> is an extension to Common Lisp.
See also	<code>trace</code>

trace-print-pretty		<i>Variable</i>
Summary	Controls the amount of whitespace in trace output.	
Package	<code>hcl</code>	
Initial Value	<code>nil</code>	
Description	<code>*trace-print-pretty*</code> controls the amount of whitespace printed during output from tracing. If it is not <code>nil</code> then extra	

whitespace is inserted to make the output more comprehensible. ***print-pretty*** is bound to the value of ***trace-print-pretty*** while printing tracing information.

Examples

```

USER 6 > (trace macroexpand-1)

MACROEXPAND-1
USER 7 > (setq *trace-print-pretty* t
                 *print-pretty* nil)

NIL
USER 8 > (defmacro sum (n)
            '(do ((i 0 (1+ i))
                  (res 0 (+ i res)))
                 ((= i ,n) res)))

SUM
USER 9 > (macroexpand-1 '(sum 3))

0 MACROEXPAND-1 > ((SUM 3))
0 MACROEXPAND-1 < ((DO ((I 0 (1+ I))
                           (RES 0 (+ I RES)))
                           ((= I 3)
                            RES))
                           T)
                           (DO ((I 0 (1+ I)) (RES 0 (+ I RES))) ((= I 3) RES))
                           T

```

Notes

trace-print-pretty is an extension to Common Lisp.

See also

trace

trace-verbose

Variable

Summary Controls how arguments and values are printed in trace output.

Package **hcl**

Initial Value **:only**

Description	<p>*<code>trace-verbose</code>* controls the way arguments and values are printed in trace output.</p> <p>If the value is not <code>nil</code> then <code>trace</code> attempts to decode the arguments and values, and prints them.</p> <p>When the value is <code>:only</code>, <code>trace</code> does not print the lists of arguments and values after the function name.</p>
Notes	* <code>trace-verbose</code> * is an extension to Common Lisp.
See also	<code>trace</code>

try-compact-in-generation *Function*

Summary	Compacts the most fragmented segment(s) in a generation in 32-bit LispWorks.
Package	<code>hcl</code>
Signature	<code>try-compact-in-generation <i>generation-number time-threshold</i>&optional <i>fraction-threshold</i> => <i>result</i></code>
Arguments	<p><i>generation-number</i></p> <p>0 for the most recent generation, 1 for the most recent two generations, and so on up to a maximum (usually 3). Numbers outside this range signal an error.</p> <p><i>time-threshold</i> A real number.</p> <p><i>fraction-threshold</i> A real number between 0 and 1, defining the minimum fragmentation to actually compact. The default is 0.25.</p>
Values	<i>result</i> A boolean.

Description	<code>try-compact-in-generation</code> finds the most fragmented segment in the generation specified. If <i>time-threshold</i> is positive, it compacts this segment, and repeats this operation until <i>time-threshold</i> seconds have elapsed. At this point <code>try-compact-in-generation</code> returns, with value <code>t</code> if at least one segment was compacted and value <code>nil</code> otherwise. Because the operation cannot be stopped in the middle, the actual time taken will always be larger than <i>time-threshold</i> . If <i>fraction-threshold</i> is 1, <code>try-compact-in-generation</code> does nothing. If <i>fraction-threshold</i> is 0, <code>try-compact-in-generation</code> will compact all uncompacted segments (unless it runs out of time). With the default (0.25) <code>try-compact-in-generation</code> compacts only moderately fragmented segments. If <i>time-threshold</i> is negative, then <code>try-compact-in-generation</code> does not actually compact any segments. <i>result</i> is a boolean indicating whether <code>try-compact-in-generation</code> would actually try to compact a segment if it were to be called with a positive <i>time-threshold</i> and the other arguments unchanged.
See also	<code>check-fragmentation</code> <code>try-move-in-generation</code>

	<i>Function</i>
Summary	Moves objects out of the most fragmented segment(s) in a generation, leaving them empty in 32-bit LispWorks.
Package	<code>hcl</code>
Signature	<code>try-move-in-generation generation-number time-threshold &optional fraction-threshold => result</code>
Arguments	<p><i>generation-number</i> 0 for the most recent generation, 1 for the most recent two generations, and so on up to a maximum (usually 3). Numbers outside this range signal an error.</p> <p><i>time-threshold</i> A real number.</p> <p><i>fraction-threshold</i> A real number between 0 and 1, defining the minimum fragmentation to actually move. The default is 0.25.</p>
Values	<i>result</i> A boolean.
Description	<p><code>try-move-in-generation</code> finds the most fragmented segment in the generation specified. If <i>time-threshold</i> is positive, it moves objects out of this segment, leaving it empty, and repeats this operation until <i>time-threshold</i> seconds have elapsed. At this point <code>try-move-in-generation</code> returns, with value <code>t</code> if at least one segment was moved and value <code>nil</code> otherwise. Because the operation cannot be stopped in the middle, the actual time taken will always be larger than <i>time-threshold</i>.</p> <p>If <i>fraction-threshold</i> is 1, <code>try-move-in-generation</code> does nothing. If <i>fraction-threshold</i> is 0, <code>try-move-in-generation</code> will move all uncompacted segments (unless it runs out of time).</p>

With the default (0.25) `try-move-in-generation` moves only moderately fragmented segments.

If `time-threshold` is negative, then `try-move-in-generation` does not actually move any segments. `result` is a boolean indicating whether `try-move-in-generation` would actually try to move a segment if it were to be called with a positive `time-threshold` and the other arguments unchanged.

This function is typically used after a call to `check-fragmentation`. For more information, see the LispWorks User Guide.

Note: `try-move-in-generation` is implemented only in 32-bit LispWorks. It is not relevant to the Memory Management API in 64-bit implementations, where `marking-gc` with the `what-to-copy` argument offers similar functionality (although `set-blocking-gen-num` is intended to solve the problem of fragmentation automatically).

See also `check-fragmentation`
 `try-compact-in-generation`

who-binds	<i>Function</i>
Summary	Returns the definitions which bind a special variable.
Package	<code>hcl</code>
Signature	<code>who-binds symbol => result</code>
Arguments	<code>symbol</code> A special variable.
Values	<code>result</code> A list.
Description	The function <code>who-binds</code> returns a list of dspecs naming the definitions which bind the special variable <code>symbol</code> .

Note: The cross-referencing information used by `who-binds` is generated when code is compiled with source-level debugging switched on.

See also `binds-who`
 `toggle-source-debugging`

who-calls *Function*

Summary Returns the callers of a function.

Package `hcl`

Signature `who-calls dspec => callers`

Arguments `dspec` A `dspec`.

Values `callers` A list.

Description The function `who-calls` returns a list of `dspecs` naming the definitions which call the function named by `dspec`.
See also the Editor commands `List Callers` and `Show Paths To`.

Note: The cross-referencing information used by `who-calls` is generated when code is compiled with source-level debugging switched on.

See also `calls-who`
 `toggle-source-debugging`

who-references *Function*

Summary Returns the definitions which reference a special variable.

Package	<code>hcl</code>	
Signature	<code>who-references symbol => result</code>	
Arguments	<code>symbol</code>	A special variable.
Values	<code>result</code>	A list.
Description	The function <code>who-references</code> returns a list of dspecs naming the definitions which reference the special variable <i>symbol</i> .	
	<p>Note: The cross-referencing information used by <code>who-references</code> is generated when code is compiled with source-level debugging switched on.</p>	
See also	<code>references-who</code> <code>toggle-source-debugging</code>	

		<i>Function</i>
who-sets		
Summary	Returns the definitions which set a special variable.	
Package	<code>hcl</code>	
Signature	<code>who-sets symbol => result</code>	
Arguments	<code>symbol</code>	A special variable.
Values	<code>result</code>	A list.
Description	The function <code>who-sets</code> returns a list of dspecs naming the definitions which set the value of the special variable <i>symbol</i> .	
	<p>Note: The cross-referencing information used by <code>who-sets</code> is generated when code is compiled with source-level debugging switched on.</p>	

See also **`sets-who`**
 `toggle-source-debugging`

with-heavy-allocation *Macro*

Summary	Slows up garbage collection during the execution of code that allocates a lot of space.	
Package	<code>hcl</code>	
Signature	<code>with-heavy-allocation &rest body => result</code>	
Arguments	<i>body</i>	The forms for which you want the garbage collector to behave differently from normal.
Values	<i>result</i>	The result of executing <i>body</i> .
Description	The macro <code>with-heavy-allocation</code> is for use with code that allocates a lot of space but is not interactive. It ensures that garbage collection (GC) is carried out less frequently while these forms are being executed. However, each GC may take longer.	
Compatibility note	In LispWorks 5.0 <code>with-heavy-allocation</code> is implemented only in 32-bit LispWorks. In version 5.1 and later it is implemented in 64-bit LispWorks as well.	
See also	<code>avoid-gc</code> <code>gc-if-needed</code> <code>get-gc-parameters</code> <code>mark-and-sweep</code> <code>normal-gc</code> <code>set-gc-parameters</code> <code>finish-heavy-allocation</code> <code>without-interrupts</code>	

	<i>Function</i>
with-output-to-fasl-file	
Summary	Sends output to a fasl file on disk.
Package	<code>hcl</code>
Signature	<code>with-output-to-fasl-file (stream pathname &rest options)</code> <code>&body body => nil</code>
Arguments	<p><code>stream</code> Stream to be bound to the fasl file to be created.</p> <p><code>pathname</code> Name of the fasl file to be created.</p> <p><code>body</code> Forms, some of which may be dumped.</p>
Values	Returns <code>nil</code> .
Description	<p><code>with-output-to-fasl-file</code> is used in conjunction with <code>dump-form</code>. The <code>body</code> forms are executed, and during the execution, <code>dump-form</code> may be called to dump selected forms. Dumped forms are evaluated if the file <code>pathname</code> is later loaded by <code>load-data-file</code>.</p> <p>Supply an appropriate fasl extension in <code>pathname</code>. A simple way to achieve this is by calling <code>compile-file-pathname</code>. A complete list of fasl extensions for supported platforms may be found in <code>compile-file</code>.</p> <p>If the file <code>pathname</code> already exists, it is superseded.</p> <p>A fasl file created using <code>with-output-to-fasl-file</code> must be loaded only by <code>load-data-file</code>, and not by <code>load</code>.</p>

Example

```
CL-USER 12 > (with-output-to-fasl-file (s
  "/tmp/foo.fasl")
    (dump-form '(print 'hello) s))
NIL

CL-USER 13 > (let ((sys:*binary-file-type* "fasl"))
  (sys:load-data-file "/tmp/foo.fasl"))
; Loading fasl file "/tmp/foo.fasl"

HELLO
#P"/tmp/foo.fasl"
```

See also

`dump-form`
`dump-forms-to-file`
`load-data-file`

9

The LINK-LOAD Package

This chapter describes the symbols in the `LINK-LOAD` package.

Note: this chapter applies only to LispWorks for UNIX only (not LispWorks for Linux or LispWorks for FreeBSD).

		<i>Function</i>
Package	<code>link-load</code>	
Signature	<code>break-on-unresolved-functions &optional stream</code>	
Arguments	<code>stream</code>	An output stream for message reporting. If set to <code>nil</code> , then no output will be produced. By default this is <code>t</code> .
Description		The <code>break-on-unresolved-functions</code> function produces break-on-entry code for all currently undefined but referenced (that is, unresolved) foreign symbols, so that if an undefined foreign function is called from within the foreign code, a Lisp error will occur. Break-on-entry code will also be

produced for any new unresolved symbols loaded later in your Lisp session.

The special variable `foreign:*break-on-unresolved-functions*` will, when set to non-`nil`, produce break-on-entry code for all new unresolved symbols that are loaded, but won't do so for symbols already loaded. By default this variable is set to `nil`.

See also [read-foreign-modules](#)

foreign-symbol-address *Function*

Package [link-load](#)

Signature `foreign-symbol-address name &key errorp functionp => result`

Arguments `name` The name of a foreign symbol.
`errorp` A boolean.
`functionp` A boolean.

Values `result` The address of `name` or `nil`.

Description The `foreign-symbol-address` function is used to find out whether a foreign symbol is defined, by looking for it in the foreign-symbol table. If its associated object code has been loaded into the image, its address is returned. Otherwise `nil` is returned, unless `errorp` is `nil`.

The `errorp` keyword defines the behavior of the function when a symbol has not been defined. If it is non-`nil` (which is the default value), then an error will be signalled. If it is `nil`, no error will be reported, and the function will return `nil`.

The `functionp` keyword is used to specify the kind of symbol sought. If it is `t`, `foreign-symbol-address` will assume that

name is the name of a function. If it is `nil` it will assume that *name* is the name of a variable. The default value is `t`.

Example `(foreign-symbol-address 'chmod)`

See also `get-foreign-symbol`

get-foreign-symbol

Function

Package `link-load`

Signature `get-foreign-symbol name &optional force => result`

Arguments *name* A symbol or string.

force A keyword.

Values *result* A foreign symbol.

Description This function gets a foreign symbol or it may be used to explicitly register an undefined symbol.

name is a symbol or string to look up or to create as a foreign symbol. If it is a symbol, the symbol looked for is that which the function `lisp-name-to-foreign-name` would produce. If *name* is a string, it is taken literally

If supplied and the symbol is not already defined as a foreign symbol, *force* forces it to be an undefined foreign symbol. This provides a reference to the symbol so that a subsequent call to `read-foreign-modules` will attempt to resolve it

Example `(get-foreign-symbol 'my-func-not-yet-loaded t)`

Notes It is not usually necessary to use this function. In order to examine whether a foreign symbol is defined, use `foreign-symbol-address`. The act of defining a foreign function using

`fli:define-foreign-function` makes the symbol undefined, so the use of force is not usually needed.

See also `foreign-symbol-address`
`lisp-name-to-foreign-name`
`read-foreign-modules`

	<i>Function</i>				
Package	<code>link-load</code>				
Signature	<code>lisp-name-to-foreign-name name &key language</code>				
Arguments	<table border="0"> <tr> <td style="vertical-align: top;">name</td><td>A symbol representing a Lisp name. (Strings are passed unchanged through the function.)</td></tr> <tr> <td style="vertical-align: top;">language</td><td>If :c then an equivalent ‘C’ name is produced. :FORTRAN is an alternative.</td></tr> </table>	name	A symbol representing a Lisp name. (Strings are passed unchanged through the function.)	language	If :c then an equivalent ‘C’ name is produced. :FORTRAN is an alternative.
name	A symbol representing a Lisp name. (Strings are passed unchanged through the function.)				
language	If :c then an equivalent ‘C’ name is produced. :FORTRAN is an alternative.				
Description	This function provides an equivalent foreign name for a Lisp name, depending on the keyword language.				
Values	A string is returned which is a foreign equivalent of the Lisp name supplied. If name is a string, the function returns the string unchanged. If language is a symbol, the ‘C’ version replaces occurrences of ‘-’ with ‘_’ and adds a leading underscore. The Fortran version replaces occurrences of ‘-’ with ‘_’ and adds a leading and trailing underscore.				
Example	<pre>(lisp-name-to-foreign-name 'lisp-name-with-hyphens) "_lisp_name_with_hyphens"</pre>				
See also	<code>get-foreign-symbol</code>				

	<i>Function</i>
read-foreign-modules	
Package	link-load
Signature	<code>read-foreign-modules &rest module-names => t</code>
Arguments	<i>module-names</i> A sequence of strings or pathnames.
Values	<i>t</i>
Description	<p>The <code>read-foreign-modules</code> function reads object files of various formats into the Lisp image. Unresolved references are resolved wherever possible and the names of the foreign functions are made available to the Lisp for direct calling from the Lisp if desired. With no argument, <code>read-foreign-modules</code> scans the default libraries looking for definitions of referenced but undefined symbols.</p> <p>The <i>module-names</i> argument is a sequence of items representing object files to be loaded. The items may be of type string or pathname, and will be used to look up a corresponding file in the file system. The only exception is if an item is a string beginning “-l” in which case the rest of the string is used to look up a library file using format strings constructed from the values of the variable <code>*default-library-name-search-paths*</code>, the environment variable <code>LD_LIBRARY_PATH</code> and the variable <code>*default-library-names*</code>. Object files of various formats and library files can be handled by <code>read-foreign-modules</code>.</p>
Example	<pre>(read-foreign-modules "/usr/users/clc/projects/head.o" "~clc/projects/libs.a" "-lw")</pre>
Notes	The <code>read-foreign-modules</code> function actually adds the module-names to the list of modules in the variable <code>*default-libraries*</code> and then tries to resolve any undefined symbols using this list. The function <code>get-foreign-symbol</code> may be

called to explicitly force a symbol onto the undefined list or the act of defining a foreign function (`fli:define-foreign-function`) will do it implicitly.

`read-foreign-modules` may be called at any time during the running of a program and a particular object file may be loaded as often as is necessary.

A warning of any new unresolved references will be printed out after the reading has finished if the flag `*unresolved-messages*` is set to `t` (the default is `nil`). By default messages are printed out about which object modules are being loaded. This may be switched off by setting `*coff-loading-verbose*` to `nil`.

See also `get-foreign-symbol`

10

The LISPWORKS Package

This chapter describes symbols available in the `LISPWORKS` package. This package is used by default. Its symbols are visible in the `CL-USER` package.

		<i>Type</i>
Summary	The 8 bit string type.	
Package	<code>lispworks</code>	
Signature	<code>8-bit-string length</code>	
Arguments	<code>length</code>	The length of the string (or *, meaning any).
Description		The type of strings that can hold simple chars of codes 0...255. This is the string type that is guaranteed to always take 8 bits per element.

16-bit-string	<i>Type</i>
Summary	The 16 bit string type.
Package	lispworks
Signature	16-bit-string <i>length</i>
Arguments	<i>length</i> The length of the string (or *, meaning any).
Description	The type of strings that can hold simple chars of codes 0...65533. This is the string type that is guaranteed to always take 16 bits per element.
appendf	<i>Macro</i>
Summary	Appends lists to the end of a given list.
Package	lispworks
Signature	appendf <i>place &rest lists => result</i>
Arguments	<i>place</i> A place. <i>lists</i> A set of lists.
Values	<i>result</i> An object.
Description	The modify macro appendf appends the lists given by <i>lists</i> to the end of the list in <i>place</i> . See append for more details.
See also	removef
base-character	<i>Type</i>
Summary	The base character type.

Package	<code>lispworks</code>
Signature	<code>base-character</code>
Description	The type of base characters. <code>base-character</code> is a synonym for the Common Lisp type <code>base-char</code> .
See also	<code>base-char-code-limit</code>

base-character-p *Function*

Summary	Tests if an object is a base character	
Package	<code>lispworks</code>	
Signature	<code>base-character-p object => bool</code>	
Arguments	<code>object</code>	The object to be tested.
Values	<code>bool</code>	<code>t</code> if <code>object</code> is a base character; <code>nil</code> otherwise.
Description	This is the predicate for base characters.	
See also	<code>base-character</code>	

base-char-p *Function*

Summary	Tests if an object is a base character	
Package	<code>lispworks</code>	
Signature	<code>base-char-p object => bool</code>	
Arguments	<code>object</code>	The object to be tested.

Values	<i>bool</i>	<i>t</i> if <i>object</i> is a base character; <i>nil</i> otherwise.
Description	This is also the predicate for base characters, only with standard spelling.	
See also		base-character-p

base-char-code-limit *Constant*

Summary	Upper bound for character codes in base characters.
Package	lispworks
Description	The upper exclusive bound for values of (char-code <i>char</i>) among base characters.

base-string-p *Function*

Summary	Tests if an object is a base string.	
Package	lispworks	
Signature	base-string-p <i>object</i> => <i>bool</i>	
Arguments	<i>object</i>	The object to be tested.
Values	<i>bool</i>	<i>t</i> if <i>object</i> is a base string; <i>nil</i> otherwise.
Description	This is the predicate for base strings.	
See also	base-string	

browser-location	<i>Variable</i>
Signature	<code>*browser-location*</code>
Package	<code>lispworks</code>
Initial Value	<code>nil</code>
Description	<p>Controls how the online documentation interface and the function <code>open-url</code> find a web browser executable (either Netscape, Firefox, Mozilla or Opera) to use. The value should be <code>nil</code> or a string.</p> <p>If the value is <code>nil</code>, LispWorks attempts to find the browser using the value of the environment variable <code>PATH</code>.</p> <p>If the value is a string, it specifies the directory in which the browser is installed. Typical values are <code>"/usr/bin/"</code> and <code>"/usr/local/bin/"</code>.</p> <p>Note: do not omit the trailing slash.</p> <p>Note: <code>*browser-location*</code> is used only in the Motif-based IDE.</p>
See also	<code>open-url</code>

call-next-advice	<i>Function</i>
Summary	Calls the next piece of advice associated with a function.
Package	<code>lispworks</code>
Signature	<code>call-next-advice args</code>
Arguments	<code>args</code> are arguments to be given to the next piece of advice to be called. Any number of arguments may be given in this way, including keyword arguments, and there is no require-

ment for pieces of around advice to receive the same number of arguments as the original definition expected.

Values	<code>call-next-advice</code> returns the values produced by the call to the next piece of advice (or to the combination of before and after advice and the original definition).
Description	<code>call-next-advice</code> is the local function used to invoke the next item in the ordering of pieces of advice associated with a function. It can only be called from within the scope of the around advice. Advice may be attached to a function by <code>defadvice</code> and this allows the behavior of a function to be modified. Extra code to be performed before or after the function may be simply added by creating before or after advice for it. Around advice is more powerful and replaces the original definition. All the advice for a function is ordered with the around advice coming first. The first piece of around advice receives the arguments to the function and may return any values at all. It has access to the rest of the advice, and to the original definition, by means of <code>call-next-advice</code> . A call to this from within the body of the around advice invokes the next piece of around advice with the arguments given to <code>call-next-advice</code> . The last piece of around advice in the ordering invokes the sequence of before advice, the original definition, and after advice if it calls <code>call-next-advice</code> . Around advice may contain any number of calls to <code>call-next-advice</code> , including no calls.
Notes	<code>call-next-advice</code> is an extension to Common Lisp. See the <i>LispWorks User Guide</i> for a broader discussion of <code>advice</code> .
See also	<code>defadvice</code>

compile-system	<i>Function</i>
Summary	The function compile-system compiles all the files in a system necessary to make a consistent set of object files.
Package	lispworks
Signature	compile-system <i>system-name</i> & key <i>force simulate load args target-directory</i>
Arguments	<p><i>system-name</i> A symbol representing the name of the system. The system must have been defined already using the defsystem macro.</p> <p><i>force</i> If t then all the files in the system are compiled regardless. (This argument was formerly called <i>force-p</i>. The old name is currently still accepted for compatibility.)</p> <p><i>simulate</i> If nil or not present then compile-system works silently. Otherwise a plan of the actions which compile-system intends to carry out is printed. What happens next depends on the value of <i>simulate</i>:</p> <ul style="list-style-type: none"> t — do nothing. :ask — you are asked if you wish the plan to be carried out using y-or-n-p. :each — compile-system displays each action in the plan one at a time, and asks you whether you want to carry out this particular action. The answer c executes the rest of the plan without further prompting, returns from compile-system without further processing, and y and n work as expected. <p>:simulate may be abbreviated as :sim.</p>

<i>load</i>	If <i>t</i> then <code>load-system</code> is called after <code>compile-system</code> has finished. If <code>:no</code> then no files are loaded at all. The default is <code>nil</code> .
<i>args</i>	Arguments to be passed directly to the compiler.
<i>target-directory</i>	This must be a string representing a valid directory. It defaults to the <code>:default-pathname</code> option to <code>defsystem</code> . This is the directory where the object files created are put. If the <i>target-directory</i> is given then dependency information expressed in the system rules is ignored. <code>:target-directory</code> may be abbreviated as <code>:t-dir</code> .
Values	<code>compile-system</code> returns <code>nil</code> .
Examples	<pre>(compile-system 'blackboard :simulate :ask) (compile-system 'tms :load t) (compile-system 'packages :load :no :target-directory "/usr/users/386i/")</pre>
Notes	<p>If the <code>:load</code> keyword is set to <i>t</i> then by default <code>load-system</code> is called after <code>compile-system</code>. This behavior can be changed to loading any file immediately after it is compiled by setting the variable <code>defsystem::*load-when-compile*</code> to non-<code>nil</code>.</p> <p>C source files, for example <code>foo.c</code>, can be included in a system (see the use of <code>:default-type</code> and <code>:type</code> in <code>defsystem</code>). The corresponding object file name is <code>foo.so</code> on Linux, and on Unix it is <code>foon.o</code> where <i>n</i> is a platform-specific integer. On Mac OS X the object file name is <code>foo.dylib</code> and on Windows the object file name is <code>foo.dll</code>.</p>
See also	<code>concatenate-system</code> <code>defsystem</code> <code>load-system</code>

	<i>Function</i>
Summary	Produces a single, concatenated fasl from a <code>defsystem</code> system or systems.
Package	<code>scm</code>
Signature	$\text{concatenate-system } \text{output system} \&\text{key force simulate sim}$ $\text{source-only args target-directory t-dir script-p} \Rightarrow \text{result}$ $\text{system} ::= \text{system-name}^*$
Arguments	<p><i>output</i> The name of the required concatenated fasl.</p> <p><i>system-name</i> The name of a system defined using <code>defsystem</code>.</p> <p><i>simulate</i> Verbosity conditions, see Description for more detail.</p> <p><i>sim</i> Same as <i>simulate</i>.</p> <p><i>force</i> If <i>t</i>, then all files in the system will be concatenated.</p> <p><i>source-only</i> If <i>t</i>, the source files of the system are concatenated.</p> <p><i>target-directory</i> The directory to search for the object files.</p> <p><i>t-dir</i> Same as <i>target-directory</i>.</p>
Values	A list containing the name or names of the concatenated systems.
Description	<p>This function produces a single, concatenated fasl, <i>output-file</i>, from a list of individual systems (named amongst the <i>args</i>).</p> <p>Since concatenated fasl files may be produced in this way, you do not need to be wary of MS filename conventions if developing sources on UNIX for a Microsoft Windows application. This clearly allows more freedom for naming source</p>

files. However, *output-file* must, in such cases, be a MS-Window-compatible filename.

If *simulate* is `nil` or is not present, `concatenate-system` will work silently. Otherwise, a plan of the actions which `concatenate-system` intends to carry out is printed. What happens next depends upon the value of *simulate*:

- If it is `t`, the function does nothing.
- If `:ask`, then the user is asked, using `y-or-n-p`, if the plan should be carried out.
- If it is `:each`, the user is asked at each stage in the plan if the current action should be carried out. The responses `y` and `n` work as normal. If `e` is typed, `concatenate-system` exits without further processing.

If *source-only* is `t`, files will be loaded only if they are sources.

If, when searching *target-directory* for an object file, the file cannot be found, the appropriate source file from the system's default directory will be loaded instead.

See also

`compile-system`
`defsystem`
`load-system`

		<i>Function</i>
current-pathname		
Summary	Computes a pathname relative to the current path.	
Package	<code>lispworks</code>	
Signature	<code>current-pathname &optional relative-pathname type -> pathname</code>	
Arguments	<code>relative-pathname</code> A pathname designator.	
	<code>type</code> A string or <code>nil</code> . The default is <code>nil</code> .	

Values	<i>pathname</i>	A pathname.
Description	The function <code>current-pathname</code> is useful for loading other files relative to a file. <code>current-pathname</code> computes a pathname from the current operation as follows:	
	When loading a file	 Uses <code>*load-pathname*</code> .
	When compiling a file	 Uses <code>*compile-file-pathname*</code> .
	When evaluating or compiling an Editor buffer	 Uses the pathname of the buffer, if available, otherwise uses the current working directory.
	Otherwise	 Uses the current working directory.
		The pathname computed above is then translated to a physical pathname, and the argument <i>relative-pathname</i> is merged with this physical pathname. The <code>pathname-type</code> of the result <code>pathname</code> is set to <i>type</i> if supplied, the <code>pathname-version</code> is set to <code>:newest</code> , and <code>pathname</code> is returned.
		A useful value for <i>type</i> is <code>nil</code> , which can be used to allow <code>load</code> to choose between lisp or fasl regardless of the type of the current pathname.
		Note: <code>defsystem</code> uses <code>current-pathname</code> with its <code>:default-host</code> argument.
Examples	Suppose you want the file <code>foo</code> to load the file <code>bar</code> . While loading the source file <code>foo.lisp</code> :	 <code>(current-pathname "bar")</code>

```
=>
#P"C:/temp/bar.lisp"
```

While loading the binary file `foo.ofasl`:

```
(current-pathname "bar")
=>
#P"C:/temp/bar.ofasl"
```

To load `bar.lisp` or `bar.ofasl` according to the value of `*load-fasl-or-lisp-file*`, regardless of whether `foo.lisp` or `foo.ofasl` is being loaded, specify `type nil`:

```
(load (current-pathname "bar" nil))
```

See also

`defsystem`
`pathname-location`

defadvice *Macro*

Summary Defines a new piece of advice.

Package `lispworks`

Signature `defadvice (dspec name advice-type &key where documentation)
lambda-list &body body => nil`

dspec ::= `fn-name` |
`macro-name` |
`(method generic-fn-name [(class*)])`

advice-type ::= `:before` | `:after` | `:around`

Arguments `dspec` Specifies the functional definition to which the piece of advice belongs. There are three forms which this specification may take. The first one above specifies a function by its name; the second one specifies a macro by name; the third specifies a method by the name of its generic function and by a list of classes to specialize the arguments to the

method. In the case of a method the list of classes must correspond exactly to the classes of the specialized parameters of an existing method, and the advice is then attached to this method.

When advice is provided for a macro using `defadvice`, then the function with which the advice is associated is the expansion function for that macro. Thus before and after advice for a macro receive the arguments given to the macro expansion function, which are normally the macro call form and an environment.

<i>name</i>	A symbol naming the piece of advice being created. It should of course be unique to the advised function, but does not need to be globally unique.
<i>advice-type</i>	A keyword specifying the kind of advice wanted.
<i>where</i>	Specifies where this advice should be placed in the ordering of pieces of advice for the function. By default a piece of advice is placed at the start of the corresponding section. If this argument is present and is <code>:end</code> then the advice is instead placed at the end of its section. The other permissible value for this argument is <code>:start</code> , which places the advice at the start of its section in the ordering (as in the default behavior).
<i>documentation</i>	A string providing documentation on the piece of advice.

<i>lambda-list</i>	A lambda list for the piece of advice. In the case of before and after advice this should be compatible with the lambda-list for the original definition, since such advice receives the same arguments as that function.
<i>body</i>	The main body of the advice.
Values	<code>defadvice</code> returns <code>nil</code> .
Description	<p><code>defadvice</code> is the macro used to define a new piece of advice. Advice provides a way to change the behavior of existing functional definitions in the system. In a simple instance advice might be used to carry out some additional actions before or after the original definition. More sophisticated uses allow the definition to be replaced by new code that can access the original function repeatedly or as rarely as desired, and that can receive different numbers of arguments and return any values. A function may have any number of pieces of advice attached to it by using <code>defadvice</code>.</p> <p>There are three kinds of advice that may be defined: before, after and around advice. The first two kinds attach auxiliary code to be carried out alongside the original definition (before it for before advice, after it in the case of after advice). Around advice replaces the function altogether; it may define code that never accesses the original definition, that receives different numbers of arguments, and returns different values. All the pieces of advice for a function are ordered. The ordering is important in determining how all the pieces of advice for a function are combined. Around advice always comes first, then before advice, then the original definition, and lastly the after advice.</p> <p>Conceptually the before advice, the original definition and the after advice are amalgamated into one new construct. If this gets called then each of its components receives the same arguments in turn, and the values returned are those produced by the last piece of after advice to be called in this way</p>

(or the original function if there is no after advice). The code associated with before and after advice should not destructively modify its arguments.

If around advice is present then the first piece of around advice is called, instead of the combination involving before and after advice discussed above. It does not have to access any of the other advice, nor the original definition. Its only link to the rest of the advice is by means of a call to `call-next-advice`. It may invoke this as often as it chooses, and by doing so it accesses the next piece of around advice if present, or else it accesses the combination of before and after advice together with the original definition.

Remove advice using `remove-advice` or `delete-advice`.

Notes `defadvice` is an extension to Common Lisp.

See also `call-next-advice`
 `delete-advice`
 `remove-advice`

default-action-list-sort-time *Variable*

Summary Determines when actions in action lists are sorted.

Package `lispworks`

Signature `*default-action-list-sort-time*`

Initial value `:execute`

Description Contains a keyword that is either `:execute` or `:define-action`, denoting when actions in action-lists are sorted (see `define-action-list` for an explanation of ordering specifiers). Actions are sorted either at time of definition (`:define-`

`action`) or when their action-list is executed (`:execute`). The default sort time is `:execute`.

See also `define-action`

default-character-element-type *Parameter*

Summary Provides defaults for all character type parameters.

Package `lispworks`

Description This variable provides defaults for all character type parameters. The legal values are `base-char`, `lw:simple-char`, and `character`. Its value must only be set via a call to `lw:set-default-character-element-type`.

This is intended mainly for running old 8-bit applications efficiently. If you write for a fat character implementation you should already be aware of these issues, and make some attempt to provide explicit types.

When the compiler does type inferencing it behaves as if this variable was bound to `character`; if you want assumptions about types to be hard-coded into your program, you must supply explicit declarations and type arguments.

See also `string`
`open`
`set-default-character-element-type`
`with-output-to-string`

define-action *Macro*

Summary Adds a new action to a specified list.

Package `lispworks`

Signature	<code>define-action <i>name-or-list</i> <i>action-name</i> <i>data</i> &rest <i>specs</i> =></code>
Arguments	<p><i>name-or-list</i> A list or action list object.</p> <p><i>action-name</i> A general lisp object.</p> <p><i>data</i> An object.</p> <p><i>specs</i> A list.</p>
Description	<p>The <code>define-action</code> macro adds a new action to the specified list; this action will be executed according to the action-list's execution-function (see <code>execute-actions</code>) when executed. If the action-list specified by <i>name-or-list</i> does not exist, then this is handled according to the value of <code>*handle-missing-action-list*</code>.</p> <p><i>name-or-list</i> is evaluated to give either a list UID (to be looked up in the global registry of lists) or an action list object. <i>action-name</i> is a UID (general lisp object, to be compared by <code>equalp</code>). It uniquely identifies this action within its list (as opposed to among all lists).</p> <p><i>data</i> specifies an object referring to data relevant to the action.</p> <p><i>specs</i> is a free-form list of ordering specifiers and extra keywords, used to control more details of how and when this action is executed.</p> <p>Action-items are normally expected not to be redefined. If an action-item with that action-name already exists in the action-list (that is, one with an identifier <code>equalp</code> to the action-name), then the notification and subsequent handling of this attempt is controlled by the values in the list <code>*handle-existing-action-in-action-list*</code>. This is to prevent problems due to re-evaluating an action definition inappropriately. Notification and redefine behavior can be overridden by using the <code>:force</code> keyword argument. In this case, any required redefinition is performed unconditionally and without notification.</p>

The following keywords are recognized in the *specs* argument:

:after	The following element in <i>specs</i> is a UID. : after specifies that the action-item being defined must be run after the action-item named. If there is no action-item with a matching name, the restriction is ignored.
:before	Like :after , but this action-item must be run before the one specified.
:after and :before can be specified as many times as necessary to describe the ordering constraints of this action-item with respect to its neighbors.	
:once	Specifies that this action-item should be executed only once; after execution, it is disabled.
:force	Specifies that this definition should override any previous definition of this action-item, rather than be subject to the value of *handle-existing-action-in-action-list* .

Example `(define-action :network-startup "Reset decnet buffers"
 '(decnet::reset-network-buffers
 net-buffers)
 :after "Reset core network"
 :once))`

See also `undefined-action`

define-action-list *Macro*

Summary Defines a registered action list.

Package `lispworks`

Signature	<code>define-action-list <i>uid</i> &key <i>documentation</i> <i>sort-time</i> <i>dummy-actions</i> <i>default-order</i> <i>execution-function</i> =></code>
Arguments	<p><i>uid</i> A Lisp object.</p> <p><i>documentation</i> A string.</p> <p><i>sort-time</i> One of <code>:execute</code> or <code>:define-action</code>.</p> <p><i>dummy-actions</i> A list.</p> <p><i>default-order</i> A list.</p> <p><i>execution-function</i> A function.</p>
Description	<p>The <code>define-action-list</code> macro defines an action list.</p> <p><i>uid</i> is a unique identifier, and must be a general Lisp object, to be compared by <code>equalp</code>. It names the list in the global registry of lists. See <code>make-unregistered-action-list</code> to create unnamed, “unregistered” action-lists. The <i>uid</i> may be quoted, but is not required to be. It is possible, but not recommended, to define an action-list with unique identifier <code>nil</code>. If a registered action-list with the <i>uid</i> already exists (that is, one which returns <code>t</code> when compared with <code>equalp</code>), then notification and subsequent handling is controlled by the value of the <code>*handle-existing-action-list*</code> variable.</p> <p>The <i>documentation</i> string allows you to provide documentation for the action list.</p> <p><i>sort-time</i> is a keyword specifying when added actions are sorted for the given list — either <code>:execute</code> or <code>:define-action</code> (see <code>*default-action-list-sort-time*</code>).</p> <p><i>dummy-actions</i> is a list of action-names that specify placeholder actions; they cannot be executed and are constrained to the order specified in this list, for example</p> <pre>'(:beginning :middle :end)</pre> <p><i>default-order</i> specifies default ordering constraints for subsequently defined action-items where no explicit ordering constraints are specified. An example is</p>

```
'(:after beginning :before :end)
```

execution-function specifies a user-defined function accepting arguments of the form:

```
(the-action-list other-args-list &rest keyword-value-pairs)
```

where the two required arguments are the action-list and a list of additional arguments passed to `execute-actions`, respectively. The remaining arguments are any number of keyword-value pairs that may be specified in the call to `execute-actions`. If no execution function is specified, then the default execution function will be used to execute the action-list.

See also

- *`default-action-list-sort-time`*
- *`handle-existing-action-list`*
- `undefine-action-list`

defsystem

Macro

Summary

`defsystem` is used to define systems for use with the LispWorks system tools. A system is a collection of files and other systems that, together with rules expressing the interdependencies of those files and subsystems, make a complete program. The LispWorks system tools support the development and maintenance of large programs.

Package

`lispworks`

Signature

```
defsystem system-name options &key members rules => system
```

Arguments

system-name The name of the system to be made.

options are expressed as a list of keyword argument pairs. The following keywords are recognized:

:package	The default package that files are compiled and loaded in. If not specified, this defaults to the value of *package* at macroexpansion time.
:default-pathname	Used to compute a default pathname in which to find files. defsystem uses current-pathname to compute the pathname. defsystem checks that all the files given as members actually exist.
:default-host	The root pathname of a system is defined to be the :default-host if it is given. Otherwise, it is taken to be the directory containing the defsystem file. Absolute pathnames are interpreted literally, and relative pathnames are taken relative to the root pathname.
:default-type	This is the default type of the members of the system. This may be :lisp-file , :lsp-file , :c-file , OR :system . The corba module adds :idl-file , :idl-client-definition , :idl-client-definition-only , :idl-server-definition and :idl-server-definition-only . The com module adds the type :midl-file and the automation module adds :midl-type-library-file . The default is :lisp-file , which means files with file type (extension) "lisp".
:documentation	This is a string.
:object-pathname	

A string or pathname specifying a directory where object files are written.

Note: This option will not work if the names in *members* represent absolute pathnames.

:optimize

A declaration specifying default compilation qualities within the scope of `compile-system`. These settings override the current global setting. They can be overridden per member by the `:optimize` option (for subsystems) or `proclaim` (in files). The `:optimize defsystem` option accepts the same optimize qualities as `proclaim` and which are fully described in the *LispWorks User Guide*. See below for examples.

members is a list defining the members of the system. Each element of the list may be a symbol or a string representing the name of the physical file or system referred to, or a list of format (*name {keyword value}**) where *name* is once again a symbol or a string referring to the system or physical file, and the possible keywords are:**

:type

The type of this member. Allowed values are as for `:default-type`. If not specified it defaults to the value of `:default-type` given as an *option*.

:root-module

If `nil` then this member is not loaded unless its loading is specifically requested as a result of a dependency on another module

:source-only

Only the source file for this member is ever loaded

:load-only

The member is never compiled by `defsystem`, objects are loaded in preference to source files

:load-for-compile-only

The member is only loaded as necessary during compilation and is never loaded independently

:features The member is only considered during planning if the feature expression is true.

:package A default package for the member.

On Windows, the automation module adds the keyword **:com** for a member with type **:midl-type-library-file**. Then a member of the form

```
("mso97.tlb" :type :midl-type-library-file :com nil)
```

can be specified when you use only Automation client code, reducing the memory used.

rules is a list of rules of the following format :

```
({:in-order-to} action {:all | ({ member-name }* )}  
 (:caused-by {(action {:previous |{member-name }* }) }*)  
 (:requires {(action {:previous |{ member-name }* }) }*))
```

The keyword **:all** refers to all the members of the system. It provides a shorthand for specifying that a rule should apply to all the system's members. The keyword **:previous** refers to all the members of the system that are before the member in the list of members. This makes it easy, for example, to specify that in order to compile a file in a system, all the members that come before it must be loaded.

Values The name of the system is returned.

Examples

```
(defsystem defsys-macros  
  (:default-pathname "/usr/users/james/scm/defsys/"  
    :default-type :lisp-file  
    :package defsystem)  
  :members ("new-macros"  
            "scm-timemacros"))
```

```

(defsystem clos-sys
  (:default-pathname "/usr/users/clc/defsys/"
    :default-type :lisp-file
    :package defsystem)
  :members
  (( "defsys-macros" :type :system :root-module nil)
    "class"
    "time-methods"
    ("scm-pathname" :source-only t)
    "execute-plan"
    "file-types"
    "make-system"
    "conv-defsys")
  :rules
  ((:in-order-to :compile ("class" "time-methods")
    (:caused-by (:compile "defsys-macros"))
    (:requires
      (:load "defsys-macros")))
   (:in-order-to :compile
     ("time-methods" "execute-plan")
     (:requires (:load "class")))))

(defsystem dataworks-demo
  (:default-type :system)
  :members (
    "db-class"
    "planar"
    "dataworks-dep"
    "dataworks-interface-tk"
    "dataworks-interface-tools"
    "drugs-demo"
    ("gen-demo" :type :lisp-file)
    ("load-icon" :type :lisp-file :source-only t)
    )
  :rules ((:in-order-to :compile :all
  (:requires (:load :previous)))))


```

This last example illustrates the use of `:optimize`.

```

(defsystem foo (:optimize ((speed 3) (space 3)
                           (safety 0)))
  :members ("bar"
            "baz")
  :rules ((:compile :all
    (:requires (:load :previous)))))


```

Notes	Systems that are members of another system must be declared in the system declaration file before the system of which they are a part. The ordering of members is important and reflects the order in which operations are carried out on the members of the system.
-------	---

See also	<code>load-system</code> <code>compile-system</code> <code>concatenate-system</code> <code>current-pathname</code> <code>*defsystem-verbose*</code>
----------	---

defsystem-verbose *Variable*

Summary	Controls the amount of messages printed by <code>defsystem</code> about system (re)definition.
Package	<code>lispworks</code>
Initial value	
Description	The variable <code>*defsystem-verbose*</code> is a generalized boolean controlling the amount of messages printed by <code>defsystem</code> . When the value is true, the system prints messages about system definition and redefinition. The default value is <code>t</code> .
See also	<code>defsystem</code>

delete-directory *Function*

Summary	Deletes a directory.
Package	<code>lispworks</code>

Signature	<code>delete-directory directory &optional error => result</code>	
Arguments	<code>directory</code>	A pathname designator.
	<code>error</code>	<code>nil</code> , <code>:error</code> or <code>:no-error</code> .
Value	<code>result</code>	<code>t</code> or <code>nil</code> .
Description	<p>The function <code>delete-directory</code> attempts to delete the directory <code>directory</code>. It returns <code>t</code> on success, and on failure either returns <code>nil</code> or signals an error.</p> <p><code>error</code> determines what happens when <code>delete-directory</code> fails. When <code>error</code> is <code>nil</code> (the default), if <code>directory</code> does not exist <code>delete-directory</code> returns <code>nil</code>, otherwise any failure causes an error to be signaled. If <code>error</code> is <code>:no-error</code>, <code>delete-directory</code> returns <code>nil</code> on any failure. If <code>error</code> is <code>:error</code>, any failure causes an error to be signaled.</p> <p>Typical reasons for failures in <code>delete-directory</code> are that <code>directory</code> is not empty, or that the user does not have the right permissions.</p>	

		<i>Function</i>
deliver		
Summary	The main interface to the Delivery tools.	
Package	<code>lispworks</code>	
Signature	<code>deliver function file level &rest keywords</code>	
Description	<p>The function <code>deliver</code> is the main interface to the LispWorks delivery tools. You use it to create LispWorks executable applications and dynamic libraries.</p> <p>For more information about Delivery including a detailed description of <code>deliver</code>, see the <i>LispWorks Delivery User Guide</i>.</p>	

For information about invoking `deliver` using the IDE, see "The Application Builder" in the *Common LispWorks User Guide*.

See also `save-image`

describe-length *Variable*

Summary Determines how many attributes of a composite object are described.

Package `lispworks`

Initial Value 20

Description The variable `*describe-length*` controls how many attributes of a composite object the function `describe` describes.

This means the number of elements of a sequence, entries in a hash table, slots of a structure instance, and so on.

If `*describe-length*` is `nil` then `describe` describes all of the attributes. Use this value only with care.

Note: the `describe` functionality is load-on-demand in the LispWorks image as shipped. Therefore if you have not done `(require "describe")` or called `describe`, `*describe-length*` may be unbound.

See also `describe`

describe-level *Variable*

Summary Controls the depth to which `describe` describes arrays, structures and conses.

Package	<code>lispworks</code>
Initial Value	1
Description	The variable <code>*describe-level*</code> controls the depth to which the function <code>describe</code> describes arrays, structures and conses. Note: the <code>describe</code> functionality is load-on-demand in the LispWorks image as shipped. Therefore if you have not done <code>(require "describe")</code> or called <code>describe</code> , <code>*describe-level*</code> may be unbound.
Example	<pre>CL-USER 23 > (describe 1) [... load output not shown ...] 1 is a BIT DECIMAL 1 HEX 1 OCTAL 1 BINARY 1 CL-USER 24 > *describe-level* 1 CL-USER 25 > (defstruct foo a s d) FOO CL-USER 26 > (defmethod describe-object ((f foo) (s stream)) (format s "FOO ~S~%" f) (describe (foo-a f) s)) #<STANDARD-METHOD DESCRIBE-OBJECT NIL (FOO STREAM) 2068295C> CL-USER 27 > (describe (make-foo :a (vector 1 2 3) :s 42)) FOO #S(FOO A #(1 2 3) S 42 D NIL) #(1 2 3)</pre>

To make `describe` operate on objects inside the structure instance, increase the value of `*describe-level*`:

```
CL-USER 28 > (setf *describe-level* 2)
2

CL-USER 29 > (describe (make-foo :a (vector 1 2 3) :s
42))

FOO #S(FOO A #(1 2 3) S 42 D NIL)
#(1 2 3) is a SIMPLE-VECTOR
 0      1
 1      2
 2      3
```

See also [describe](#)

describe-print-length *Variable*

Summary Specifies a print length for `describe` and `apropos`.

Package [lispworks](#)

Initial Value 10

Description If `*print-length*` is nil, `describe` and `apropos` bind
`*print-length*` to the value of `*describe-print-length*`.

See also [describe](#)

describe-print-level *Variable*

Summary Specifies a print level for `describe` and `apropos`.

Package [lispworks](#)

Initial Value 10

Description If `*print-level*` is nil, `describe` and `apropos` bind
`*print-level*` to the value of `*describe-print-level*`.

See also **describe**

dll-quit	<i>Function</i>
Summary	Makes a LispWorks dynamic library quit.
Package	lispworks
Signature	dll-quit &key kill-all-processes timeout output force => result, quit-output
Arguments	<p>kill-all-processes A generalized boolean.</p> <p>timeout A positive integer or nil.</p> <p>output An output stream designator.</p> <p>force A generalized boolean.</p>
Values	<p>result t or nil.</p> <p>quit-output A string or nil.</p>
Description	<p>The function dll-quit makes a LispWorks dynamic library (or DLL) quit on returning from the callback in which it was called. It must be called only:</p> <ul style="list-style-type: none"> • In an image running as a dynamic library, meaning an image created by save-image with :dll-exports or by deliver with :dll-exports, and • Inside the dynamic scope of a callback into the dynamic library. That is, not in a process that was started by process-run-function. <p>dll-quit sets up the internal state such that just before returning into its caller in the LispWorks dynamic library it causes LispWorks to quit. After quitting the callback returns as normal. The library can be unloaded using FreeLibrary, or you can re-use it (without re-loading).</p>

By default `kill-all-processes` is `nil` which means that, if there are other running processes, `d11-quit` just returns `nil`. If `kill-all-processes` is non-`nil`, `d11-quit` tries to kill all the other processes, and if it succeeds, it quits.

If `kill-all-processes` is true, `timeout` is a maximum time to wait after killing the other processes. It allows `timeout` seconds for all processes to die.

`d11-quit` should be called when no other processes are running, whether they were created by a callback or by `process-run-function`. If such processes exist, by default `d11-quit` does nothing and returns `nil`. If `force` is non-`nil`, `d11-quit` always tries to set LispWorks up for quitting. LispWorks will quit even after a failure to kill all other processes and complete any required shut down operations. A true value of `force` automatically implies `kill-all-processes` true. However, if any of the other processes is stuck in a foreign call, the quitting may fail to finish properly. The default value of `force` is `nil`.

If `output` is supplied, `d11-quit` generates output if it is called when other processes are still running, or a required shut down operation was not completed. `output` can be an output stream, `t` (interpreted as `*standard-output*`) or `nil`. If `output` is `nil`, `d11-quit` collects the output and returns it as second argument `quit-output`. Otherwise it writes the output to the stream and `quit-output` is `nil`.

The output contains a list of the other processes that are still running. If `kill-all-processes` or `force` was supplied, and killing the other processes failed, the output also contains backtraces of the other processes, and possibly other debugging information.

`result` is `t` on success: the LispWorks dynamic library is set to quit on returning from the callback. `result` is `nil` when other processes are running: the image is not set to quit.

quit-output contains the output which was generated when *output nil* was passed. Otherwise *quit-output* is *nil*.

If `d11-quit` is called inside a recursive foreign callback, the LispWorks dynamic library quits only when the outermost callback returns.

Note: `d11-quit` is intended for use when a LispWorks dynamic library is loaded by a main process which you (the LispWorks programmer) do not control. If you control the main process, then use `QuitLispWorks` instead.

It is expected that the main process will call into the dynamic library with some "shutdown" call, and then calls `FreeLibrary` to free the library. The shutdown call should close and free everything that needs to be closed or freed, call `d11-quit`, and return.

Note: `d11-quit` is supported only where LispWorks can be a dynamic library. Currently this is in 32-bit LispWorks on Microsoft Windows, Intel Macintosh, Linux and FreeBSD, and in 64-bit LispWorks on Windows, Intel Macintosh and Linux.

See also `deliver`
`save-image`

		<i>Function</i>
dotted-list-length		
Summary	Similar to <code>list-length</code>	
Package	<code>lispworks</code>	
Signature	<code>dotted-list-length list => result</code>	
Arguments	<code>list</code>	A list.
Value	<code>result</code>	An integer.

Description	The function <code>dotted-list-length</code> performs the same action as <code>list-length</code> , except that if the last <code>cdr</code> is not <code>nil</code> then instead of signalling an error, it returns the number of <code>conses</code> plus 1.
See also	<code>dotted-list-p</code>

dotted-list-p *Function*

Summary Tests whether a `cons` is a list ending in a non-`nil` `cdr`.

Package `lispworks`

Signature `dotted-list-p list => bool`

Arguments `list` A list, which must be a `cons`.

Values `bool` A generalized boolean.

Description The function `dotted-list-p` is a predicate which tests whether `list` (which must be a `cons`) is a list ending in a non-`nil` `cdr`. It returns a true value if this is the case, otherwise it returns `nil`.

See also `dotted-list-length`

do-nothing *Function*

Summary Ignores its arguments and returns an unspecified value.

Package `lispworks`

Signature `do-nothing &rest ignore => unspecified`

Arguments `ignore` All arguments are ignored.

Values	<i>unspecified</i>	An unspecified value.
Description	The function <code>do-nothing</code> ignores its arguments and returns an unspecified value. It is useful as a function argument.	
See also	<code>false</code> <code>true</code>	

enter-debugger-directly *Variable*

Summary	Controls direct entry into the Debugger tool.
Package	<code>lispworks</code>
Initial value	<code>nil</code>
Description	<p>The variable <code>*enter-debugger-directly*</code> is a generalised boolean which affects the behavior of the LispWorks IDE when an error is signalled outside of the Listener REPL.</p> <p>Value <code>nil</code> causes an error notifier window to be displayed (from which you can abort, report a bug, or raise a Debugger tool).</p> <p>A true value causes the Debugger tool to be displayed immediately, and no error notifier appears.</p> <p>Note: Errors signalled in a Listener Read-Eval-Print loop are handled in the REPL and therefore <code>*enter-debugger-directly*</code> has no effect on the behavior in this case.</p>

environment-variable *Function*

Summary	Reads the value of an environment variable from the environment table of the calling process.
---------	---

Package	<code>lispworks</code>	
Signature	<code>environment-variable name => result</code>	
Arguments	<code>name</code>	A string.
Values	<code>result</code>	A string or <code>nil</code> .
Description	<p>The function <code>environment-variable</code> reads the environment variable specified by <code>name</code> and returns its value, or <code>nil</code> if the variable could not be found.</p> <p>A <code>setf</code> method is also defined, allowing you to set the value of an environment variable:</p>	

```
(setf (environment-variable name) value)
```

If `value` is a string, then `name` is set to be `value`. If `value` is `nil` then `name` is removed from the environment table.

Example	In this first example the value of the environment variable <code>PATH</code> is returned:
	<pre>(environment-variable "PATH")</pre>

The result is a string of all the defined paths:

```
"c:\\hqbin\\nt\\x86;c:\\hqbin\\nt\\x86\\perl;c:\\hqbin\\\nwin32;c:\\usr\\local\\bin;C:\\WINNT35\\system32;C:\\WINNT35;;C:\\MSTOOLS\\bin;C:\\TGS3D\\PROGRAM;c:\\program files\\devstudio\\sharedide\\bin\\ide;c:\\program files\\devstudio\\sharedide\\bin;c:\\program files\\devstudio\\vc\\bin;c:\\msdev\\bin;C:\\WINDOWS;C:\\WINDOWS\\COMMAND;C:\\WIN95\\COMMAND;C:\\MSINPUT\\MOUSE"
```

In the second example, the variable `MYTZONE` is found not to be in the environment table:

```
(environment-variable "MYTZONE")
```

```
NIL
```

It is set to be `GMT` using the `setf` method:

```
(setf (environment-variable "MYTZONE") "GMT")
```

errno-value	<i>Function</i>
--------------------	-----------------

errno-value	<i>Function</i>	
Summary	Returns the current value of the UNIX variable <code>errno</code> .	
Package	<code>lispworks</code>	
Signature	<code>errno-value => value</code>	
Arguments	None.	
Values	<code>value</code>	The current value of <code>errno</code> .
Description	The function <code>errno-value</code> returns the current value of the UNIX variable <code>errno</code> . Note: this is implemented only on UNIX/Linux/Mac OS X.	
Example	<pre>USER 10 > (errno-value) 2 USER 11 > (get-unix-error 2) "no such file or directory"</pre>	
See also	<code>get-unix-error</code>	

example-file	<i>Function</i>
---------------------	-----------------

example-file	<i>Function</i>
Summary	Returns a path in the <code>examples</code> folder.
Package	<code>lispworks</code>
Signature	<code>example-file file => path</code>

Arguments	<i>file</i>	A pathname designator.
Values	<i>path</i>	A pathname.
Description		The function <code>example-file</code> returns an absolute path to a file <i>file</i> in the <code>examples</code> folder of the LispWorks library. It does not actually test for the existence of the file.
Example		<pre>(example-file "capi/applications/othello.lisp") => #P"C:/Program Files/LispWorks/lib/5-1-0- 0/examples/capi/applications/othello.lisp"</pre>
See also		<code>example-compile-file</code>

example-compile-file		<i>Function</i>
Summary		Compiles a file in the <code>examples</code> folder to a temporary output file.
Package		<code>lispworks</code>
Signature		<code>example-compile-file file &rest args => output-truename, warnings-p, failure-p</code>
Arguments	<i>file</i>	A pathname designator.
	<i>args</i>	Arguments passed to <code>compile-file</code> .
Values	<i>output-truename</i>	A pathname or <code>nil</code> .
	<i>warnings-p</i>	A generalized boolean.
	<i>failure-p</i>	A generalized boolean.
Description		The function <code>example-compile-file</code> constructs the path to <i>file</i> in the <code>examples</code> folder of the LispWorks library, and a

path to an output file in a temporary location which is likely to be writable.

It then calls `compile-file` with these paths as the *input-file* and *output-file*, also passing the other *args*, and returns the values returned by `compile-file`.

See also `example-file`

example-load-binary-file *Function*

Summary Loads a fasl file compiled by `example-compile-file`.

Package `lispworks`

Signature `example-load-binary-file file => generalized-boolean`

Arguments `file` A pathname designator.

Values *generalized-boolean*

The value returned by `load`.

Description The function `example-load-binary-file` constructs the path to an output file in a temporary location which would be used as the *output-file* by `example-compile-file`.

It then calls `load` on that path, and returns the values returned by `load`.

See also `example-compile-file`

execute-actions *Macro*

Summary Executes in sequence the actions on a given list.

Package `lispworks`

Signature	<code>execute-actions name-or-list &rest keyword-value-pairs &rest other-args =></code>
Arguments	<p><i>name-or-list</i> An action list</p> <p><i>keyword-value-pairs</i> See description.</p> <p><i>other-args</i> A list.</p>
Description	<p>The <code>execute-actions</code> macro executes, in sequence, the actions on the specified list. If the action-list specified by <i>name-or-list</i> does not exist, then this is handled according to the value of <code>*handle-missing-action-list*</code>. Note that <i>name-or-list</i> is evaluated.</p> <p>If a user-defined execution function was specified when the action list was defined, then it should accept the following arguments:</p> <p><code>(action-list other-args &rest keyword-value-pairs)</code></p> <p>Note that <i>other-args</i> is passed as a single list.</p> <p>If a user-defined execution function was not specified when the action list was defined, then the following default mapping occurs. Each action's data is invoked via <code>apply</code> on <i>other-args</i>:</p> <p><code>(apply data other-args)</code></p> <p>This behavior is modified by the keyword-value-pairs, thus:</p> <ul style="list-style-type: none"> • If the keyword parameter <code>:ignore-errors-p</code> is non-<code>nil</code>, any otherwise-unhandled errors signalled inside the execution of that function will be trapped, and a warning issued. Execution continues with the next action-item. If <code>:ignore-errors-p</code> is <code>nil</code> (or not specified), then the error is not trapped.

- If the keyword parameter `:post-process` is non-`nil`, the first value returned by each action is handled, according to `:post-process`, thus:

<code>:collect</code>	collect values into list
<code>:and</code>	return <code>t</code> only if all values are <code>t</code> . Return <code>nil</code> immediately if any value is <code>nil</code>
<code>:or</code>	return first non- <code>nil</code> value

See also `define-action`
 `with-action-list-mapping`

extended-char *Type*

Summary	The extended character type.
Package	<code>lispworks</code>
Signature	<code>extended-char</code>
Description	The type of extended characters. A synonym for <code>extended-character</code> , but with standard spelling.

extended-character *Type*

Summary	The extended character type.
Package	<code>lispworks</code>
Signature	<code>extended-character</code>
Description	The type of extended characters.

extended-character-p *Function*

Summary	Tests if an object is an extended character.	
Package	lispworks	
Signature	extended-character-p <i>object</i> => <i>bool</i>	
Arguments	<i>object</i>	The object to be tested.
Values	<i>bool</i>	t if <i>object</i> is an extended character; nil otherwise.
Description	This is the predicate for extended characters.	
See also	extended-character	

extended-char-p *Function*

Summary	Tests if an object is an extended character.	
Package	lispworks	
Signature	extended-char-p <i>object</i> => <i>bool</i>	
Arguments	<i>object</i>	The object to be tested.
Values	<i>bool</i>	t if <i>object</i> is an extended character; nil otherwise.
Description	This is also the predicate for extended characters, only with standard spelling.	
See also	extended-char extended-character-p	

external-formats		<i>Variable</i>
Summary	A list of the names of the defined external formats.	
Package	<code>lispworks</code>	
Initial value	Microsoft Windows platforms: <pre>(WIN32:CODE-PAGE FLI::LATIN-1-WCHAR FLI:ASCII-WCHAR :MACOS-ROMAN :UTF-8 :GBK :WINDOWS-CP936 EXTERNAL- FORMAT:DOUBLE-BYTE-TABLE-LOOKUP :JIS :EUC-JP :SJIS :LATIN-1-TERMINAL :UNICODE :LATIN-1-SAFE :LATIN-1- CHECKED :EUC :SHIFT-JIS :NIHONGO-MS :NIHONGO-EUC :NIHONGO-JIS CHARACTER EXTERNAL-FORMAT::BYTE-SWAPPED- SIMPLE-CHARACTER EXTERNAL-FORMAT::RAW-SIMPLE-CHARACTER EXTERNAL-FORMAT::RAW-BASE-CHARACTER :ASCII-TERMINAL :ASCII :LATIN-1)</pre>	
	On all other platforms: <pre>(FLI::LATIN-1-WCHAR FLI:ASCII-WCHAR :MACOS-ROMAN :UTF-8 :GBK :WINDOWS-CP936 EXTERNAL-FORMAT:DOUBLE-BYTE-TABLE- LOOKUP :JIS :EUC-JP :SJIS :LATIN-1-TERMINAL :UNICODE :LATIN-1-SAFE :LATIN-1-CHECKED :EUC :SHIFT-JIS :NIHONGO-MS :NIHONGO-EUC :NIHONGO-JIS EXTERNAL- FORMAT::HOST-PORTABLE EXTERNAL-FORMAT::LATIN-PORTABLE CHARACTER EXTERNAL-FORMAT::BYTE-SWAPPED-SIMPLE- CHARACTER EXTERNAL-FORMAT::RAW-SIMPLE-CHARACTER EXTERNAL-FORMAT::RAW-BASE-CHARACTER :ASCII-TERMINAL :ASCII :LATIN-1)</pre>	
Description	The variable <code>*external-formats*</code> contains a list of the names of the defined external formats. The platform-specific external format names are: <code>code-page</code> <code>latin-portable</code>	
		Uses the encoding in the Microsoft Windows code page specified by the <code>:id</code> parameter.
		Intended for use when communicating with X servers, for example when passing XLFID names. Uses the X Portable Character Set.

host-portable

A synonym for **latin-portable**.

false*Function*

Summary	Ignores its arguments and returns nil .	
Package	lispworks	
Signature	false &rest ignore -> nil	
Arguments	<i>ignore</i>	All arguments are ignored.
Value	nil	
Description	The function false takes any number of arguments, which it ignores, and returns nil . It is useful as a functional argument.	
See also	do-nothing true	

file-directory-p*Function*

Summary	Tests for the presence of a directory.	
Package	lispworks	
Signature	file-directory-p pathname => bool	
Arguments	<i>pathname</i>	A pathname, string, or file-stream.
Values	<i>bool</i>	If t , the pathname represented by pathname exists and is a directory. If nil , it either does not exist, or it is not a directory.

Description	<code>file-directory-p</code> tests whether the pathname represents a directory.
Examples	<pre>CL-USER 70 > (file-directory-p "~") T CL-USER 71 > (file-directory-p ".login") NIL</pre>

find-regexp-in-string *Function*

Summary	Matches a regular expression.	
Package	<code>lispworks</code>	
Signature	<code>find-regexp-in-string pattern string &key start end from-end case-sensitive => pos, len</code>	
Arguments	<i>pattern</i>	A string or a precompiled regular expression object.
	<i>string</i>	A string.
	<i>start, end</i>	Bounding index designators of <i>string</i> .
	<i>from-end</i>	A generalized boolean.
	<i>case-sensitive</i>	A generalized boolean.
Values	<i>pos</i>	A non-negative integer or <code>nil</code> .
	<i>len</i>	A non-negative integer or <code>nil</code> .
Description	<p>The function <code>find-regexp-in-string</code> searches the string <i>string</i> for a match for the regular expression <i>pattern</i>. The index in <i>string</i> of the start of the first match is returned in <i>pos</i>, and the length of the match is <i>len</i>.</p> <p>If <i>from-end</i> is <code>nil</code> (the default value) then the search starts at index <i>start</i> and ends at index <i>end</i>. <i>start</i> defaults to 0 and <i>end</i></p>	

defaults to `nil`. If `from-end` is true, then the search direction is reversed.

`pattern` should be a precompiled regular expression object or a string. If `pattern` is a string then `find-regexp-in-string` first makes a precompiled regular expression object. This operation allocates, therefore if you need to repeatedly call `find-regexp-in-string` with the same pattern, it is better to call `precompile-regexp` once and pass its result, a precompiled regular expression object, as `pattern`.

`case-sensitive` controls whether a string `pattern` is precompiled as a case sensitive or case insensitive search. A true value other than `:default` means a case sensitive search. The value `nil` means a case insensitive search. The default value of `case-sensitive` is `:default` which means that a string `pattern` is compiled with case sensitivity according to the value of the Editor variable `DEFAULT-SEARCH-KIND`.

The regular expression syntax used by `find-regexp-in-string` is similar to that used by Emacs, as described in the "Regular expression syntax" section of the *LispWorks Editor User Guide*.

Example

This form allocates several regular expression objects:

```
(loop with pos = 0
      with len = 0
      while pos
      do (multiple-value-setq (pos len)
                             (find-regexp-in-string "[0,2,4,6,8]"
                                     "0123456789"
                                         :start (+ pos len)))
         when pos
         do (format t "~&Match at pos ~D len ~D~%" pos len))
```

This form does the same matching but allocates just one pre-compiled regular expression object:

```
(loop with pattern = (precompile-regexp "[0,2,4,6,8]")
      with pos = 0
      with len = 0
      while pos
        do (multiple-value-setq (pos len)
                               (find-regexp-in-string pattern "0123456789"
                                                     :start (+ pos len)))
      when pos do (format t "~&Match at pos ~D len ~D~%" pos len))
```

See also [precompile-regexp](#)
[regexp-find-symbols](#)

function-lambda-list *Function*

Summary	Returns the argument list of the given function.	
Package	lispworks	
Signature	<code>function-lambda-list <i>function</i> &optional <i>error-p</i> => <i>args</i></code>	
Arguments	<i>function</i>	A symbol or a function.
	<i>error-p</i>	A boolean.
Values	<i>args</i>	A list, or the symbol :none
Description	<p><i>function</i> is the function whose arguments are required</p> <p>If <i>error-p</i> is nil, then <code>function-lambda-list</code> returns :none if <i>function</i> is not defined, and does not start the debugger. The default value of <i>error-p</i> is t, meaning that an error is signalled if <i>function</i> is undefined.</p>	
Example	<pre>TEST 2 > (function-lambda-list 'editor:create-buffer-command) (EDITOR::P &OPTIONAL EDITOR:BUFFER-NAME)</pre>	

get-inspector-values

Generic Function

Summary	Customizes the information displayed in the Common Lisp-Works inspector tool.	
Package	<code>lispworks</code>	
Signature	<code>get-inspector-values object mode</code>	
Arguments	<p><i>object</i> The object to be inspected. <i>mode</i> Name of a mode, or <code>nil</code>. <code>nil</code> defines the default inspection format for <i>object</i>.</p>	
Values	Returns five values: <i>names</i> , <i>values</i> , <i>getter</i> , <i>setter</i> and <i>type</i> . <i>names</i> and <i>values</i> are the two lists displayed in columns in the inspector window. <i>getter</i> is ignored. <i>setter</i> is a function used to update slot values. <i>type</i> is displayed at the foot of the inspector window.	
Description	<p>This generic function allows you to customize the Common LispWorks Inspector by adding new formats (corresponding to different values of mode) in which instances of a particular class can be inspected. Mode <code>nil</code> is the default mode, which is always present (it can be overwritten).</p> <p>LispWorks includes methods for:</p> <pre>(get-inspector-values (object nil)) (get-inspector-values (standard-object nil)) (get-inspector-values (structured-object nil)) (get-inspector-values (sequence nil)) (get-inspector-values cons nil))</pre> <p>and so on.</p>	
Example	This example allows inspection of a CLOS object, displaying only direct slots from a chosen class in its class precedence list. This can be useful when an object inherits many slots from superclasses, and the inherited slots are of no interest.	

```
(defmethod lispworks:get-inspector-values
  ((object standard-object)
   (mode (eql 'direct-as)))
  (declare (ignore mode))
  (loop with object-class =
        (class-of object)
        with precedence-list =
        (class-precedence-list object-class)
        with items =
        (loop for super in precedence-list
              collecting (list*
                           (format nil "~a"
                                   (class-name super))
                           super)))
        with class =
        (or (capi:prompt-with-list items
                                    "Direct slots as ...")
            object-class)
            ;; default if no selection
        with slots =
        (class-direct-slots class)
        for slot in slots
        for name =
        (clos::slot-definition-name slot)
        collect name into names
        collect (if (slot-boundp object name)
                   (slot-value object name)
                   :slot-unbound)
        into values
        finally
        (return
         (values
          names
          values
          nil
          #'(lambda
              (x slot-name index new-value)
              (declare (ignore index))
              (setf (slot-value x slot-name)
                    new-value))
          (format nil "~a - direct slots as ~a"
                  (class-name object-class)
                  (class-name class))))))
```

	<i>Function</i>
Summary	Returns the text associated with a given error.
Package	<code>lispworks</code>
Signature	<code>get-unix-error number => error</code>
Arguments	<code>number</code> The <code>errno</code> value whose text is required.
Values	<code>error</code> The text associated with the error.
Description	The <code>get-unix-error</code> function returns the text associated with the specified value of the UNIX variable <code>errno</code> . Note: this is implemented only on UNIX/Linux/Mac OS X/FreeBSD.
See also	<code>errno-value</code>

	<i>Variable</i>
Package	<code>lispworks</code>
Summary	Determines the search utility used by Grep searches in the Search Files tool in the LispWorks IDE.
Initial Value	<code>"grep"</code> on Unix/Linux/Mac OS X/FreeBSD platforms. <code>nil</code> on Windows.
Description	If the value is a string, it is the search utility to run in the Search Files tool. If the value is <code>nil</code> , then the value of <code>(sys:lispworks-file "etc/grep")</code>

is expected to be an executable, which is run. On Windows a suitable `grep.exe` is included with LispWorks in this location.

The search utility is passed arguments constructed using `*grep-command-format*` and `*grep-fixed-args*`.

See the *Common LispWorks User Guide* for more information about the Search Files tool.

See also `*grep-command-format*`
`*grep-fixed-args*`

grep-command-format *Variable*

Package `lispworks`

Summary The format string used to construct the arguments passed to the Search Files tool to perform a **Grep** search.

Initial Value `"cd '~a'; ~a ~a ~a /dev/null"` on Unix/Linux/Mac OS X.
`"~a ~a ~a NUL"` on Windows.

Description On Unix/Linux/Mac OS X the first format argument is the current directory.

The remainder of the format arguments are:

- the value of `*grep-command*` or, if this is `nil`, the value of `(sys:lispworks-file "etc/grep")`.
- the value of `*grep-fixed-args*`.
- the arguments you specify.

See the *Common LispWorks User Guide* for more information about the Search Files tool.

See also `*grep-command*`
`*grep-fixed-args*`

grep-fixed-args *Variable*

Package	<code>lispworks</code>
Summary	Arguments added to the command string of a Grep search in the Search Files tool.
Initial Value	<code>"-n"</code>
Description	The variable <code>*grep-fixed-args*</code> provides arguments added to a Grep command string in the Search Files tool. The value should ensure that the line number is output at the start of each match. See the <i>Common LispWorks User Guide</i> for more information about the Search Files tool.
See also	<code>*grep-command*</code> <code>*grep-command-format*</code>

handle-existing-action-in-action-list *Variable*

Summary	Contains keywords determining behavior on exceptions raised when an action definition already exists in a given action list.
Package	<code>lispworks</code>
Initial value	<code>(:warn :redefine)</code>
Description	A list containing one of <code>:warn</code> , or <code>:silent</code> , determining whether to notify the user, and one of <code>:skip</code> , or <code>:redefine</code> , to determine what to do about an action definition when the action already exists in the given action list. It is used by <code>define-action</code> .

See also [define-action](#)

handle-existing-action-list *Variable*

Summary Contains keywords determining what to do about a given action list operation when the action list already exists.

Package [lispworks](#)

Initial value `(:warn :skip)`

Description A list containing either `:warn` or `:silent`, determining whether to notify the user, and either `:skip` or `:redefine` to determine what to do about an action list operation when the action list already exists. The initial value is `(:warn :skip)`.

It is used by the `define-action-list` macro.

See also [define-action-list](#)

handle-missing-action-list *Variable*

Summary Defines how to handle an operation on a missing action list.

Package [lispworks](#)

Signature `*handle-missing-action-list*`

Initial value `:error`

Description A keyword; one of `:warn`, `:error`, or `:ignore`, denoting how to handle an operation on a missing action-list. The default value is `:error`. It is used by `undefine-action-list`, `print-actions`, `execute-actions`, `define-action` and `undefine-action`.

See also [define-action](#)
[execute-actions](#)
[print-actions](#)
[undefine-action](#)
[undefine-action-list](#)

handle-missing-action-in-action-list

Variable

Summary Denotes how to handle an operation on a missing action.

Package **lispworks**

Initial value : warn

Description A keyword; one of `:warn`, `:error` or `:ignore`, denoting how to handle an operation on a missing action. Its initial value is `:warn`. It is used by `undefined-action`.

See also [undefined-action](#)

handle-warn-on-redefinition

Variable

Summary Specifies the action on defining a symbol in certain packages.

Package **lispworks**

Initial value : error

Description	* handle-warn-on-redefinition * specifies what action should be taken on defining external symbols in the packages specified in the variable * packages-for-warn-on-definition *. If * handle-warn-on-redefinition * is set to : warn then you are warned. If it is set to : quiet or nil , the definition is done quietly. If, however, it is set to : error , then Lisp-Works signals an error.
-------------	--

See also ***packages-for-warn-on-redefinition***
redefinition-action

hardcopy-system

Function

Summary	Print each file of a system to a printer.
Package	lispworks
Signature	hardcopy-system system-name &key command simulate => nil
Arguments	<p>system-name A symbol representing the name of the system. The system must have been defined using the defsystem macro.</p> <p>simulate If nil or not present then hardcopy-system works silently. Otherwise a plan of the actions which hardcopy-system intends to carry out is printed. What happens next depends on the value of <i>simulate</i>:</p> <ul style="list-style-type: none"> t — do nothing. :ask — you are asked, using y-or-n-p, if you want the plan to be carried out. :each — hardcopy-system displays each action in the plan one at a time, and asks you if you want to carry out this particular action. The answer executes the rest of the plan without further prompting, e returns from hardcopy-system without further processing, and y and n work as expected.
Values	hardcopy-system returns nil .
Examples	<pre>(hardcopy-system 'blackboard) (hardcopy-system 'tms :simulate :ask :command "lpr")</pre>

Notes By default, `hardcopy-system` uses `*print-command*` as the command sent to the shell.

See also `defsystem`
`*print-command*`

init-file-name *Variable*

Summary The default user initialization file.

Package `lispworks`

Initial value `"~/.lispworks"`

Description The variable `*init-file-name*` is the name of the default user initialization file.

However, if the user initialization file is specified by either:

- the command line argument `-init`, or
- user preferences (as set via the Global Preferences dialog in the Common LispWorks IDE)

then the value of `*init-file-name*` is not used.

inspect-through-gui *Variable*

Summary Controls what `inspect` does in the development environment.

Package `lispworks`

Initial Value `nil`

Description The variable `*inspect-through-gui*` controls what `inspect` does in the development environment.

When the value is `nil`, `inspect` uses a command line interface in the REPL.

When the value is true, `inspect` invokes an Inspector tool in the Common LispWorks IDE.

<code>lisp-image-name</code>	<i>Function</i>
Summary	Returns the name of the running image.
Package	<code>lispworks</code>
Signature	<code>lisp-image-name => name</code>
Arguments	None.
Values	<code>name</code> A string.
Description	The function <code>lisp-image-name</code> returns a string representing the full path to the running LispWorks image. The example below is in typical LispWorks for Windows and LispWorks for Linux installations. In resaved and delivered images (including dynamic libraries such as Windows DLLs), the appropriate path is returned.
Example	On Windows: <pre>CL-USER 1 > (lisp-image-name) "C:\\Program Files\\LispWorks\\lispworks-5-1-0-x86- win32.exe"</pre> On Linux: <pre>CL-USER 1 > (lisp-image-name) "/usr/bin/lispworks-5-1-0-x86-linux"</pre>
See also	<code>*line-arguments-list*</code>

lispworks-directory *Variable*

Summary	The main LispWorks installation directory.
Package	<code>lispworks</code>
Initial value	<p>The initial value is</p> <p><code>#P"/usr/lib/lispworks/"</code> on Unix.</p> <p><code>#P"/usr/local/lib/LispWorks/"</code> on Linux (for an installation from the tar archive) or FreeBSD.</p> <p><code>#P"C:\Program Files\LispWorks\"</code> on Microsoft Windows.</p> <p><code>#P"/Applications/LispWorks 5.1/Library/"</code> on Mac OS X.</p> <p>Note however that the value can be set when configuring an image or on startup.</p>
Description	<p>The variable <code>*lispworks-directory*</code> holds the name of the directory where various files important for the running of LispWorks are located.</p> <p>When LispWorks starts in a directory which contains an appropriate numbered subdirectory such as <code>lib/5-1-0-0/</code>, then it assumes this is the LispWorks installation directory and sets <code>*lispworks-directory*</code> accordingly. Additionally, LispWorks for Macintosh running on Cocoa looks for such a subdirectory in the <code>Library</code> folder alongside its application bundle, and if found it sets <code>*lispworks-directory*</code> accordingly.</p> <p>On non-Windows platforms, LispWorks then consults the Unix environment variable <code>LISPWORKS_DIRECTORY</code>. If this is set, then <code>*lispworks-directory*</code> is set accordingly.</p> <p>The <code>lib/5-1-0-0/</code> subdirectory of <code>*lispworks-directory*</code> should include these subdirectories:</p> <p><code>config</code>, which contains the configuration files.</p>

patches, which contains any public (numbered) patches that are distributed by LispWorks Ltd.

private-patches, which is the place to put private (named) patches that are sent to you by Lisp Support.

postscript, which contains configuration files for printing using the CAPI printing library. See "Customization of LispWorks" in the *LispWorks User Guide* for more information on printer configuration.

examples, which contains various files of example code.

Other directories are **etc**, **load-on-demand** and **manual**. There is also **app-defaults** for platforms where Motif is supported.

	<i>Function</i>
load-all-patches	
Summary	Loads all patch files into the image.
Package	lispworks
Signature	load-all-patches => nil
Arguments	None.
Values	Returns nil .
Description	Loads into the image all appropriate files from the directory patches in the directory determined by *lispworks-directory* , and then loads the file private-patches/load.lisp where load forms for any private patches may be placed. When the appropriate patches have successfully been loaded, the updated version of the image can be saved using save-image .

You should call `load-all-patches` before starting the Common LispWorks environment. Thus, you normally place the call to this function in your `.lispworks` file.

The system expects all patches to be loaded sequentially. If a patch is missing, there is a warning message. In this situation, it is advisable to contact Lisp Support to obtain a copy of the missing patch.

load-system	<i>Function</i>
Summary	Load each file of a system into the Lisp image if either the file has not been loaded, or the file has been written since it was last loaded.
Package	<code>lispworks</code>
Signature	<code>load-system <i>system-name</i> &key <i>force</i> <i>simulate</i> <i>source-only</i> <i>target-directory</i> => nil</code>
Arguments	<p><code><i>system-name</i></code> A symbol representing the name of the system. The system must have been defined using the <code>defsystem</code> macro.</p> <p><code><i>force</i></code> If <code>t</code> then all the files in the system are loaded regardless. (This argument was formerly called <code>force-p</code>. The old name is currently still accepted for compatibility.)</p> <p><code><i>simulate</i></code> If <code>nil</code> or not present then <code>load-system</code> works silently. Otherwise a plan of the actions which <code>load-system</code> intends to carry out is printed. What happens next depends on the value of <code>simulate</code>:</p> <ul style="list-style-type: none"><code>t</code> — do nothing.<code>:ask</code> — you are asked, using <code>y-or-n-p</code>, if you want to carry out the plan.

:each — `load-system` displays each action in the plan one at a time, and asks you if you want to carry out this particular action. The answer executes the rest of the plan without further prompting, `e` returns from `load-system` without further processing, and `y` and `n` work as expected.

source-only If `t` the source files of the system are loaded. This only applies to file types where it makes sense to load a source file.

target-directory This is the directory to search for the object files. If the object file cannot be found here then the source file from the system's default directory are loaded.

Values `load-system` returns nil.

Examples

```
(load-system 'blackboard)

(load-system 'tms :simulate :ask :source-only t)
```

Notes For Lisp files `load-system` loads the object file (if it exists) into the image, unless over-ridden by the `:source-only` keyword argument. This behavior can be changed so that the newest file (whether source or object) is loaded by setting the variable `*load-source-if-newer*` to `t`.

C source files, for example `foo.c`, can be included in a system (see the use of `:default-type` and `:type` in `defsystem`). The corresponding object file name is `foo.so` on Linux, and on Unix it is `foon.o` where `n` is a platform-specific integer. On Windows the object file name is `foo.dll`.

See also

- `defsystem`
- `compile-system`
- `concatenate-system`

make-unregistered-action-list *Function*

Summary	Makes an unregistered action list.
Package	<code>lispworks</code>
Signature	<code>make-unregistered-action-list &key documentation sort-time dummy-actions default-order execution-function =></code>
Arguments	<p><i>documentation</i> A string.</p> <p><i>sort-time</i> One of <code>:execute</code> or <code>:define-action</code>.</p> <p><i>dummy-actions</i> A list.</p> <p><i>default-order</i> A list.</p> <p><i>execution-function</i>A function.</p>
Description	<p>Return an action-list not registered in the global registry of lists. The keyword arguments are as for <code>define-action-list</code>.</p> <p>The <i>documentation</i> string allows you to provide documentation for the action list.</p> <p><i>sort-time</i> is a keyword specifying when added actions are sorted for the given list — either <code>:execute</code> or <code>:define-action</code> (see <code>*default-action-list-sort-time*</code>).</p> <p><i>dummy-actions</i> is a list of action-names that specify placeholder actions; they cannot be executed and are constrained to the order specified in this list, for example</p> <pre>'(:beginning :middle :end)</pre> <p><i>default-order</i> specifies default ordering constraints for subsequently defined action-items where no explicit ordering constraints are specified. An example is</p> <pre>'(:after :beginning :before :end)</pre> <p><i>execution-function</i> specifies a user-defined function accepting arguments of the form:</p>

(*the-action-list other-args-list &rest keyword-value-pairs*)

where the two required arguments are the action-list and a list of additional arguments passed to `execute-actions`, respectively. The remaining arguments are any number of keyword-value pairs that may be specified in the call to `execute-actions`. If no execution function is specified, then the default execution function will be used to execute the action-list.

See also `define-action-list`
 `*handle-warn-on-redefinition*`

make-**mt-random-state** *Function*

Summary	Creates an object of type <code>mt-random-state</code> .	
Package	<code>lispworks</code>	
Signature	<code>make-mt-random-state &optional state => new-state</code>	
Arguments	<code>state</code>	<code>nil</code> , <code>t</code> or an object of type <code>mt-random-state</code> . The default is <code>nil</code> .
Values	<code>new-state</code>	A new object of type <code>mt-random-state</code> .
Description	<p>The function <code>make-mt-random-state</code> creates a new object of type <code>mt-random-state</code> which is suitable for use as the value of <code>*mt-random-state*</code>.</p> <p>If <code>state</code> is an object of type <code>mt-random-state</code>, then <code>new-state</code> is a copy of <code>state</code>. If <code>state</code> is <code>nil</code>, then <code>new-state</code> is a copy of the value of <code>*mt-random-state*</code>. If <code>state</code> is <code>t</code> then <code>new-state</code> is an object of type <code>mt-random-state</code> initialized using a call to <code>get-universal-time</code>.</p> <p><code>make-mt-random-state</code> is analogous to <code>cl:make-random-state</code>.</p>	

See also **mt-random**
 mt-random-state
 mt-random-state

mt-random *Function*

Summary	Returns a pseudo-random number using the Mersenne Twister algorithm.	
Package	lispworks	
Signature	mt-random arg &optional state => random-number	
Arguments	arg A positive integer or a positive float . state An object of type mt-random-state . The default is the value of *mt-random-state* .	
Values	random-number A non-negative number less than arg and of the same type as arg .	
Description	The function mt-random returns a pseudo-random number which is non-negative, less than arg and is of the same type as arg . <i>random-number</i> is generated using the Mersenne Twister algorithm published by Makoto Matsumoto and Takuji Nishimura at http://www.math.keio.ac.jp/~matumoto/emt.html . We thank the authors for making the algorithm freely available. mt-random is analogous to cl:random .	
See also	make-mt-random-state *mt-random-state*	

mt-random-state *Variable*

Summary	The default random state used by <code>mt-random</code> .
Package	<code>lispworks</code>
Description	The variable <code>*mt-random-state*</code> contains an object of type <code>mt-random-state</code> which is the default state used by <code>mt-random</code> if a state is not supplied.
	<code>*mt-random-state*</code> is analogous to <code>cl:*random-state*</code> .
See also	<code>make-mt-random-state</code> <code>mt-random</code> <code>mt-random-state</code>

mt-random-state *Type*

Summary	The type of objects containing state information used by <code>mt-random</code> .
Package	<code>lispworks</code>
Description	The Mersenne Twister pseudo-random number generator uses state data contained in a object of type <code>mt-random-state</code> . <code>mt-random-state</code> is analogous to <code>cl:random-state</code> .
See also	<code>*mt-random-state*</code> <code>mt-random</code> <code>mt-random-state-p</code>

mt-random-state-p *Function*

Summary	The predicate for objects of type <code>mt-random-state</code> .
---------	--

Package	<code>lispworks</code>	
Signature	<code>mt-random-state-p arg => result</code>	
Arguments	<code>arg</code>	An object.
Values	<code>result</code>	A boolean.
Description	<p>The function <code>mt-random-state-p</code> returns <code>t</code> if <code>arg</code> is an object of type <code>mt-random-state</code>, and <code>nil</code> otherwise.</p> <p><code>mt-random-state-p</code> is analogous to <code>cl:random-state-p</code>.</p>	
See also	<code>mt-random-state</code>	

pathname-location *Function*

Summary	Returns the location of a file.
Signature	<code>pathname-location pathname => location</code>
Arguments	<code>pathname</code> A pathname designator.
Values	<code>location</code> A pathname.
Description	The function <code>pathname-location</code> returns a pathname <code>location</code> that represents the directory where the file <code>pathname</code> resides. Each of the name, type and version components of <code>location</code> are <code>nil</code> .
Example	<p>Due to the ANSI Common Lisp definition of the <code>directory</code> function and the fact that LispWorks returns fully specified truenames, the form</p> <pre>(directory (truename "/tmp/"))</pre> <p>will always signal an error or return the list (#P"/tmp/"). To obtain the contents of the <code>/tmp</code> directory, use the form</p>

```
(directory (pathname-location (truename "/tmp/")))
```

See also `current-pathname`
`directory`

precompile-regexp *Function*

Summary	Precompiles a regular expression object.	
Package	<code>lispworks</code>	
Signature	<code>precompile-regexp string => pattern</code>	
Arguments	<code>string</code>	A string.
Values	<code>pattern</code>	A precompiled regular expression object.
Description	The function <code>precompile-regexp</code> returns a precompiled regular expression object suitable for passing as <code>pattern</code> to <code>find-regexp-in-string</code> .	
See also	<code>find-regexp-in-string</code>	

print-actions *Function*

Summary	Generates a listing of the action items on a given action list in order.	
Package	<code>lispworks</code>	
Signature	<code>print-actions name-or-list &optional stream</code>	
Arguments	<code>name-or-list</code>	An action list.
	<code>stream</code>	A stream.

Description	Generates a listing of the action items on this action-list, in order. If the action-list specified by <i>name-or-list</i> does not exist, then this is handled according to the value of *handle-missing-action-list*.
	<i>stream</i> is an optional argument specifying where to print the output. The default value of <i>stream</i> is the value of *standard-output*.
See also	<code>print-action-lists</code>

print-action-lists *Function*

Summary	Prints a list of all the actions lists in the global registry.
Package	<code>lispworks</code>
Signature	<code>print-action-lists &optional stream</code>
Arguments	<i>stream</i> A stream.
Description	Generates a listing of all the action lists in the global registry. The ordering of the action lists is random.
	<i>stream</i> is an optional argument specifying where to print the output. The default value of <i>stream</i> is the value of *standard-output*.
See also	<code>print-actions</code>

print-command *Variable*

Summary	A command used for some printing operations.
Package	<code>lispworks</code>

Initial Value	" <code>print</code> " on Windows. " <code>lpr</code> " on UNIX/Linux/Mac OS X/FreeBSD systems.
Description	This variable is used as the command sent by LispWorks to the shell in <code>hardcopy-system</code> .
See also	<code>hardcopy-system</code>

print-nickname *Variable*

Summary	Controls the package prefix used when a symbol is printed.
Package	<code>lispworks</code>
Initial Value	<code>nil</code>
Description	<p>The variable <code>*print-nickname*</code> controls which package prefix is used when a symbol is printed and the symbol's package needs to be output.</p> <p>If <code>*print-nickname*</code> is true and the package has at least one nickname, then the first of the nicknames (that is, the first nickname in the list returned by <code>package-nicknames</code>) is output. Otherwise, the package name is output.</p>

prompt *Variable*

Summary	Defines the LispWorks listener prompt.
Package	<code>lispworks</code>
Initial Value	" <code>~%~A ~D~[~: ; ~: * : ~D~] > "</code>
Description	The variable <code>*prompt*</code> defines the LispWorks listener prompt. Its value can be a:

Function designator

A function of zero arguments which should return the prompt as a string.

String A format string with processing three arguments: the current package name, the next history number, and the debug level.

A form The form is passed to `eval` and should return a format string, which is used as for the string case above.

Examples

```
CL-USER 1 > (defvar *default-prompt* *prompt*)
*DEFAULT-PROMPT*

CL-USER 2 > (progn
  (setf *prompt*
    '(string-append "~&" ;(sys:get-user-name)
      #\Space
      (subseq *default-
*prompt* 2)))
  nil)
NIL
dubya CL-USER 3 >
```

quit

Function

Summary Quits LispWorks.

Package `lispworks`

Signature `quit &key status confirm ignore-errors-p return`

Arguments `status` An integer.

`confirm` A generalized boolean.

`ignore-errors-p` A generalised boolean.

`return` A generalized boolean.

Values	<code>quit</code> does not return, or returns <code>t</code> .
Description	The function <code>quit</code> exits LispWorks unless the user cancels the operation. There are two stages which may allow the user the chance to cancel.
	<ol style="list-style-type: none"> 1. First the action items of the action list "<code>Confirm when quitting image</code>" are run. If any action item returns <code>nil</code>, then LispWorks does not exit. 2. Otherwise, if <code>confirm</code> is true (the default value is <code>nil</code>) then a question like <code>"Do you really want to exit LispWorks?"</code> is presented to the user. If the answer No is supplied, then LispWorks does not exit. Otherwise, the action items of the action list "<code>When quitting image</code>" are run, and then LispWorks exits, and the value <code>status</code> is returned to the Operating System as the exit value of the LispWorks process. The default value of <code>status</code> is 0.
	If <code>ignore-errors-p</code> is true, then any error signalled during the running of the action list items or the confirm prompt is ignored and <code>quit</code> proceeds to exit the image. If <code>ignore-errors-p</code> is <code>nil</code> and an error is signalled during the running of the action list items, then a restart is available allowing the user to choose to continue to exit the image. The default values of <code>ignore-errors-p</code> is <code>nil</code> .
	If <code>return</code> is true and LispWorks is going to exit, then <code>quit</code> returns <code>t</code> . This can be used if you want some other Lisp process to kill the current one later, rather than it self-destructing immediately. This can be useful to allow more precise control over process termination. If <code>return</code> is <code>nil</code> then <code>quit</code> does not return. The default value of <code>return</code> is <code>nil</code> .
	Note: To make a Cocoa application quit cleanly from inside the <code>Quit</code> menu command you need to call <code>capi:destroy</code> on the application interface instead of calling <code>quit</code> . See

`capi:default-cocoa-application-interface` in the *CAPI Reference Manual* for more information.

See also [save-image](#)

rebinding *Macro*

Summary	Ensures unique names for all the variables in a groups of forms.				
Package	<code>lispworks</code>				
Signature	<code>rebinding (&rest vars) &body body => form</code>				
Arguments	<table><tr><td>vars</td><td>The variables to be rebound.</td></tr><tr><td>body</td><td>A body of forms, the variables in which should be unique.</td></tr></table>	vars	The variables to be rebound.	body	A body of forms, the variables in which should be unique.
vars	The variables to be rebound.				
body	A body of forms, the variables in which should be unique.				
Values	Returns the body wrapped in a form that creates unique names for each variable.				
Description	Returns the <i>body</i> wrapped in a form which creates a unique name for each of the variables (compare with <code>gensym</code>) and binds these names to the values of the variables. This ensures that the body can refer to the variables without name clashes with other variables elsewhere.				
Example	After defining				

```
(defmacro lister (x y)
  (rebinding (x y)
    '(list ,x ,y)))
```

the form `(lister i j)` macroexpands to

```
(LET* ((#:X-77 I)
       (#:Y-78 J))
  (LIST #:X-77 #:Y-78))
```

See also [with-unique-names](#)

regexp-find-symbols *Function*

Summary	Returns a list of symbols that match a supplied regular expression.	
Package	lispworks	
Signature	<code>regexp-find-symbols <i>regexp-string</i> &key <i>case-sensitive</i> <i>packages</i> <i>test</i> <i>external-only</i> => <i>symbols</i></code>	
Arguments	<i>regexp-string</i>	A string.
	<i>case-sensitive</i>	A boolean.
	<i>packages</i>	A list of package designators, a single package designator, or the keyword :all.
	<i>test</i>	A function of one argument returning a boolean result.
	<i>external-only</i>	A generalized boolean.
Values	<i>symbols</i>	A list of symbols.
Description	<p>The function regexp-find-symbols returns a list of symbols that match the regular expression in <i>regexp-string</i>.</p> <p><i>case-sensitive</i> determines whether the match is case sensitive. The default value of <i>case-sensitive</i> is <code>nil</code>.</p> <p><i>packages</i> specifies in which packages to search. The default value of <i>packages</i> is :all, meaning search in all packages.</p> <p><i>test</i>, if supplied, must be a function of one argument, which returns <code>t</code> if the argument should be returned, and <code>nil</code> otherwise. The function <i>test</i> is applied to each symbol that matches <i>regexp-string</i>, and if it returns <code>nil</code> the symbol is not included</p>	

in the returned value *symbols*. If *test* is `nil` all matches are returned. The default value of *test* is `nil`.

external-only, if true, specifies that only external symbols should be checked, which makes the search much faster. The default value of *external-only* is `nil`.

The regular expression syntax used by `regexp-find-symbols` is similar to that used by Emacs, as described in the "Regular expression syntax" section of the *LispWorks Editor User Guide*.

Examples

To find all exported symbols that start with DEF:

```
(lw:regexp-find-symbols "^def" :external-only t)
```

To find all symbols that contain lower case "slider":

```
(regexp-find-symbols "slider" :case-sensitive t)
```

See also

`apropos`
`find-regexp-in-string`

remove-advice

Function

Summary

Remove a piece of advice.

Package

`lispworks`

Signature

```
remove-advice dspec name => nil
dspec ::= fn-name |
           macro-name |
           (method generic-fn-name [(class*)])
```

Arguments

dspec

Specifies the functional definition to which the piece of advice belongs. The specification contains the name of the associated function. In the case of a method the list of classes is used to identify from which particular method the advice should come. This list must correspond exactly with the classes

corresponding to the specialized parameters for some method belonging to the generic function.

<i>name</i>	A symbol naming the piece of advice to be removed. Since several pieces of advice may be attached to a single functional definition, the name is necessary to indicate which one is to be removed.
Values	<code>remove-advice</code> returns <code>nil</code> .
Description	<code>remove-advice</code> is the function used to remove a piece of advice. Advice is a way of altering the behavior of functions. Pieces of advice are associated with a function using <code>defadvice</code> . They define additional actions to be performed when the function is invoked, or alternative code to be performed instead of the function, which may or may not access the original definition. As well as being attached to ordinary functions, advice may be attached to methods and to macros (in this case it is in fact associated with the macro's expansion function).
	<code>hcl:delete-advice</code> is a macro, identical in effect to <code>remove-advice</code> , except that you do not need to quote the arguments.
Notes	<code>remove-advice</code> is an extension to Common Lisp.
See also	<code>defadvice</code> <code>delete-advice</code>

removef	<i>Macro</i>
Summary	Removes an item from a sequence.
Package	<code>lispworks</code>

Signature	<code>removef place item &key test test-not start end key => result</code>	
Arguments	<i>place</i>	A place.
	<i>item</i>	An object.
	<i>test</i>	A test function.
	<i>test-not</i>	A test function.
	<i>start</i>	An integer.
	<i>end</i>	An integer or <code>nil</code> .
	<i>key</i>	A key function.
Values	<i>result</i>	A sequence.
Description	The modifying macro <code>removef</code> removes an item from a sequence using <code>remove</code> . See <code>remove</code> for more details.	
See also	<code>appendf</code>	

		<i>Variable</i>
* require-verbose*		
Summary	Controls the output of <code>require</code> .	
Package	<code>lispworks</code>	
Initial value	<code>t</code>	
Description	The variable <code>*require-verbose*</code> is a generalized boolean controlling whether <code>require</code> prints the names of the files which are being loaded.	

	<i>Function</i>
Summary	Rounds the given float to single-precision format (32 bits) and returns it as a double-float (64 bits).
Package	lispworks
Signature	round-to-single-precision float => double-float
Arguments	float A float
Values	double-float A double-float with single-float precision.
Description	<p>The argument is rounded to single-precision format (32 bits) and returned as a double-float (64 bits). This function allows you to model the rounding behaviour of a machine or implementation that performs 32-bit floating point arithmetic.</p> <p>The default size on Windows and Linux is 64 bits as specified by the IEEE standard.</p> <p>LispWorks supports 3 floating point formats, short-float, single-float and double-float. If this function is called with a single-float or a short-float, it returns the equivalent double-float, that is, it is the same as doing</p> <pre>(coerce float 'double-float)</pre>
Compatibility Note	LispWorks 4.4 and previous on Windows and Linux platforms supports just one floating point format. In LispWorks 5.0 and later, three floating point formats are supported on all platforms.
Example	<pre>CL-USER 197 > pi 3.141592653589793D0</pre> <pre>CL-USER 198 > round-to-single-precision pi 3.1415927410125732D0</pre>

	<i>Function</i>
Summary	The accessor for simple base strings.
Package	<code>lispworks</code>
Signature	<code>sbchar string index => value</code>
Arguments	<p><code>string</code> A <code>simple-base-string</code>.</p> <p><code>index</code> An index.</p>
Values	<code>value</code> The character in <code>string</code> at <code>index</code> .
Description	This is the accessor for simple base strings. <code>setf</code> is allowed.
See also	<code>simple-base-string</code>

	<i>Function</i>
Summary	Configures the value of <code>lw:*default-character-element-type*</code> .
Package	<code>lispworks</code>
Signature	<code>set-default-character-element-type type => type-defaults</code>
Arguments	<p><code>type</code> A character type. This can take any of the values <code>base-char</code>; <code>lw:simple-char</code> and <code>character</code></p>
Values	<code>type-defaults</code> The new value of <code>lw:*default-character-element-type*</code> .
Description	The function <code>set-default-character-element-type</code> sets the value of <code>lw:*default-character-element-type*</code> , ensuring that the system's internal state is also updated accordingly.

If you are running an existing 8-bit application you will only need to have this in your site or user configuration file:

```
(lw:set-default-character-element-type 'base-char)
```

It would be a mistake to call this function in a loadable package and it is not intended to be called while running code. In particular, it is global, not thread-specific.

Hence we consider `lw:*default-character-element-type*` a parameter.

See also	<code>string</code> <code>open</code> <code>*default-character-element-type*</code> <code>with-output-to-string</code>
----------	---

simple-base-string-p

Function

Summary	Tests if an object is a simple base string.	
Package	<code>lispworks</code>	
Signature	<code>simple-base-string-p object => bool</code>	
Arguments	<code>object</code>	The object to be tested.
Values	<code>bool</code>	<code>t</code> if <i>object</i> is a simple base string; <code>nil</code> otherwise.
Description	This is the predicate for simple base strings.	
See also	<code>simple-base-string</code>	

simple-char *Type*

Summary	The simple character type.
Package	<code>lispworks</code>
Signature	<code>simple-char</code>
Description	The type of simple characters (standard term for chars with null implementation-defined attributes, that is, no bits).

simple-char-p *Function*

Summary	Tests if an object is a simple character.	
Package	<code>lispworks</code>	
Signature	<code>simple-char-p object => bool</code>	
Arguments	<i>object</i>	The object to be tested.
Values	<i>bool</i>	<code>t</code> if <i>object</i> is a simple character; <code>nil</code> otherwise.
Description	The predicate for simple characters.	
See also	<code>simple-char</code>	

simple-text-string *Type*

Summary	The simple text string type.
Package	<code>lispworks</code>
Signature	<code>simple-text-string length</code>

Arguments	<i>length</i>	The length of the string (or *, meaning any).
Description	This is the simple version of <code>text-string</code> , that is, the string itself is simple. Equivalent to:	
		<code>(simple-vector lw:simple-char <i>length</i>)</code>

simple-text-string-p *Function*

Summary	Tests if an object is a simple text string.	
Package	<code>lispworks</code>	
Signature	<code>simple-text-string-p <i>object</i> => <i>bool</i></code>	
Arguments	<i>object</i> The object to be tested.	
Values	<i>bool</i> <code>t</code> if <i>object</i> is a simple text string; <code>nil</code> otherwise.	
Description	This is the predicate for simple text strings.	
See also	<code>simple-text-string</code>	

start-tty-listener *Function*

Summary	Starts a listener in the startup shell.	
Package	<code>lispworks</code>	
Signature	<code>start-tty-listener <i>force</i> => <i>process</i></code>	
Arguments	<i>force</i> A generalized boolean.	

Values	<i>process</i>	A listener process, or <code>nil</code> .
Description	<p>The function <code>start-tty-listener</code> returns a process that runs a listener read-eval-print loop connected to <code>*terminal-io*</code>. If <i>force</i> is <code>nil</code>, then <code>start-tty-listener</code> checks if the default listener process is alive or if there is a live process with name "TTY Listener". If such a process exists, <code>start-tty-listener</code> simply returns <code>nil</code> and does not start a new process. If no such process exists, or if <i>force</i> was <code>t</code>, then <code>start-tty-listener</code> starts a new listener process named "TTY Listener", and returns it.</p> <p>If a REPL with I/O through <code>*terminal-io*</code> (such as a REPL started by <code>start-tty-listener</code>) is in the debugger, then by default it blocks multiprocessing. This behavior is controlled by the value of <code>*terminal-debugger-block-multiprocessing*</code>.</p>	
See also	<code>*terminal-debugger-block-multiprocessing*</code>	

stchar		<i>Function</i>
Summary	The accessor for simple text strings.	
Package	<code>lispworks</code>	
Signature	<code>stchar string index => value</code>	
Arguments	<i>string</i>	A <code>simple-text-string</code> .
	<i>index</i>	An index.
Values	<i>value</i>	The character in <i>string</i> at <i>index</i> .
Description	This is the accessor for simple text strings. <code>setf</code> is allowed.	
See also	<code>simple-text-string</code>	

string-append	<i>Function</i>
Summary	Constructs a single string from a number of strings.
Package	lispworks
Signature	string-append &rest strings => string
Arguments	strings Any number of strings or string designators.
Values	string A string.
Description	<p>The string-append function takes any number of string designators and constructs a single string from them.</p> <p>A string designator is a string, a symbol or a character object.</p> <p>Each of the elements of the <i>strings</i> argument are first coerced into a string using the string function if they are not already a string.</p> <p><i>string</i> is a string of the "widest" type amongst <i>strings</i>. That is, the constructed string is of the same type as the argument with the largest element type.</p>
Example	<pre>(readtable-case *readtable*) => :UPCASE (string-append "foo" 'bar) => "fooBAR" (type-of (string-append (coerce "A" 'simple-base-string) (coerce "A" 'simple-text-string)))) => SIMPLE-TEXT-STRING</pre>

text-string	<i>Type</i>
Summary	The text string type.
Package	<code>lispworks</code>
Signature	<code>text-string length</code>
Arguments	<i>length</i> The length of the string (or <code>*</code> , meaning any).
Description	The type of strings that can hold any simple character, that is, (<code>vector lw:simple-char length</code>). This is the string type that is guaranteed to always hold any character used in writing text (program text or natural language). It will not hold character objects which have non-null attributes. It is equivalent to <code>16-bit-string</code> .
See also	<code>8-bit-string</code> <code>16-bit-string</code>

text-string-p	<i>Function</i>
Summary	Tests if an object is a text string.
Package	<code>lispworks</code>
Signature	<code>text-string-p object => bool</code>
Arguments	<i>object</i> The object to be tested.
Values	<code>t</code> if <i>object</i> is a text string; <code>nil</code> otherwise.
Description	This is the predicate for text strings.
See also	<code>text-string</code>

true	<i>Function</i>
Summary	Ignores its arguments and returns <code>t</code> .
Package	<code>lispworks</code>
Signature	<code>true &rest ignore => t</code>
Arguments	<code>ignore</code> All arguments are ignored.
Values	<code>t</code>
Description	The function <code>true</code> ignores all its arguments and returns <code>t</code> . It is useful as a functional argument.
See also	<code>do-nothing</code> <code>false</code>

undefine-action	<i>Macro</i>
Summary	Removes an action from a specified list.
Package	<code>lispworks</code>
Signature	<code>undefine-action name-or-list action-name =></code>
Arguments	<code>name-or-list</code> A list or action list object. <code>action-name</code> A general lisp object.
Description	The <code>undefine-action</code> macro removes the specified action from the specified list. If the action specified by <code>action-name</code> does not exist, then this is handled according to the value of <code>*handle-missing-action-in-action-list*</code> . <code>name-or-list</code> is evaluated to give either a list UID (to be looked up in the global registry of lists) or an action list object. <code>action-</code>

name is a UID (general lisp object, to be compared by `equalp`). It uniquely identifies this action within its list (as opposed to among all lists).

See also `define-action`

undefine-action-list

Macro

Summary	Removes a given defined action list.
Package	<code>lispworks</code>
Signature	<code>undefine-action-list <i>uid</i> =></code>
Arguments	<i>uid</i> A lisp object.
Values	None.
Description	The <code>undefine-action-list</code> flushes the specified list (and all its action-items). If the action-list specified by <i>uid</i> does not exist, then handling is controlled by the value of the <code>*handle-missing-action-list*</code> variable.
See also	<code>define-action-list</code>

user-preference

Function

Summary	Gets or sets a persistent value in the user's registry.
Package	<code>lispworks</code>
Signature	<code>user-preference <i>path</i> <i>value-name</i> &key <i>product</i> => <i>value</i>, <i>valuep</i></code>

Signature	<code>(setf user-preference) value path value-name &key product => value</code>	
Arguments	<i>path</i>	A string or a list of strings.
	<i>value-name</i>	A string.
	<i>product</i>	A keyword.
Values	<i>value</i>	A Lisp object.
	<i>valuep</i>	A boolean.
Description	<p>The function <code>user-preference</code> reads the value of the registry entry <i>value-name</i> under <i>path</i> under the registry path defined for <i>product</i> by <code>(setf product-registry-path)</code>. If the registry entry was found a second value <code>t</code> is returned. If the registry entry was not found, then <i>value</i> is <code>nil</code>.</p> <p>The function <code>(setf user-preference)</code> sets the value of that registry entry to <i>value</i>.</p> <p>If <i>path</i> is a list of strings, then it is interpreted like the directory component of a pathname. If <i>path</i> is a string, then any directory separators should be appropriate for the platform - that is, use backslash on Windows, and forward slash on Unix/Linux/Mac OS X systems.</p> <p>Note: when <i>value</i> is a string, <code>user-preference</code> stores a print-escaped string in the registry and reads it back with <code>read-from-string</code>. Therefore it may not work with string values stored by other software.</p> <p>Note: while <i>product</i> can in principle be any Lisp object, values of <i>product</i> are compared by <code>eq</code>, so you should use keywords.</p> <p>Note: The CAPI provides a way to store window geometry - see the entry for <code>capi:top-level-interface-save-geometry-p</code> in the <i>LispWorks CAPI Reference Manual</i>.</p>	
Example	This example is on Microsoft Windows. Note the use of backslashes as directory separators in the <i>path</i> argument:	

```
(setf (user-preference "My Stuff\\FAQ"
                      "Ultimate Answer"
                      :product :deep-thought)
      42)
=>
42
```

This is equivalent to the previous example, and is portable because we avoid the explicit directory separators in the *path* argument:

```
(setf (user-preference (list "My Stuff" "FAQ")
                      "Ultimate Answer"
                      :product :deep-thought)
      42)
=>
42
```

We can retrieve values on Windows like this:

```
(user-preference "My Stuff\\FAQ"
                  "Ultimate Answer"
                  :product :deep-thought)
=>
42
t
```

We can retrieve values on any platform like this:

```
(user-preference (list "My Stuff" "FAQ")
                  "Ultimate Question"
                  :product :deep-thought)
=>
nil
nil
```

See also [copy-preferences-from-older-version](#)
[product-registry-path](#)

when-let

Macro

Summary Executes a body of code if a form evaluates to non-`nil`, propagating the result of the form through the body of code.

Package	<code>lispworks</code>
Signature	<code>when-let (var form) &body body => result</code>
Arguments	<p><code>var</code> A variable whose value is used in the evaluation of <i>body</i>.</p> <p><code>form</code> A form, which must evaluate to <code>non-nil</code>.</p> <p><code>body</code> A body of code to be evaluated conditionally on the result of <i>form</i>.</p>
Values	<code>result</code> The result of evaluating <i>body</i> using the value <i>var</i> .
Description	This macro executes the body of code if the form evaluates to a <code>non-nil</code> result. Within the body, the variable <i>var</i> is bound to the result of the <i>form</i> .
Example	<p>The form</p> <pre>(when-let (position (search string1 string2)) (print position))</pre> <p>macroexpands to</p> <pre>(let ((position (search string1 string2))) (when position (print position)))</pre>
See also	<code>when-let*</code>

when-let*	<i>Macro</i>
Summary	Executes a body of code if a series of forms evaluates to <code>non-nil</code> , propagating the results of the forms through the body of code.
Package	<code>lispworks</code>

Signature	<code>when-let* bindings &body body => result</code> <code>bindings ::= ((var form)*)</code>	
Arguments	<code>var</code>	A variable whose value is used in the evaluation of <i>body</i> .
	<code>form</code>	A form, which must evaluate to non- <code>nil</code> .
	<code>body</code>	A body of code to be evaluated conditionally on the result of <i>form</i> .
Values	<code>result</code>	The result of evaluating <i>body</i> using the value <i>var</i> .
Description	<p>The macro <code>when-let*</code> expands into nested <code>when-let</code> forms. The bindings are evaluated in turn as long as each returns non-<code>nil</code>. If the last binding evaluates to non-<code>nil</code>, <i>body</i> is executed. Within the code <i>body</i>, each variable <i>var</i> is bound to the result of the corresponding form <i>form</i>.</p>	
Example	<pre>(defmacro divisible (n &rest divisors) `(when-let* ,(loop for div in divisors collect (list (gensym) (zerop (mod n div))))) t))</pre>	
See also	<code>when-let</code>	

whitespace-char-p *Function*

Summary	Tests whether a character represents white space.
Package	<code>lispworks</code>
Signature	<code>whitespace-char-p char => bool</code>
Arguments	<code>char</code> A character.

Values	<i>bool</i>	<i>t</i> if char represents white space; <i>nil</i> otherwise.
Description	This predicate recognizes <code>[whitespace]</code> , as described in the standard: “Space and non-graphic characters that only moved the print position.”	
See also	<code>*extended-spaces*</code>	

with-action-item-error-handling Macro

Summary	Executes a body of code across action lists and items, signaling errors and then continuing to the next action item.		
Package	<code>lispworks</code>		
Signature	<code>with-action-item-error-handling <i>action-list-var</i> <i>action-item-var</i> <i>ignore-errors-p</i> &body <i>body</i></code>		
Arguments	<p><i>action-list-var</i> A variable.</p> <p><i>action-item-var</i> A variable.</p> <p><i>ignore-errors-p</i> A boolean.</p> <p><i>body</i> A body of Lisp code.</p>		
Description	The <code>with-action-item-error-handling</code> macro executes the <i>body</i> with <i>action-list-var</i> and <i>action-item-var</i> are bound to the action list and item respectively. If <i>ignore-errors-p</i> is set to <i>t</i> then errors are handled. The behavior of the handler is to signal a warning in which the action-list, item and original error are all reported; execution then continues with the next action-item.		

Example

```
(defun my-execution-function (the-action-list
                                other-args
                                &key ignore-errors-p
                                &allow-other-keys)
  (with-action-list-mapping (the-action-list
                            an-action-item
                            action-item-data)
    (with-action-item-error-handling (the-action-list
                                       an-action-item
                                       ignore-errors-p)
      (do-something-interesting-first)
      (apply (car action-item-data) other-args (cdr
action-item-data)))))
```

If this function was invoked with the keyword argument `:ignore-errors-p t`, and an error was signalled while executing the body-form(s) for one of the action-items, then a warning such as:

```
Warning: Got an error 'The variable *PREV-STATE* is
unbound.' while executing action "Initialize State" in
list "Startup Inits".
```

would be signalled and execution would continue with the next action-item.

See also [*handle-missing-action-in-action-list*](#)

with-action-list-mapping *Macro*

Summary Maps over an action list's actions with given variables bound to the executing action and its data.

Package **lispworks**

Signature **with-action-list-mapping *action-list* *item-var* *data-var***
&optional *post-process* &*body* *body*)

Arguments *action-list* An action list.

item-var A Lisp symbol.

<i>data-var</i>	A Lisp symbol.
<i>post-process</i>	A keyword.
<i>body</i>	A body of Lisp code.
Description	<p>The <code>with-action-list-mapping</code> macro maps over an action-list's action-items. During execution, the symbols specified for <i>item-var</i> and <i>data-var</i> are bound to the executing action-item and its data respectively. See <code>execute-actions</code> for more on post-processing.</p> <p>If this function is invoked with the keyword argument <code>:post-process :collect</code>, a list the values returned by each action-item's setf operation are returned.</p>
Examples	<pre>(defun my-execution-function (the-action-list other-args &key (post-process nil) &allow-other-keys) (declare (ignore other-args)) (with-action-list-mapping (the-action-list an-action-item action-item-data post-process) (do-something-interesting-first) (setf (symbol-value (car action-item-data)) (apply (cadr action-item-data) (cddr action-item-data)))))</pre>
See also	<code>execute-actions</code>

with-unique-names *Macro*

Summary	Returns a body of code with each specified name bound to a similar name.
Package	<code>lispworks</code>
Signature	<code>with-unique-names (&rest names) &body body => result</code>

Arguments	<i>names</i>	The names to be rebound in <i>body</i> .
	<i>body</i>	The body of code within which <i>names</i> are rebound.
Values	<i>result</i>	The result of evaluating <i>body</i> .
Description		Returns the body with each <i>name</i> bound to a symbol of a similar name (compare <code>gensym</code>).

Example After defining

```
(defmacro lister (p q)
  (with-unique-names (x y)
    `(let ((,x (x-function))
          (,y (y-function)))
      (list ,p ,q ,x ,y))))
```

the form `(lister i j)` macroexpands to

```
(LET* ((#:X-88 (X-FUNCTION))
      (#:Y-89 (Y-FUNCTION)))
  (LIST i j #:X-88 #:Y-89))
```

See also [rebinding](#)

11

The MP Package

This chapter describes symbols available in the `mp` package, giving you access to the multiprocessing capabilities of LispWorks. You should use this chapter in conjunction with the relevant chapter in the *LispWorks User Guide*.

		<i>Function</i>
Summary		Changes the priority of a process.
Package		<code>mp</code>
Signature		<code>change-process-priority process new-priority => new-priority</code>
Arguments	<i>process</i>	A process.
	<i>new-priority</i>	A fixnum.
Description		Changes the priority of <i>process</i> to be <i>new-priority</i> .
See also		<code>process-priority</code>

	create-simple-process	<i>Function</i>
Summary	Creates and returns a simple process, which is a process with no stack of its own.	
Signature	<code>create-simple-process name function wait-function &key function-arguments wait-function-arguments priority => process</code>	
Package	<code>mp</code>	
Arguments	<p><i>name</i> A string or symbol</p> <p><i>function</i> A function</p> <p><i>wait-function</i> A function</p> <p><i>function-arguments</i> A list of arguments for <i>function</i></p> <p><i>wait-function-arguments</i> A list of arguments for <i>wait-function</i></p> <p><i>priority</i> A fixnum</p>	
Values	<i>process</i> A simple process	
Description	<p>The <code>create-simple-process</code> function creates and returns a simple process, which is a process that has no stack of its own.</p> <p>The <i>name</i> argument is a string or symbol that names the process. The <i>function</i> argument is a function to be run in the process, and the <i>wait-function</i> argument is a wait function that determines when the process function is run. The value of <i>function-arguments</i> is a list of arguments to which the process function is applied. The value of <i>wait-function-arguments</i> is a list of arguments to which the wait function is applied. The <i>:priority</i> argument is a fixnum that specifies a priority for the process. The default priority is the value of <code>*default-simple-process-priority*</code>, and is usually 0.</p>	

When the `wait` function, applied to the *wait-function-arguments*, returns a value other than `nil`, the process function is applied to the function-arguments. The process function is executed inside an `mp:without-preemption` form. If an error occurs in a simple process, that process is stopped and a continuable error is signaled in the process that was running at the time the simple process was started (or the last process to run if the system was idle). Continuing from the error restarts the simple process.

Because a simple process has no stack of its own, it can be executed on an arbitrary stack. However, simple processes have restrictions, the primary one being that they cannot block. The following interfaces cannot be used in a simple process:

- `mp:mailbox-read` (with an empty mailbox)
- `mp:process-allow-scheduling`
- `mp:process-lock`
- `mp:process-wait`
- `mp:process-wait-with-timeout`
- `cl:sleep`
- `mp:sleep-for-time`
- `mp:wait-for-mailbox`
- `mp:wait-processing-events`
- `mp:with-lock`
- CAPI functions that block

Other Common Lisp functions might not work if they attempt to block. This applies in particular to I/O functions on streams such as pipes and to (`setf gethash`) on a hash table that another process is mapping over.

For more information, see the “Multiprocessing” chapter of the *LispWorks User Guide*.

Example	The following example creates a simple process that prints the value of <code>*a*</code> to the background output when the value is other than <code>nil</code> . The process function then sets <code>*a*</code> to <code>nil</code> . From a listener, the value of <code>*a*</code> can be set to trigger the process to run once and then sleep again.
---------	--

```
(defvar *a* 'i)

*A*

(defun a ()
  (let ((a *a*))
    (setq *a* nil)
    (format mp:*background-standard-output*
            "*a* is ~a~%" a)))

A

(defun b () *a*)

B

(setq r (mp:create-simple-process 'test-proc 'a 'b))

#<MP::SIMPLE-PROCESS Name TEST-PROC Priority 0 State
NIL>
```

See also	<code>process-run-function</code>
----------	-----------------------------------

current-process *Variable*

Summary	Contains the object that is the current process.
---------	--

Package	<code>mp</code>
---------	-----------------

Description	This special variable contains the object that is the current process.
-------------	--

See also	<code>get-current-process</code>
----------	----------------------------------

		<i>Function</i>
Summary	Debug a thread other than the current process.	
Package	<code>mp</code>	
Signature	<code>debug-other-process <i>process</i></code>	
Arguments	<code><i>process</i></code>	A process or a string.
Description		The function <code>mp:debug-other-process</code> causes the debugger to be entered in the given process. If <i>process</i> is a string, the process is found as if by <code>mp:find-process-from-name</code> . The list of process names can be found via <code>mp:ps</code> .
See also		<code>find-process-from-name</code> <code>ps</code>

		<i>Variable</i>
Summary		The default priority for processes.
Package	<code>mp</code>	
Description		The <code>*default-process-priority*</code> variable contains the default priority for processes.
See also		<code>process-run-function</code> <code>create-simple-process</code> <code>*default-simple-process-priority*</code>

		<i>Variable</i>
Summary		The default priority for simple processes.

Package	<code>mp</code>
Description	The <code>*default-simple-process-priority*</code> variable contains the default priority for simple processes.
See also	<code>create-simple-process</code> <code>*default-process-priority*</code>

ensure-process-cleanup

Function

Summary	Run forms when a given process terminates.	
Package	<code>mp</code>	
Signature	<code>ensure-process-cleanup cleanup-form &optional process =></code>	
Arguments	<code>cleanup-form</code>	Form to run when <i>process</i> terminates.
	<code>process</code>	The process to watch for termination. By default, this is the value of <code>mp:*current-process*</code> .
Values	None.	
Description	<p>Ensures that the <i>cleanup-form</i> is present for the given process. When the process terminates, its cleanup forms are run. Cleanup forms can be functions of one argument (the process), or lists, in which case the <code>car</code> is applied to the process and the <code>cdr</code> of the list.</p> <p>When adding cleanup forms, this function uses <code>equal</code> to ensure that the form is only added once.</p>	
Example	A process calls <code>add-process-dependent</code> each time a dependent object is added to a process. When the process terminates, <code>inform-dependent-of-dead-process</code> is called on all dependent objects.	

```

(defun add-process-dependent (dependent)
  (mp:ensure-process-cleanup
    `(delete-process-dependent ,dependent)))

(defun delete-process-dependent (process dependent)
  (inform-dependent-of-dead-process dependent process))

```

See also [process-kill](#)

find-process-from-name *Function*

Summary Finds a process from its name.

Package [mp](#)

Signature `find-process-from-name process-name => result`

Arguments *process-name* A string.

Values *result* A `mp:process`, or `nil`.

Description The function `find-process-from-name` returns the process with the name *process-name*.

If there is no such process, the function returns `nil`.

Example CL-USER 16 > (`(mp:find-process-from-name "Listener 1")`
`#<MP:PROCESS Name "Listener 1" Priority 600000 State "Running">`

See also [get-process](#)

get-current-process *Function*

Summary Returns the current Lisp process.

Package [mp](#)

Signature	<code>get-current-process => result</code>	
Values	<code>result</code>	A <code>mp:process</code> , or <code>nil</code> .
Description	<p>The function <code>get-current-process</code> returns the actual process in which it is called. In this respect it differs from <code>*current-process*</code>, which can be bound to another process. In particular, when a process A calls the <i>wait-function</i> of process B, in the <i>wait-function</i> <code>get-current-process</code> returns the process A, but <code>*current-process*</code> is bound to process B.</p> <p><code>result</code> is <code>nil</code> if multiprocessing is off.</p>	
See also	<code>*current-process*</code>	

get-process *Function*

Summary	Returns a process corresponding to a supplied designator.	
Package	<code>mp</code>	
Signature	<code>get-process process-designator => process</code>	
Arguments	<p><code>process-designator</code></p> <p>A <code>mp:process</code>, a string, a stack-group, a function, a symbol or a fixnum.</p>	
Values	<code>process</code>	A <code>mp:process</code> .
Description	<p>The function <code>get-process</code> returns a process according to the supplied <code>process-designator</code>, which is interpreted as follows:</p> <ul style="list-style-type: none"> <code>mp:process</code> Return it. A string Find the first process (highest priority) with matching name. Process names are compared by <code>string=</code>. 	

A stack-group	Return the process of the stack-group.
A function	Return the first process that has <i>process-designator</i> as its function (that is, the third argument of <code>process-run-function</code>).
A symbol	First search for a process using the symbol name as a string, and (if that fails) then search using the symbol as a function.
A fixnum	Find a process for which <i>process-designator</i> is its unique id. The unique id of the current process can be found by <code>(sys:current-thread-unique-id)</code> . <i>result</i> is <code>nil</code> if multiprocessing is off.

See also `find-process-from-name`

	<i>Function</i>
Summary	Initializes multiprocessing before use.
Package	<code>mp</code>
Signature	<code>initialize-multiprocessing &rest main-process-args => nil</code>
Arguments	<code>main-process-args</code> A set of arguments for <code>process-run-function</code> .
Values	Returns <code>nil</code> .
Description	The function <code>initialize-multiprocessing</code> initializes multiprocessing, and it does not return until multiprocessing is finished.

`initialize-multiprocessing` applies the function `process-run-function` to each of the entries in `*initial-processes*` to create the initial processes.

When called with `main-process-args`, it creates an `mp:process` object for the initial thread using the arguments in that list as if in the call

```
(apply 'mp:process-run-function main-process-args)
```

Supplying `main-process-args` is useful on Mac OS X if you want to run a pure Cocoa application, since the main thread needs to run the Cocoa event loop.

It is not necessary to call `initialize-multiprocessing` when the Common LispWorks GUI is running (that is, after `env:start-environment` has been called), as this automatically starts up multiprocessing.

Note: On Microsoft Windows, Linux, FreeBSD and Mac OS X (using the Cocoa image), the Common LispWorks GUI starts up by default.

See also `*initial-processes*`
`process-run-function`

initial-processes		<i>Special Variable</i>
Summary	A list of the processes the system initializes on startup.	
Package	<code>mp</code>	
Description	The variable <code>*initial-processes*</code> specifies the processes which the system initializes on startup. Each element of the <code>*initial-processes*</code> list is a set of arguments for <code>process-run-function</code> .	

Example To create a listener process as well as your own processes, evaluate this form before saving your image:

```
(push mp::*default-listener-process*
      mp::*initial-processes*)
```

See also [process-run-function](#)

list-all-processes *Function*

Summary Lists all the Lisp processes currently in the system.

Package `mp`

Signature `list-all-processes => process-list`

Arguments None.

Values `process-list` A list of all the currently active Lisp processes.

Description Returns a list of all the active Lisp processes in LispWorks.

Example	<code>CL-USER 71 > (pprint (mp:list-all-processes))</code>
---------	---

```
(#<MP:PROCESS Name "Editor 1" Priority 70000000 State
"Waiting for events">
#<MP:PROCESS Name "Listener 1" Priority 70000000 State
"Running">
#<MP:PROCESS Name "LispWorks 5.1.0" Priority 70000000
State "Waiting for events">
#<MP:PROCESS Name "default listener process" Priority
60000000 State "Waiting for terminal input.">
#<MP:PROCESS Name "CAPI Execution Listener 1" Priority
60000000 State "Running">
#<MP:PROCESS Name "Background execute 2" Priority
50000000 State "Waiting for job to execute">
#<MP:PROCESS Name "Background execute 1" Priority
50000000 State "Waiting for job to execute">
#<MP:PROCESS Name "Editor DDE server" Priority 0 State
"Waiting for an event">
#<MP:PROCESS Name "The idle process" Priority -
536870912 State "Running (preempted)">)
```

lock-name	<i>Function</i>
Summary	Returns the name of a lock.
Package	<code>mp</code>
Signature	<code>lock-name <i>lock</i> => <i>name</i></code>
Arguments	<code><i>lock</i></code> A lock object
Values	<code><i>name</i></code> A string
Description	The <code>lock-name</code> function takes a lock object as its argument and returns the name of the lock object.
Example	<pre>(let ((lock (mp:make-lock :name "my lock"))) (mp:lock-name lock)) => "my lock"</pre>

See also

`make-lock`
`with-lock`
`process-lock`
`process-unlock`
`lock-owner`

lock-owner

Function

Summary	Returns the owner of a lock.	
Package	<code>mp</code>	
Signature	<code>lock-owner lock => result</code>	
Arguments	<code>lock</code>	A lock object
Values	<code>result</code>	A process, <code>t</code> or <code>:unknown</code>
Description	<p>The <code>lock-owner</code> function returns the process that currently owns the lock, or <code>nil</code>.</p> <p>If <code>lock</code> is locked then <code>result</code> is normally the process that locked it. If <code>lock</code> was locked while multiprocessing was not running then <code>result</code> is <code>t</code>. Also, if <code>lock</code> was locked by an unknown process (for example, the process is killed whilst holding the lock) then <code>result</code> is <code>:unknown</code>.</p> <p><code>result</code> is <code>nil</code> if <code>lock</code> is not locked.</p>	

Example	<pre>CL-USER 1 > (let ((lock (mp:make-lock :name "my lock"))) (mp:lock-owner lock)) NIL CL-USER 2 > (let ((lock (mp:make-lock :name "my lock"))) (mp:with-lock (lock) (mp:lock-owner lock))) #<MP:PROCESS Name "CAPI Execution Listener 1" Priority 0 State "Running"></pre>
See also	make-lock with-lock process-lock process-unlock lock-name

	<i>Function</i>
Summary	Tests whether a mailbox is empty.
Package	<code>mp</code>
Signature	<code>mailbox-empty-p <i>mailbox</i> => <i>bool</i></code>
Arguments	<code><i>mailbox</i></code> A mailbox
Values	<code><i>bool</i></code> A generalized boolean
Description	The <code>mailbox-empty-p</code> function returns <code>t</code> if the given <code>mailbox</code> is empty and <code>nil</code> otherwise.
See also	mailbox-send mailbox-peek mailbox-read make-mailbox

mailbox-peek *Function*

Summary	Peeks at the first object in a mailbox.	
Package	<code>mp</code>	
Signature	<code>mailbox-peek mailbox => result</code>	
Arguments	<code>mailbox</code>	A mailbox.
Values	<code>result</code>	Any object or <code>nil</code> .
Description	The <code>mailbox-peek</code> function returns the first object in the mailbox without removing it. If the mailbox is empty, <code>nil</code> is returned.	
See also	<code>mailbox-empty-p</code> <code>mailbox-send</code> <code>mailbox-read</code> <code>make-mailbox</code>	

mailbox-read *Function*

Summary	Reads the next object in a mailbox.	
Package	<code>mp</code>	
Signature	<code>mailbox-read mailbox &optional wait-reason timeout => object</code>	
Arguments	<code>mailbox</code>	A mailbox.
	<code>wait-reason</code>	A string or <code>nil</code> .
	<code>timeout</code>	A non-negative number or <code>nil</code> .
Values	<code>object</code>	Any object.

Description	The <code>mailbox-read</code> function returns the next object from the mailbox <i>mailbox</i> , or <code>nil</code> . If <i>mailbox</i> is empty and <i>timeout</i> is <code>nil</code> , then <code>mailbox-read</code> blocks until an object is placed in <i>mailbox</i> . If <i>mailbox</i> is empty and <i>timeout</i> is a number, then <code>mailbox-read</code> blocks until an object is placed in <i>mailbox</i> or <i>timeout</i> seconds have passed. If the timeout occurs, then <code>mailbox-read</code> returns <code>nil</code> . The <i>wait-reason</i> argument (or the string " <code>Waiting for message</code> " if <i>wait-reason</i> is <code>nil</code>) and the <i>timeout</i> argument are both passed to <code>process-wait-with-timeout</code> . The default value of <i>wait-reason</i> is <code>nil</code> and the default value of <i>timeout</i> is <code>nil</code> .
See also	<code>mailbox-empty-p</code> <code>mailbox-peek</code> <code>mailbox-send</code> <code>mailbox-wait-for-event</code> <code>make-mailbox</code> <code>process-wait-with-timeout</code>

		<i>Function</i>
Summary	Returns the reader process of the mailbox.	
Package	<code>mp</code>	
Signature	<code>mailbox-reader-process mailbox => process</code>	
Arguments	<code>mailbox</code>	A mailbox
Values	<code>process</code>	A process
Description	The <code>mailbox-reader-process</code> function returns the reader process of <i>mailbox</i> .	

See also **process-send**

mailbox-send *Function*

Summary Sends an object to a mailbox.

Package **mp**

Signature **mailbox-send** *mailbox object* =>

Arguments *mailbox* A mailbox
object An object

Description The **mailbox-send** sends *object* to *mailbox*. The object is queued in the mailbox for retrieval by the reader.

See also **mailbox-empty-p**
mailbox-peek
mailbox-read
make-mailbox

mailbox-wait-for-event *Function*

Summary Waits for an event in a "windowing friendly" way.

Package **mp**

Signature **mailbox-wait-for-event** *mailbox &key wait-reason wait-function process-other-messages-p no-hang-p stop-at-user-operation-p*
=> *event*

Arguments *mailbox* A mailbox.
wait-reason A string or **nil**.
wait-function A function designator.

	<i>process-other-messages-p</i>	
		A generalized boolean.
	<i>no-hang-p</i>	A generalized boolean.
	<i>stop-at-user-operation-p</i>	
		A generalized boolean.
Values	<i>result</i>	An event or <code>nil</code> .
Description	<p>The function <code>mailbox-wait-for-event</code> waits for an event in a mailbox in a "windowing friendly" way. It reads an event from the mailbox <i>mailbox</i>. If there is no event in the mailbox, it waits for an event (unless <i>no-hang-p</i> is true).</p> <p>The value <i>event</i> is any object that was put in the mailbox, or <code>nil</code> if the mailbox is empty, possibly after waiting.</p> <p><code>mailbox-wait-for-event</code> is the appropriate way to wait for an event in a mailbox in an application with a graphical user interface, because it interacts correctly with the windowing system. Most importantly, on Microsoft Windows, when <i>process-other-messages-p</i> is true it processes Windows messages while it is waiting. The default value of <i>process-other-messages-p</i> is <code>t</code>.</p> <p><i>wait-function</i> is the wait function to be used, which is called with the mailbox <i>mailbox</i> as its argument. If <i>wait-function</i> is not supplied, a function that returns <code>t</code> when the mailbox is not empty is used internally.</p> <p><i>wait-reason</i> is used as the wait reason if it needs to wait. The default value of <i>wait-reason</i> is "Waiting for an event".</p> <p><i>process-other-messages-p</i> controls processing of other messages. On Microsoft Windows this means Windows messages. On other platforms it has no effect.</p> <p><i>no-hang-p</i> controls whether <code>mailbox-wait-for-event</code> should really wait. If <i>no-hang-p</i> is true and there is no event, it returns immediately except on Microsoft Windows, where it may</p>	

first process all Windows messages (depending on the value of *process-other-messages-p*). The default value of *no-hang-p* is `nil`.

stop-at-user-operation-p on Microsoft Windows causes `mailbox-wait-for-event` to return if it received a user operation message (meaning keyboard or mouse input). It has no effect on other platforms. The default value of *stop-at-user-operation-p* is `nil`.

If `mailbox-wait-for-event` is called when not Lisp is not multiprocessing, it returns immediately. The return value is an event or `nil`.

See also

`mailbox-read`
`mailbox-send`
`make-mailbox`
`process-wait-for-event`

main-process

Variable

Summary The process associated with the main thread.

Package `mp`

Description This special variable contains the process associated with the main thread of the application. On Mac OS X with the Cocoa GUI, this is the thread that runs the Cocoa event loop. On other platforms, this variable is always `nil`.

make-lock

Function

Summary Makes a lock.

Package `mp`

Signature	<code>make-lock &key name important-p safep => lock</code>	
Arguments	<code>name</code>	A string.
	<code>important-p</code>	A generalized boolean.
	<code>safep</code>	
Values	<code>lock</code>	The lock object.
Description	<p><code>make-lock</code> returns a lock object. See the "Multiprocessing" chapter of the <i>LispWorks User Guide</i> for a general description of locks.</p> <p><code>name</code> names the lock and can be queried with <code>mp:lock-name</code>. The default value of <code>name</code> is "Anon".</p> <p><code>important-p</code> controls whether the lock is automatically freed when the holder process finishes. When <code>important-p</code> is true, the lock is pushed onto the list <code>mp:*important-locks*</code>. Locks in this list are automatically freed when the holder process finishes. <code>important-p</code> should be <code>nil</code> for locks which are managed completely by the application, as it is wasteful to record all locks in a global list if there is no need to free them automatically. This might be appropriate when two processes sharing a lock must both be running for the system to be consistent. If one process dies, then the other one kills itself. Thus the system does not need to worry about freeing the lock because no process could be waiting on it forever after the first process dies. The default value of <code>important-p</code> is <code>nil</code>.</p> <p><code>safep</code> controls whether the lock is safe. A safe lock gives an error if <code>process-unlock</code> is called on it when it is not locked by the current process, and potentially in other 'dangerous' circumstances. An unsafe lock does not signal these errors. The default value of <code>safep</code> is <code>t</code>.</p>	

Example

```

CL-USER 3 > (setq *my-lock* (mp:make-lock
                                         :name "my-lock"))
#<MP:LOCK "my-lock" Unlocked 2008CAC7>

CL-USER 4 > (mp:process-lock *my-lock*)
T

CL-USER 5 > *my-lock*
#<MP:LOCK "my-lock" Locked once by "CAPI Execution
Listener 1" 2008CAC7>

CL-USER 6 > (mp:process-lock *my-lock*)
T

CL-USER 7 > *my-lock*
#<MP:LOCK "my-lock" Locked 2 times by "CAPI Execution
Listener 1" 2008CAC7>

```

See also

- [create-simple-process](#)
- [process-lock](#)
- [process-unlock](#)
- [schedule-timer](#)
- [with-lock](#)

make-mailbox *Function*

Summary

Make a new mailbox for the current process.

Package

`mp`

Signature

`make-mailbox &key size => mailbox`

Arguments

<code>size</code>	An integer
-------------------	------------

Values

<code>mailbox</code>	A mailbox
----------------------	-----------

Description

The function `make-mailbox` returns a new mailbox.
`size` specifies the initial size of the mailbox.
The reader process is set to the current process.

See also [mailbox-empty-p](#)
[mailbox-peek](#)
[mailbox-read](#)
[mailbox-send](#)

make-named-timer *Function*

Summary Creates and returns a named timer.

Package `mp`

Signature `make-named-timer name function &rest arguments => timer`

Arguments `name` A string or symbol

`function` A function

`arguments` A set of arguments to `function`

Values `timer` A timer

Description The `make-named-timer` function creates and returns a named timer. The first argument is a string or symbol naming the timer. The second argument is a function to be applied to the remaining arguments when the timer expires. Use the function `schedule-timer` or `schedule-timer-relative` to set an expiration time.

In comparison, the function `make-timer` creates an unnamed timer.

Example

```
(setq timer (mp:make-named-timer 'timer-1 'print 10
*standard-output*))
```

```
#<Time Event TIMER-1 : PRINT>
```

See also [make-timer](#)
[schedule-timer](#)
[schedule-timer-milliseconds](#)

```
schedule-timer-relative  
schedule-timer-relative-milliseconds  
timer-expired-p  
timer-name  
unschedule-timer
```

make-timer	<i>Function</i>
-------------------	-----------------

Summary Creates and returns an unnamed timer.

Signature **make-timer** *function &rest arguments => timer*

Package **mp**

Arguments *function* A function

arguments A set of arguments to *function*

Values *timer* A timer

Description The **make-timer** function creates and returns an unnamed timer. The *function* argument is a function to be applied to the remaining arguments when the timer expires. Use the function **schedule-timer** or **schedule-timer-relative** to set an expiration time.

Note that the function **make-named-timer** creates a named timer.

Example

```
(setq timer
      (mp:make-timer 'print 10 *standard-output*))

#<Time Event : PRINT>
```

See also **make-named-timer**
make-timer
schedule-timer
schedule-timer-milliseconds

```

schedule-timer-relative
schedule-timer-relative-milliseconds
timer-expired-p
timer-name
unschedule-timer

```

map-all-processes *Function*

Summary	Calls the function for all processes.	
Package	<code>mp</code>	
Signature	<code>map-all-processes</code> <i>function</i> =>	
Arguments	<i>function</i>	A function taking one argument
Values	None.	
Description	<p>The function <code>map-all-processes</code> calls <i>function</i> for every process, including simple processes.</p> <p><i>function</i> is passed each process as its single argument.</p>	
See also	<code>map-processes</code>	

map-all-processes-backtrace *Function*

Summary	Produces a backtrace for every known process.	
Package	<code>mp</code>	
Signature	<code>map-all-processes-backtrace</code> &optional <i>function</i>	
Arguments	<i>function</i>	A function taking one argument

Values	None.
Description	The <code>map-all-processes-backtrace</code> function calls <i>function</i> , which defaults to <code>print</code> , for every known process and each line of its backtrace.
See also	<code>map-process-backtrace</code>

map-process-backtrace *Function*

Summary	Produces a backtrace for a process	
Package	<code>mp</code>	
Signature	<code>map-process-backtrace process function</code>	
Arguments	<i>process</i>	A process
	<i>function</i>	A function taking one argument
Values	None.	
Description	The <code>map-process-backtrace</code> function collects a backtrace for the process specified by <i>process</i> , and the function <i>function</i> is called on each line of the backtrace in turn.	

Example

```
CL-USER 1 > (mp:map-process-backtrace mp:*current-
process* 'print)

DBG:::GET-CALL-FRAME
MP:MAP-PROCESS-BACKTRACE
SYSTEM::%INVOKE
SYSTEM::%EVAL
EVAL
SYSTEM::DO-EVALUATION
SYSTEM::%TOP-LEVEL-INTERNAL
SYSTEM::%TOP-LEVEL
SYSTEM::LISTENER-TOP-LEVEL
CAPI::CAPI-TOP-LEVEL-FUNCTION
CAPI::INTERACTIVE-PANE-TOP-LOOP
(SUBFUNCTION MP::PROCESS-SG-FUNCTION MP::INITIALIZE-
PROCESS-STACK)
SYSTEM::%%FIRST-CALL-TO-STACK
NIL
```

See also

[map-all-processes-backtrace](#)

map-processes*Function***Summary**

Calls the function for all non-simple processes.

Package

`mp`

Signature

`map-processes function`

Arguments

function A function taking one argument

Values

None.

Description

The function `map-processes` calls *function* for every non-simple process.

function is passed each such process as its single argument.

See also

[map-all-processes](#)

notice-fd *Function*

Summary	Add a file descriptor to the set of interesting input file descriptors.	
Package	<code>mp</code>	
Signature	<code>notice-fd fd</code>	
Arguments	<code>fd</code>	A UNIX file descriptor
Values	None.	
Description	<p>The <code>notice-fd</code> function adds the given <i>fd</i> to the set of fds that cause LispWorks to wake up when they contain input.</p> <p>This function is not implemented on Microsoft Windows.</p>	
See also	<code>unnotice-fd</code>	

process-alive-p *Function*

Summary	Determines if a process is alive.	
Package	<code>mp</code>	
Signature	<code>process-alive-p process => bool</code>	
Arguments	<code>process</code>	A process
Values	<code>bool</code>	A boolean
Description	<p>The <code>process-alive-p</code> function returns <code>t</code> if <i>process</i> is alive, that is, if <code>mp:process-kill</code> has not been called on the process.</p>	
Example	<code>(mp:process-alive-p mp:*current-process*)</code>	

```
=> T

(let ((process (mp:process-run-function
                 "test" nil 'identity nil)))
  (sleep 2)
  (mp:process-alive-p process))

=> NIL
```

process-allow-scheduling *Function*

Summary Allows scheduling within a process, so that the process is interruptible.

Package `mp`

Signature `process-allow-scheduling =>`

Arguments None.

Values None.

Description This gives other Lisp processes a chance to run.

process-arrest-reasons *Function*

Summary Returns a list of the reasons why a Lisp process has stopped.

Package `mp`

Signature `process-arrest-reasons process => reasons`

Arguments `process` A process.

Values `reasons` A list of reasons.

Description	Returns a list of the reasons why a Lisp process has stopped. A process is inactive if it has any arrest reasons.
	Use of <code>(setf mp:process-arrest-reasons)</code> is deprecated. You should use <code>process-stop</code> instead. If you set the arrest reasons of the current process, this causes the current process to stop immediately, before returning from <code>mp:process-</code> <code>arrest-reasons</code> (like <code>process-stop</code>).
Compatibility note	The immediate stopping behavior of <code>(setf mp:process-</code> <code>arrest-reasons)</code> is different from LispWorks 5.0 and previous versions.
See also	<code>process-run-reasons</code> <code>process-stop</code>

process-break *Function*

Summary	Breaks a Lisp process and enters the debugger.
Package	<code>mp</code>
Signature	<code>process-break process =></code>
Arguments	<code>process</code> A process.
Values	None.
Description	The function <code>process-break</code> forces the process <code>process</code> to break and enter the debugger.

process-continue *Function*

Summary	Wakes up a process.
---------	---------------------

Package	<code>mp</code>
Signature	<code>process-continue process => nil</code>
Arguments	<code>process</code> A <code>mp:process</code> object.
Description	The function <code>process-continue</code> wakes up the process <code>process</code> , regardless of whether it is sleeping, stopped or waiting. <code>process-continue</code> returns <code>nil</code> .

process-idle-time *Function*

Summary	Returns the time for which a process has been idle.
Package	<code>mp</code>
Signature	<code>process-idle-time process => time</code>
Arguments	<code>process</code> A process.
Values	<code>time</code> A non-negative integer.
Description	The function <code>process-idle-time</code> returns the length of time in internal time units that <code>process</code> has been idle. If the process is running (for example the current process) then the return value is 0.
See also	<code>process-run-time</code>

process-initial-bindings *Special Variable*

Summary	Specifies the variables initially bound in a new process.
Package	<code>mp</code>

Description This specifies the variables that are initially bound in a Lisp process when that process is created. This variable is an association list of symbols and initial value forms. The initial value forms are processed by a simple evaluation that handles symbols and function call forms, but not special operators. As a special case, if the value form is the same as the symbol and that symbol is unbound, then the symbol will be unbound in the new process.

Examples This example shows a typical use with a bound symbol:

```
(defvar *binding-1* 10)

(let ((mp:*process-initial-bindings*
       (cons '(*binding-1* . 20)
             mp:*process-initial-bindings*)))
  (mp:process-run-function
   "binding-1"
   '()
   #'(lambda (stream)
        (format stream "~&Binding 1 is ~S.~%" *binding-
1*))
   *standard-output*)
  (sleep 1))
=>
Binding 1 is 20.
```

This example shows the special case with an unbound symbol:

```
(defvar *binding-2*)
```

```

(let ((mp:*process-initial-bindings*
      (cons '(*binding-2* . *binding-2*)
            mp:*process-initial-bindings*)))
  (flet ((check-binding-2 ()
          (mp:process-run-function
           "binding-2"
           '()
           #'(lambda (stream)
               (if (boundp '*binding-2*)
                   (format stream "~&Binding 2 is ~S.~%" 
*binding-2*)
                   (format stream "~&Binding 2 is
unbound.~%")))
               *standard-output*)
           (sleep 1)))
        (check-binding-2)
        (let ((*binding-2* 123))
          (check-binding-2))))
  =>
Binding 2 is unbound.
Binding 2 is 123.

```

process-interrupt *Function*

Summary	Interrupts a process.	
Package	<code>mp</code>	
Signature	<code>process-interrupt <i>process function &rest arguments</i> =></code>	
Arguments	<p><i>process</i> A process.</p> <p><i>function</i> A function to apply on resuming <i>process</i>.</p> <p><i>arguments</i> Arguments to supply to <i>function</i>.</p>	
Values	None.	
Description	Causes the Lisp process <i>process</i> to apply <i>function</i> to <i>arguments</i> when it is next resumed. Afterwards the process resumes its normal execution. A waiting process is temporarily woken up.	

process-kill *Function*

Summary	Kills the specified Lisp process.	
Package	<code>mp</code>	
Signature	<code>process-kill process =></code>	
Arguments	<code>process</code>	A process.
Values	None.	
Description	The function <code>process-kill</code> kills the specified Lisp process.	
See also	<code>ensure-process-cleanup</code>	

process-lock *Function*

Summary	Claims the lock for the current process.	
Package	<code>mp</code>	
Signature	<code>process-lock lock &optional whostate timeout => bool</code>	
Arguments	<code>lock</code>	A lock object (see <code>make-lock</code>).
	<code>whostate</code>	The status of the current Lisp process, before <code>process-lock</code> returns, that is, the status while the current process is waiting to time-out. This can be seen in the Process Browser.
	<code>timeout</code>	A timeout interval, in seconds. If this is not given, <code>process-lock</code> waits until the lock can be set by the current Lisp process. A process can set a lock more than once.
Values	<code>result</code>	A boolean.

Description	<code>process-lock</code> attempts to lock <i>lock</i> and returns <i>t</i> if successful, or <code>nil</code> if timed out. If <i>lock</i> is already locked and the owner of the lock is the value of <code>*current-process*</code> , then <i>lock</i> remains locked and an internal count is incremented. The Lisp process sleeps until the lock is claimed or the timeout period expires.
	<i>result</i> is <i>t</i> if <i>lock</i> was successfully locked, and <code>nil</code> otherwise.
Example	(<code>process-lock *my-lock* "waiting to lock" 10</code>)
See also	<code>create-simple-process</code> <code>make-lock</code> <code>process-unlock</code> <code>schedule-timer</code> <code>with-lock</code>

	<i>Function</i>
Summary	Accesses the mailbox associated with a process.
Package	<code>mp</code>
Signature	<code>process-mailbox process => result</code> <code>(setf process-mailbox) result process => result</code>
Arguments	<code>process</code> A process.
Values	<code>result</code> A mailbox object, or <code>nil</code> .
Description	<code>process-mailbox</code> is an accessor function which returns or sets the mailbox associated with <i>process</i> .
Example	(<code>setf (mp:process-mailbox mp:*current-process*)</code> <code>(mp:make-mailbox)</code>)

process-name	<i>Function</i>
Summary	Returns the name of a specified process.
Package	<code>mp</code>
Signature	<code>process-name process => name</code>
Arguments	<code>process</code> A process.
Values	<code>name</code> The name of the process specified by <code>process</code> .
Description	Returns the name of the specified Lisp process.
process-p	<i>Function</i>
Summary	A predicate to indentify non-simple processes
Package	<code>mp</code>
Signature	<code>process-p object => bool</code>
Arguments	<code>object</code> Any object
Values	<code>bool</code> A generalized boolean.
Description	The <code>process-p</code> function returns <code>t</code> if <code>object</code> is a non-simple process, and <code>nil</code> otherwise.
process-plist	<i>Function</i>
Summary	Returns the plist associated with a process.
Package	<code>mp</code>

Signature	<code>process-plist process => plist</code>	
Arguments	<code>process</code>	A process
Values	<code>plist</code>	A plist
Description	The <code>process-plist</code> function returns the plist associated with <code>process</code> .	
Example	<pre>(setf (getf (mp:process-plist mp:*current-process*) 'foo) 'foo-value) => FOO-VALUE (getf (mp:process-plist mp:*current-process*) 'foo) => FOO-VALUE</pre>	

	<i>Function</i>
Summary	Returns the numerical priority of the Lisp process.
Package	<code>mp</code>
Signature	<code>process-priority process => priority</code>
Arguments	<code>process</code> A process.
Values	<code>priority</code> A fixnum, the priority of <code>process</code> .
Description	Returns the numerical priority of the Lisp process. This can be modified by calling <code>mp:change-process-priority</code> .
Example	<code>CL-USER 17 > (mp:process-priority mp:*current-process*)</code> <code>600000</code>
See also	<code>change-process-priority</code>

		<i>Function</i>
Summary		Resets a process by discarding its current state.
Package		<code>mp</code>
Signature		<code>process-reset process =></code>
Arguments	<i>process</i>	A process.
Values		None.
Description		<code>process-reset</code> interrupts the execution of <i>process</i> and “throws away” its current state. Upon resuming execution, the process calls its function with its initial argument and priority.

		<i>Function</i>
Summary		Create a new process, passing it a function to run.
Package		<code>mp</code>
Signature		<code>process-run-function name keywords function &rest arguments => process</code>
Arguments	<i>name</i>	A name for the new process.
	<i>keywords</i>	Keywords specifying properties of the new process.
	<i>function</i>	A function to apply.
	<i>arguments</i>	Arguments to pass to <i>function</i> .
Values	<i>process</i>	The newly created process.

Description	This function creates a new Lisp process with name <i>name</i> . Other properties of <i>process</i> may be specified in keyword/value pairs in <i>keywords</i> :
:priority	A fixnum representing the priority for the process. If :priority is not supplied, the process priority becomes the value of the variable *default-process-priority*.
:mailbox	A mailbox object or nil, used to initialize the process-mailbox of <i>process</i> .
	The new process is preset to apply <i>function</i> to <i>arguments</i> and runs in parallel, while process-run-function returns immediately.
Example	<pre>CL-USER 253 > (defvar *stream* *standard-output*) *STREAM* CL-USER 254 > (mp:process-run-function "My process" '(:priority 42) #'(lambda (x) (loop for i below x do (and (print i *stream*) (sleep 1))) finally (print (mp:process-priority mp:*current-process*) *stream*))) 3) #<MP:PROCESS Name "My process" Priority 850000 State "Running"> 0 1 2 42 CL-USER 255 ></pre>
See also	create-simple-process *default-process-priority*

		<i>Function</i>
Summary		Returns the reasons that a specified process is running.
Package		<code>mp</code>
Signature		<code>process-run-reasons <i>process</i> => <i>reasons</i></code> <code>(setf process-run-reasons) <i>process</i> <i>reasons</i> => <i>reasons</i></code>
Arguments	<i>process</i>	A process.
Values	<i>reasons</i>	A list of run reasons.
Description		The function <code>process-run-reasons</code> returns a list of reasons for the specified Lisp process running. These can be changed using <code>setf</code> . A process is only active if it has at least one run reason and no arrest reasons.
See also		<code>process-arrest-reasons</code> <code>process-run-function</code> <code>process-whostate</code>

		<i>Function</i>
Summary		Returns the current run time for a process.
Package		<code>mp</code>
Signature		<code>process-run-time <i>process</i> => <i>time</i></code>
Arguments	<i>process</i>	A process.
Values	<i>time</i>	A positive integer or <code>nil</code> .

Description	The function <code>process-run-time</code> returns the current run time for <i>process</i> in internal time units. If the value cannot be determined (currently this is only on FreeBSD), then the return value is <code>nil</code> . Note: The value returned by <code>get-internal-run-time</code> is similar, but on some operating systems it is the total time for all Lisp processes in the image.
See also	<code>process-idle-time</code>

process-send *Function*

Summary	Sends an object to the mailbox of a given process.						
Package	<code>mp</code>						
Signature	<code>process-send process object &key change-priority =></code>						
Arguments	<table> <tr> <td><i>process</i></td> <td>A process</td> </tr> <tr> <td><i>object</i></td> <td>An object</td> </tr> <tr> <td><i>change-priority</i></td> <td>A fixnum, <code>nil</code>, <code>t</code>, or <code>:default</code></td> </tr> </table>	<i>process</i>	A process	<i>object</i>	An object	<i>change-priority</i>	A fixnum, <code>nil</code> , <code>t</code> , or <code>:default</code>
<i>process</i>	A process						
<i>object</i>	An object						
<i>change-priority</i>	A fixnum, <code>nil</code> , <code>t</code> , or <code>:default</code>						
Values	None.						
Description	<p>The <code>process-send</code> function queues <i>object</i> in the mailbox of the given process.</p> <p><i>object</i> should be a list of the form <code>(function . arguments)</code>. This causes <i>function</i> to be applied to <i>arguments</i> in the process <i>process</i>. (The results of <i>function</i> are discarded.)</p> <p>If <i>change-priority</i>, which has a default value of <code>:default</code>, is non-<code>nil</code>, it controls how the priority of that process is calculated as follows:</p>						

- `fixnum` — use the value of *change-priority* as the new priority.
- `t` — set the priority to the interactive priority.
- `:default` — set the priority to the normal running priority.

See also `mailbox-reader-process`
`mailbox-send`

process-stop *Function*

Summary Stops a process.

Package `mp`

Signature `process-stop process`

Arguments `process` A `mp:process` object.

Description The function `process-stop` stops the process *process*.

process must be a full process (that is, not one created by `create-simple-process`).

`process-stop` causes *process* to stop until some other process explicitly wakes it up. If it is called on the current process, the current process stops during the call, and returns from `process-stop` after the process gets woken up. If *process* is not the current process, *process* stops before `process-stop` returns.

You can wake up a stopped process (that is, make it runnable) by calling `process-kill`, `process-unstop` or `process-continue`.

`process-interrupt` does not wake up a stopped process.

There is a discussion of a typical use of `process-stop` in the section "Stopping and unstopping processes" in the *LispWorks User Guide*.

`process-stop` does not return any useful value.

See also	<code>process-arrest-reasons</code> <code>process-stopped-p</code> <code>process-unstop</code>
----------	--

process-stopped-p *Function*

Summary The predicate for stopped processes.

Package `mp`

Signature `process-stopped-p process => result`

Arguments `process` A `mp:process` object.

Values `result` A boolean.

Description The function `process-stopped-p` queries whether the process `process` is stopped or not.

If `process` stopped because it called `process-stop` on itself, then `process-stopped-p result` is `t` only if `process-stop` really stopped it (that is, a later call to `process-unstop` will unstopp the process).

See also	<code>process-stop</code> <code>process-unstop</code>
----------	--

process-unlock *Function*

Summary Relinquishes a lock held by the current process.

Package	<code>mp</code>	
Signature	<code>process-unlock lock &optional errorp => result</code>	
Arguments	<code>lock</code>	The lock to be relinquished.
	<code>errorp</code>	When this is <code>t</code> , an error is signalled if <code>*current-process*</code> is not the owner of the lock. The default is <code>t</code> .
Values	<code>result</code>	A boolean.
Description	<p>Attempts to release a lock. If the lock is owned by <code>*current-process*</code>, <code>process-unlock</code> decrements an internal count. If this lock count is then zero, the lock is released. Note that <code>process-unlock</code> relates only on Lisp processes.</p> <p><code>result</code> is <code>t</code> if the lock was released, and <code>nil</code> otherwise.</p>	
See also	<code>create-simple-process</code> <code>make-lock</code> <code>process-lock</code> <code>schedule-timer</code> <code>with-lock</code>	

process-unstop		<i>Function</i>
Summary	Unstops a process.	
Package	<code>mp</code>	
Signature	<code>process-unstop process => result</code>	
Arguments	<code>process</code>	A <code>mp:process</code> object.
Values	<code>result</code>	A boolean.

Description	<p>The function <code>process-unstop</code> unstops the process <i>process</i> if it is stopped.</p> <p><i>process</i> must be a full process (that is, not one created by <code>create-simple-process</code>).</p> <p>If <i>process</i> was stopped (by <code>process-stop</code>), it is unstopped and resumes execution.</p> <p><i>result</i> is <code>t</code> if <i>process</i> was stopped, and <code>nil</code> otherwise.</p> <p>There is a discussion of a typical use of <code>process-unstop</code> in the section "Stopping and unstopping processes" in the <i>Lisp-Works User Guide</i>.</p>
See also	<p><code>process-stop</code></p> <p><code>process-stopped-p</code></p>

	<i>Function</i>
process-wait	
Summary	Suspends the current process until a condition is true.
Package	<code>mp</code>
Signature	<code>process-wait wait-reason wait-function &rest wait-arguments =></code>
Arguments	<p><i>wait-reason</i> A string describing the reason that the process is waiting.</p> <p><i>wait-function</i> A function designator.</p> <p><i>wait-arguments</i> The arguments that <i>wait-function</i> is applied to.</p>
Values	None.
Description	The function <code>process-wait</code> suspends the current Lisp process until the predicate <i>wait-function</i> applied to <i>wait-arguments</i> returns <code>t</code> . This is tested periodically.

wait-function must not do a non-local exit. *wait-function* should not have side effects and, since it is called frequently, it should be efficient.

wait-reason allows you to find out why a process is waiting via the function `process-whostate`.

See also `process-wait-with-timeout`
 `process-whostate`

process-wait-for-event *Function*

Summary	Waits for an event in a "windowing friendly" way.	
Package	<code>mp</code>	
Signature	<code>process-wait-for-event &key wait-reason wait-function process-other-messages-p no-hang-p stop-at-user-operation-p => event</code>	
Arguments	<p><i>wait-reason</i> A string or <code>nil</code>. <i>wait-function</i> A function designator. <i>process-other-messages-p</i> A generalized boolean. <i>no-hang-p</i> A generalized boolean. <i>stop-at-user-operation-p</i> A generalized boolean.</p>	
Values	<i>result</i>	An event or <code>nil</code> .
Description	<p>The function <code>process-wait-for-event</code> calls <code>mailbox-wait-for-event</code> on the mailbox of the current process, after ensuring that the current process has a mailbox.</p> <p>The arguments and value are interpreted as for <code>mailbox-wait-for-event</code>.</p>	

See also [mailbox-wait-for-event](#)

process-wait-function *Function*

Summary Returns a function that determines whether a process should continue to wait.

Package [mp](#)

Signature `process-wait-function process => wait-function`

Arguments `process` A process.

Values `wait-function` A function designator.

Description The function `process-wait-function` returns the function that determines whether the Lisp process waits. The system periodically calls `wait-function` to decide whether to wake the process up.

`wait-function` is applied to `wait-arguments`, where both `wait-function` and `wait-arguments` were passed to `process-wait`.

See also [process-wait](#)

process-wait-with-timeout *Function*

Summary Suspend the current process until certain conditions are true, or until a timeout expires.

Package [mp](#)

Signature	<code>process-wait-with-timeout <i>wait-reason</i> <i>timeout</i> &optional <i>wait-function</i> &rest <i>wait-arguments</i> => <i>bool</i></code>	
Arguments	<i>wait-reason</i>	A string describing the reason that the process is waiting.
	<i>timeout</i>	A timeout, in seconds.
	<i>wait-function</i>	A function to test.
	<i>wait-arguments</i>	The arguments to apply to <i>wait-function</i> .
Values	<i>bool</i>	A boolean.
Description	<p>This function uses <code>process-wait</code> to suspend the current Lisp process until the predicate <i>wait-function</i> applied to <i>wait-arguments</i> returns <code>t</code>, or until <i>timeout</i> seconds have passed.</p> <p><i>bool</i> is <code>nil</code> if the timeout occurred before <i>wait-function</i> returned true. <i>bool</i> is true otherwise.</p>	
See also	<code>process-wait</code>	

process-whostate

Function

Summary	Returns the state of a process.	
Signature	<code>process-whostate <i>process</i> => <i>result</i></code>	
Package	<code>mp</code>	
Arguments	<i>process</i>	A process.
Values	<i>reason</i>	A string.
Description	<p>The function <code>process-whostate</code> returns a string describing the state of the process.</p> <p>Depending on the state of <i>process</i>, <i>reason</i> can be:</p>	

- "Dead"
- "Stopped",
- "Sleeping"
- "Running"
- "Running (preempted)"

reason can also be the *wait-reason* of the process, as passed to `wait-processing-events`, `process-wait` and so on.

reason can also be a string containing the *run-reasons*, as set by `(setf process-run-reasons)`.

See also `wait-processing-events`
 `process-wait`
 `process-run-reasons`

	<i>Function</i>
ps	
Summary	Prints the processes in the system
Package	<code>mp</code>
Signature	<code>ps =></code>
Arguments	None.
Values	None.
Description	Prints a list of the processes in the system, ordered by priority. (This function is analogous to the UNIX command <code>ps</code> .)

schedule-timer	<i>Function</i>						
Summary	Schedules a timer to expire at a given time after the start of the program.						
Signature	<code>schedule-timer <i>timer absolute-expiration-time</i> &optional <i>repeat-time</i> => <i>timer</i></code>						
Package	<code>mp</code>						
Arguments	<table> <tr> <td><i>timer</i></td> <td>A timer</td> </tr> <tr> <td><i>absolute-expiration-time</i></td> <td>A non-negative real</td> </tr> <tr> <td><i>repeat-time</i></td> <td>A non-negative real</td> </tr> </table>	<i>timer</i>	A timer	<i>absolute-expiration-time</i>	A non-negative real	<i>repeat-time</i>	A non-negative real
<i>timer</i>	A timer						
<i>absolute-expiration-time</i>	A non-negative real						
<i>repeat-time</i>	A non-negative real						
Values	<i>timer</i> A timer						
Description	<p>The schedule-timer function schedules a timer to expire at a given time after the start of the program. The <i>timer</i> argument is a timer, returned by <code>make-timer</code> or <code>make-named-timer</code>. The <i>absolute-expiration-time</i> argument is a non-negative real number of seconds since the start of the program at which the timer is to expire. If <i>repeat-time</i> is specified, it is a non-negative real number of seconds that specifies a repeat interval. Each time the timer expires, it is rescheduled to expire after this repeat interval.</p> <p>If the timer is already scheduled to expire at the time this function is called, it is rescheduled to expire at the time specified by the <i>absolute-expiration-time</i> argument. If that argument is <code>nil</code>, the timer is not rescheduled, but the repeat interval is set to the interval specified by the <i>repeat-time</i> argument.</p> <p>The function schedule-timer-relative schedules a timer to expire at a time relative to the call to that function.</p>						

Example The following example schedules a timer to expire 15 minutes after the start of the program and every 5 minutes thereafter.

```
(setq timer
      (mp:make-timer 'print 10 *standard-output*))

#<Time Event : PRINT>

(mp:schedule-timer timer 900 300)

#<Time Event : PRINT>
```

See also

- `make-named-timer`
- `make-timer`
- `schedule-timer-milliseconds`
- `schedule-timer-relative`
- `schedule-timer-relative-milliseconds`
- `timer-expired-p`
- `timer-name`
- `unschedule-timer`

		<i>Function</i>
Summary	Schedules a timer to expire after a given amount of time.	
Signature		<code>schedule-timer-milliseconds <i>timer absolute-expiration-time</i>&optional <i>repeat-time</i> => <i>timer</i></code>
Package		<code>mp</code>
Arguments	<code>timer</code> <i>A timer</i> <code>absolute-expiration-time</code> <i>A non-negative real</i>	
	<code>repeat-time</code> <i>A non-negative real</i>	
Values	<code>timer</code> <i>A timer</i>	

Description	<p>The <code>schedule-timer-milliseconds</code> function schedules a timer to expire at a given time after the start of the program. The <code>timer</code> argument is a timer returned by <code>make-timer</code> or <code>make-named-timer</code>. The <code>absolute-expiration-time</code> argument is a non-negative real number of milliseconds since the start of the program at which the timer is to expire. If <code>repeat-time</code> is specified, it is a non-negative real number of milliseconds that specifies a repeat interval. Each time the timer expires, it is rescheduled to expire after this repeat interval.</p> <p>If the timer is already scheduled to expire at the time this function is called, it is rescheduled to expire at the time specified by the <code>absolute-expiration-time</code> argument. If that argument is <code>nil</code>, the timer is not rescheduled, but the repeat interval is set to the interval specified by the <code>repeat-time</code> argument.</p> <p>The function <code>schedule-timer-relative-milliseconds</code> schedules a timer to expire at a time relative to the call to that function.</p>
Example	<p>The following example schedules a timer to expire 15 minutes after the start of the program and every 5 minutes thereafter.</p> <pre>(setq timer (mp:make-timer 'print 10 *standard-output*)) #<Time Event : PRINT> (mp:schedule-timer-milliseconds timer 900000 300000) #<Time Event : PRINT></pre>
See also	<code>make-named-timer</code> <code>make-timer</code> <code>schedule-timer</code> <code>schedule-timer-relative</code> <code>schedule-timer-relative-milliseconds</code> <code>timer-expired-p</code> <code>timer-name</code> <code>unschedule-timer</code>

	<i>Function</i>						
schedule-timer-relative							
Summary	Schedules a timer to expire at a given time after this function is called.						
Signature	<code>schedule-timer-relative <i>timer</i> <i>relative-expiration-time</i> &optional <i>repeat-time</i> => <i>timer</i></code>						
Package	<code>mp</code>						
Arguments	<table border="0"> <tr> <td><i>timer</i></td><td>A timer</td></tr> <tr> <td><i>relative-expiration-time</i></td><td>A non-negative real</td></tr> <tr> <td><i>repeat-time</i></td><td>A non-negative real</td></tr> </table>	<i>timer</i>	A timer	<i>relative-expiration-time</i>	A non-negative real	<i>repeat-time</i>	A non-negative real
<i>timer</i>	A timer						
<i>relative-expiration-time</i>	A non-negative real						
<i>repeat-time</i>	A non-negative real						
Values	<i>timer</i> A timer						
Description	<p>The <code>schedule-timer-relative</code> function schedules a timer to expire at a given time after the call to the function. The <i>timer</i> argument is a timer returned by <code>make-timer</code> or <code>make-named-timer</code>. The <i>relative-expiration-time</i> argument is a non-negative real number of seconds after the call to the function at which the timer is to expire. If <i>repeat-time</i> is specified, it is a non-negative real number of seconds that specifies a repeat interval. Each time the timer expires, it is rescheduled to expire after this repeat interval.</p> <p>If the timer is already scheduled to expire at the time this function is called, it is rescheduled to expire at the time specified by the <i>relative-expiration-time</i> argument. If that argument is <code>nil</code>, the timer is not rescheduled, but the repeat interval is set to the interval specified by the <i>repeat-time</i> argument.</p> <p>The function <code>schedule-timer</code> schedules a timer to expire at a time relative to the start of the program.</p>						

Example The following example schedules a timer to expire 5 seconds after the call to schedule-timer-relative and every 5 seconds thereafter.

```
(setq timer
      (mp:make-timer 'print 10 *standard-output*))

#<Time Event : PRINT>

(mp:schedule-timer-relative timer 5 5)

#<Time Event : PRINT>
```

See also [make-named-timer](#)
[make-timer](#)
[schedule-timer](#)
[schedule-timer-milliseconds](#)
[schedule-timer-relative-milliseconds](#)
[timer-expired-p](#)
[timer-name](#)
[unschedule-timer](#)

schedule-timer-relative-milliseconds *Function*

Summary Schedules a timer to expire at a given time after this function is called.

Signature `schedule-timer-relative-milliseconds timer
relative-expiration-time &optional repeat-time => timer`

Package `mp`

Arguments `timer` A timer
`relative-expiration-time`
A non-negative real
`repeat-time` A non-negative real

Values `timer` A timer

Description	<p>The <code>schedule-timer-relative-milliseconds</code> function schedules a timer to expire at a given time after the call to the function. The <code>timer</code> argument is a timer returned by <code>make-timer</code> or <code>make-named-timer</code>. The <code>relative-expiration-time</code> argument is a non-negative real number of milliseconds after the call to the function at which the timer is to expire. If <code>repeat-time</code> is specified, it is a non-negative real number of milliseconds that specifies a repeat interval. Each time the timer expires, it is rescheduled to expire after this repeat interval.</p> <p>If the timer is already scheduled to expire at the time this function is called, it is rescheduled to expire at the time specified by the <code>relative-expiration-time</code> argument. If that argument is <code>nil</code>, the timer is not rescheduled, but the repeat interval is set to the interval specified by the <code>repeat-time</code> argument.</p> <p>The function <code>schedule-timer-milliseconds</code> schedules a timer to expire at a time relative to the start of the program.</p>
Example	<p>The following example schedules a timer to expire 5 seconds after the call to <code>schedule-timer-relative-milliseconds</code> and every 5 seconds thereafter.</p> <pre>(setq timer (mp:make-timer 'print 10 *standard-output*)) #<Time Event : PRINT> (mp:schedule-timer-relative-milliseconds timer 5000 5000) #<Time Event : PRINT></pre>

See also

`make-named-timer`
`make-timer`
`schedule-timer`
`schedule-timer-milliseconds`
`schedule-timer-relative`
`timer-expired-p`
`timer-name`
`unschedule-timer`

simple-process-p *Function*

Summary	A predicate identifying simple processes.	
Package	mp	
Signature	<code>simple-process-p object=> bool</code>	
Arguments	<i>object</i>	An object
Values	<i>bool</i>	A generalized boolean
Description	The <code>simple-process-p</code> function returns <code>t</code> if <i>object</i> is a simple process and <code>nil</code> otherwise.	
See also	<code>create-simple-process</code>	

symeval-in-process *Function*

Summary	Reads the value of symbol which is dynamically bound in a given process.	
Package	mp	
Signature	<code>symeval-in-process symbol process => value, flag</code> <code>(setf symeval-in-process) value symbol process => value</code>	
Arguments	<i>symbol</i>	A symbol
	<i>process</i>	A process
Values	<i>value</i>	A Lisp object
	<i>flag</i>	One of <code>t</code> , <code>nil</code> or the keyword <code>:unbound</code>

Description	The function <code>symeval-in-process</code> reads the value of the symbol <i>symbol</i> in the process <i>process</i> if it is bound dynamically. The global value of <i>symbol</i> is never returned. If <i>symbol</i> is not bound in <i>process</i> , then <i>value</i> and <i>flag</i> are both <code>nil</code> . If <i>symbol</i> is bound in <i>process</i> but <code>makunbound</code> has been called within the dynamic scope of the binding, <i>value</i> is <code>nil</code> and <i>flag</i> is <code>:unbound</code> . Otherwise, <i>value</i> is the value of <i>symbol</i> and <i>flag</i> is <code>t</code> .
-------------	---

In addition, the form

```
(setf (symeval-in-process symbol process) value)
```

sets the value of *symbol* to *value* in *process*. It is an error if *process* has no binding for *symbol*. This `setf` form returns *value* as specified by Common Lisp.

	<i>Function</i>				
Summary	Returns <code>t</code> if a given timer has expired or is about to expire.				
Signature	<code>timer-expired-p <i>timer</i> &optional <i>delta</i> => <i>bool</i></code>				
Package	<code>mp</code>				
Arguments	<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 25%;"><i>timer</i></td><td style="width: 75%;">A timer</td></tr> <tr> <td><i>delta</i></td><td>A non-negative real</td></tr> </table>	<i>timer</i>	A timer	<i>delta</i>	A non-negative real
<i>timer</i>	A timer				
<i>delta</i>	A non-negative real				
Values	<i>bool</i> A boolean				
Description	The <code>timer-expired-p</code> function returns <code>t</code> if the specified timer is not scheduled to expire or is scheduled to expire within the number of seconds specified by the <i>delta</i> argument after the call to <code>timer-expired-p</code> . Otherwise, the function returns <code>nil</code> .				

The *timer* argument is a timer, returned by `make-timer` or `make-named-timer`. The *delta* argument, if supplied, is a non-negative real number of seconds.

Example

```
(setq timer
      (mp:make-timer 'print 10 *standard-output*))

#<Time Event : PRINT>

(mp:schedule-timer-relative timer 5)

#<Time Event : PRINT>

(mp:timer-expired-p timer)

NIL
```

See also

```
make-named-timer
make-timer
schedule-timer
schedule-timer-milliseconds
schedule-timer-relative
timer-name
unschedule-timer
```

timer-name

Function

Summary Returns the name of a specified timer.

Signature `timer-name timer => name`

Signature `(setf timer-name) name timer => name`

Package `mp`

Arguments *timer* A timer

Values *name* A string

Description	The <code>timer-name</code> function returns the name of the specified <i>timer</i> . The <i>timer</i> argument is a timer returned by <code>make-timer</code> or <code>make-named-timer</code> . If the timer has no name, <code>timer-name</code> returns <code>nil</code> .
	The name of a timer created by either <code>make-timer</code> or <code>make-named-timer</code> can be set by means of the following syntax:
Example	<pre>(setf (mp:timer-name timer) name) (setq timer (mp:make-timer 'print 10 *standard-output*)) #<Time Event : PRINT> (mp:timer-name timer) NIL (setf (mp:timer-name timer) 'timer-1) TIMER-1 (mp:timer-name timer) TIMER-1</pre>

See also

- `make-named-timer`
- `make-timer`
- `schedule-timer`
- `schedule-timer-milliseconds`
- `schedule-timer-relative`
- `timer-expired-p`
- `unschedule-timer`

	Function
Summary	Removes a file descriptor from the set of interesting input file descriptors.
Package	<code>mp</code>

Signature	<code>unnotice-fd <i>fd</i></code>	
Arguments	<code><i>fd</i></code>	A file descriptor
Values	None.	
Description	<p>The <code>unnotice-fd</code> function removes <i>fd</i> from the set of fds that cause LispWorks to wake up when they contain input.</p> <p>This function is not implemented on Microsoft Windows.</p>	
See also	<code>notice-fd</code>	

unschedule-timer *Function*

Summary	Unschedules a scheduled timer	
Signature	<code>unschedule-timer <i>timer</i> => <i>result</i></code>	
Package	<code>mp</code>	
Arguments	<code><i>timer</i></code>	A timer
Values	<code><i>result</i></code>	A timer or <code>nil</code>
Description	<p>If the specified timer has been scheduled to expire at a time after the call to <code>unschedule-timer</code>, this function unschedules the timer and returns the timer. Otherwise, the function returns <code>nil</code>.</p> <p>The argument is a timer, returned by <code>make-timer</code> OR <code>make-named-timer</code>.</p>	
Example	<pre>(setq timer (mp:make-timer 'print 10 *standard-output*)) #<Time Event : PRINT> (mp:schedule-timer-relative timer 60)</pre>	

```
#<Time Event : PRINT>
(mp:unschedule-timer timer)

#<Time Event : PRINT>
(mp:timer-expired-p timer)

T
```

See also

[make-named-timer](#)
[make-timer](#)
[schedule-timer](#)
[schedule-timer-milliseconds](#)
[schedule-timer-relative](#)
[timer-expired-p](#)
[timer-name](#)

wait-processing-events*Function*

Summary	Waits processing events.	
Signature	wait-processing-events <i>timeout</i> & key <i>wait-reason</i> <i>wait-function</i> <i>wait-args</i> => <i>result</i>	
Package	mp	
Arguments	<i>timeout</i>	A number.
	<i>wait-reason</i>	A string.
	<i>wait-function</i>	A function designator.
	<i>wait-args</i>	A list.
Values	<i>result</i>	t or nil
Description	The function wait-processing-events does not return until one of two conditions is met: <ul style="list-style-type: none"> • <i>timeout</i> seconds have passed. In this case, <i>result</i> is nil. 	

- *wait-function* returns a true value. In this case, *result* is `t`.
wait-reason provides the value returned by `process-whostate` when called on the current process.
wait-function is called periodically with arguments *wait-args*. *wait-function* may be called many times and in several places. Therefore *wait-function* should be fast and make no assumptions about its dynamic context.

`wait-processing-events` processes all events sent to the current process, including system events such as window messages on Microsoft Windows, and objects sent by other processes via `process-send`. In the latter case, the objects must be lists of the form (*function* . *arguments*), which cause *function* to be applied to *arguments* (the values are discarded).

`wait-processing-events` is a useful alternative to `sleep` in a situation where you want to process events to see window updates and so on.

See also	<code>process-send</code> <code>process-whostate</code>
----------	--

		<i>Macro</i>
Summary	Executes a body of code while holding a lock.	
Package	<code>mp</code>	
Signature	<code>with-lock (lock &rest lock-args) &body body => result</code>	
Arguments	<code>lock</code> <code>lock-args</code>	The lock. These are the optional arguments used by <code>process-lock</code> : <i>whostate</i> (the status of the process while the lock is locked, as seen in the Process Browser) and <i>timeout</i> (a timeout period, in seconds).

	<i>body</i>	The forms to execute.
Values	<i>result</i>	The result of executing <i>body</i> .
Description		<code>with-lock</code> executes <i>body</i> while holding the lock, and unlocks the lock when <i>body</i> exits. This is the recommended way of using locks. The value of <i>body</i> is returned normally. <i>body</i> is not executed if the lock could not be claimed, in which case, <code>with-lock</code> returns <code>nil</code> .
See also		<code>create-simple-process</code> <code>make-lock</code> <code>process-lock</code> <code>process-unlock</code> <code>schedule-timer</code>

without-interrupts		<i>Macro</i>
Summary		Causes any interrupts that occur during the execution of a body of code to be queued.
Package		<code>mp</code>
Signature		<code>without-interrupts &rest body => result</code>
Arguments	<i>body</i>	The forms to execute while interrupts are queued.
Values	<i>result</i>	The result of executing <i>body</i> .
Description		While <i>body</i> is executing, all interrupts (for example, preemption, keyboard break etc.) are queued. They are executed when <i>body</i> exits.

Example To ensure that the seconds and milliseconds slots are always consistent, you can use `mp:without-interrupts` within the function which sets them.

```
(defstruct elapsed-time
  seconds
  milliseconds)

(defun update-elapsed-time-atomically
  (elapsed-time seconds milliseconds)
  (mp:without-interrupts
    (setf (elapsed-time-seconds elapsed-time) seconds
          (elapsed-time-milliseconds elapsed-time)
          milliseconds)))
```

See also `without-preemption`

without-preemption *Macro*

Summary Identifies forms which should not be preempted during execution.

Package `mp`

Signature `without-preemption &rest body => result`

Arguments `body` The forms to be evaluated atomically.

Values `result` The result of executing `body`.

Description Identifies forms which should not be preempted during execution.

yield *Function*

Summary Allows preemption to happen in low safety code.

Package `mp`

Signature	<code>yield</code>
Arguments	None.
Values	None.
Description	Normally code compiled at safety 0 cannot be preempted because the necessary checks are omitted. This can be overcome by calling <code>yield</code> at regular intervals. Usually there is no need to call this if you use functions from the <code>common-lisp</code> package because these are not compiled at safety 0, but for example if you find that preemption is not working in a loop with no function calls, <code>yield</code> can be useful. Note that <code>process-allow-scheduling</code> also allows preemption, but also checks the wait functions of other processes.
See also	<code>process-allow-scheduling</code>

12

The PARSERGEN Package

This chapter describes symbols available in the **PARSERGEN** package, the Lisp-Works parser generator. You should use this chapter in conjunction with the relevant chapter in the *LispWorks User Guide*.

	<i>Macro</i>
Summary	Creates a parsing function of the given name for the grammar defined.
Package	<code>parsergen</code>
Signature	<code>defparser name {rule}* => parsing-function</code> <code>rule ::= normal-rule error-rule</code> <code>normal-rule ::= ((non-terminal {grammar-symbol}*) {form}*)</code> <code>error-rule ::= ((non-terminal :error) {form}*)</code>
Arguments	<code>name</code> The name of the parser. The rules define the productions of the grammar and the associated forms define the semantic actions for the rules.
Values	<code>parsing-function</code> The symbol name of the parsing function.

Description **defparser** creates a parsing function of the given name for the grammar defined. The parsing function is defined as if by:

```
(defun <name> (&optional (symbol-to-string  
 #'identify))
```

The *lexer* parameter is a function of no arguments that returns two values: the next grammar token on the input and the associated semantic value.

The optional symbol-to-string function can be used to define a printed representation of the grammar tokens. The function should take a grammar symbol as its single argument and returns an object to be used as a print representation for the grammar token.

For a full description and examples, see the *LispWorks User Guide*.

13

The SERIAL-PORT Package

This chapter describes the symbols available in the **SERIAL-PORT** package.

The Serial Port functionality is loaded into LispWorks by evaluating

```
(require "serial-port")
```

Note: this chapter applies only to LispWorks for Windows, and not the UNIX, Linux or Mac OS X platforms.

		<i>Function</i>
Summary		Attempts to open the named serial port and return a serial-port object.
Package		serial-port
Signature		open-serial-port name &rest args &key baud-rate data-bits stop-bits parity cts-flow-p dsr-flow-p dtr rts read-interval-timeout read-total-base-timeout read-total-byte-timeout write-total-base-timeout write-total-byte-timeout => serial-port
Arguments	<i>name</i>	A string naming a serial port.

	args	See in the Description below for details of the remaining arguments.
Values	serial-port	A serial-port object.
Description	The function <code>open-serial-port</code> attempts to open the serial port <i>name</i> and return a serial-port object. <i>name</i> is passed directly to <code>Createfile()</code> . For ports COM <i>n</i> where <i>n</i> > 9, you must take care to pass the real port name expected by Windows. At the time of writing this issue is documented at http://support.microsoft.com/kb/115831 .	
	If any of <i>baud-rate</i> , <i>data-bits</i> , <i>stop-bits</i> and <i>parity</i> are supplied then the corresponding serial port settings are changed. The values of <i>baud-rate</i> and <i>data-bits</i> should each be an appropriate integer. The value of <i>stop-bits</i> should be 1, 1.5 or 2. The value of <i>parity</i> should be one of the keywords <code>:even</code> , <code>:mark</code> , <code>:none</code> , <code>:odd</code> OR <code>:space</code> .	
	The arguments <i>cts-flow-p</i> and <i>dsr-flow-p</i> control whether write operations respond to CTS and DSR flow control. A non- <code>nil</code> value means that the corresponding flow control is used.	
	The arguments <i>dtr</i> and <i>rts</i> control whether read operations generate DTR or RTS flow control. If the value is <code>:handshake</code> then the corresponding flow control signal is generated automatically. If the value is <code>nil</code> or <code>t</code> then the initial state of the flow control signal is set and automatic flow control is not used. See <code>set-serial-port-state</code> for manual flow control.	
	The argument <i>read-interval-timeout</i> can be used to control the maximum time to wait between each input character. The value <code>:none</code> means that reading will not wait for characters at all, only returning whatever is already in the input buffer	
	The arguments <i>read-total-base-timeout</i> and <i>read-total-byte-timeout</i> can be used to control the maximum time to wait for a sequence of characters. The arguments <i>write-total-base-timeout</i> and <i>write-total-byte-timeout</i> can be used to control the maxi-	

mum time to wait when transmitting a sequence of characters. For both reading and writing the timeout is given by the expression:

```
base_timeout + nchars * byte_timeout
```

The default value of each of *read-total-base-timeout*, *read-total-byte-timeout*, *write-total-base-timeout* and *write-total-byte-timeout* is `nil` and this means that the corresponding parameter in the OS is left unchanged and there is zero timeout. Otherwise the value should be a non-negative real number specifying a timeout in seconds.

See also

`close-serial-port`
`set-serial-port-state`

close-serial-port

Function

Summary

Closes a serial port

Package

`serial-port`

Signature

`close-serial-port serial-port`

Arguments

`serial-port` A `serial-port` object.

Description

The function `close-serial-port` closes the serial port associated with the given `serial-port` object.

If `serial-port` is already closed, an error is signalled.

See also

`open-serial-port`

get-serial-port-state

Function

Summary

Queries various aspects of the state of a serial port.

Package	<code>serial-port</code>	
Signature	<code>get-serial-port-state serial-port keys => state</code>	
Arguments	<code>serial-port</code>	A <code>serial-port</code> object.
	<code>keys</code>	A list of keywords.
Values	<code>state</code>	A list.
Description	<p>The function <code>get-serial-port-state</code> queries various aspects of the state of the serial port associated with <code>serial-port</code>.</p> <p>The argument <code>keys</code> should be a list of one or more of the keywords <code>:dsr</code> and <code>:cts</code>. These cause <code>get-serial-port-state</code> to check the DSR and CTS lines respectively.</p> <p>The result <code>state</code> is a list giving the state of each line in the same order as they appear in the argument <code>keys</code>.</p>	

serial-port	<i>Class</i>
Summary	The class of objects representing serial ports.
Package	<code>serial-port</code>
Description	The class <code>serial-port</code> is the class of objects representing serial ports. These are constructed by <code>open-serial-port</code> - do not create them directly.
See also	<code>open-serial-port</code>

read-serial-port-char	<i>Function</i>
Summary	Reads a character from a serial port.
Package	<code>serial-port</code>

This chapter applies only to LispWorks for Windows

Signature	<code>read-serial-port-char serial-port &optional timeout-error-p timeout-char => char</code>	
Arguments	<code>serial-port</code>	A <code>serial-port</code> object.
	<code>timeout-error-p</code>	A boolean.
	<code>timeout-char</code>	A character.
Values	<code>char</code>	A character.
Description	<p>The function <code>read-serial-port-char</code> reads and returns a character from the serial port associated with <code>serial-port</code>. A timeout will occur if the character is not available before the read timeout (as specified by values given when the serial port was opened by <code>open-serial-port</code>). When a timeout occurs, if <code>timeout-error-p</code> is non-<code>nil</code>, then an error of type <code>serial-port-timeout</code> is signalled, otherwise <code>timeout-char</code> is returned. The default value of <code>timeout-error-p</code> is <code>t</code>.</p>	
See also	<code>read-serial-port-string</code>	

read-serial-port-string *Function*

Summary	Reads a string from a serial port.	
Package	<code>serial-port</code>	
Signature	<code>read-serial-port-string string serial-port &optional timeout-error-p &key start end => nread</code>	
Arguments	<code>string</code>	A string.
	<code>serial-port</code>	A <code>serial-port</code> object.
	<code>timeout-error-p</code>	A boolean.
	<code>start, end</code>	Bounding index designators for <code>string</code> .

Values	<i>nread</i>	An integer.
Description	The function <code>read-serial-port-string</code> reads characters from the serial port associated with <i>serial-port</i> and places them in <i>string</i> , bounded by <i>start</i> and <i>end</i> . The default values of <i>start</i> and <i>end</i> are 0 and <code>nil</code> (interpreted as the length of <i>string</i>) respectively. The number of characters requested is the difference between <i>end</i> and <i>start</i> . If the number of characters actually read, <i>nread</i> , is less than the number requested, then if <i>timeout-error-p</i> is non- <code>nil</code> an error of type <code>serial-port-timeout</code> is signalled. If <i>nread</i> is the number of characters requested, or if <i>timeout-error-p</i> is <code>nil</code> , <i>nread</i> is returned. The default value of <i>timeout-error-p</i> is <code>t</code> .	
See also		<code>read-serial-port-char</code>

		<i>Function</i>
Summary		Checks whether a character is available on a serial port.
Package		<code>serial-port</code>
Signature		<code>serial-port-input-available-p serial-port => result</code>
Arguments	<i>serial-port</i>	A <code>serial-port</code> object.
Values	<i>result</i>	A boolean.
Description		The function <code>serial-port-input-available-p</code> checks the serial port associated with <i>serial-port</i> to see if a character is available. <i>result</i> is <code>t</code> if input is available, and <code>nil</code> otherwise.

		<i>Function</i>
set-serial-port-state		
Summary	Changes various aspects of the state of a serial port.	
Package	<code>serial-port</code>	
Signature	<code>set-serial-port-state serial-port &key dtr rts break</code>	
Arguments	<code>serial-port</code>	A <code>serial-port</code> object.
	<code>dtr</code>	A boolean.
	<code>rts</code>	A boolean.
	<code>break</code>	A boolean.
Description	The function <code>set-serial-port-state</code> changes various aspects of the state of the serial port associated with <code>serial-port</code> . The argument <code>dtr</code> , if supplied, controls the DTR line. A true value means set and <code>nil</code> means clear. If <code>dtr</code> is not supplied, the state is unchanged. The argument <code>rts</code> controls the RTS line in the same way. The argument <code>break</code> controls the break state of the data line in the same way.	

		<i>Function</i>
wait-serial-port-state		
Summary	Waits for some aspect of the state of a serial port to change.	
Package	<code>serial-port</code>	
Signature	<code>wait-serial-port-state serial-port keys &key timeout => result</code>	
Arguments	<code>serial-port</code>	A <code>serial-port</code> object.
	<code>keys</code>	A list of keywords.
	<code>timeout</code>	A number.

Values	<i>result</i>	A list.
Description	<p>The function <code>wait-serial-port-state</code> waits for some state in the serial port associated with <i>serial-port</i> to change.</p> <p>The argument <i>keys</i> should be a list of one or more of the keywords <code>:cts</code>, <code>:dsr</code>, <code>:err</code>, <code>:ring</code>, <code>:rlsd</code> and <code>:break</code>.</p> <p><i>result</i> is a list giving the keys for which the state has changed.</p> <p>If <i>timeout</i> is non-<code>nil</code> then the function will return <code>nil</code> after that many seconds even if the state has not changed.</p>	

write-serial-port-char		<i>Function</i>
Summary	Writes a character to a serial port.	
Package	<code>serial-port</code>	
Signature	<code>write-serial-port-char char serial-port &optional timeout-error-p => char</code>	
Arguments	<p><i>char</i></p> <p><i>serial-port</i></p> <p><i>timeout-error-p</i></p>	<p>A character.</p> <p>A <code>serial-port</code> object.</p> <p>A boolean.</p>
Values	<i>char</i>	A character.
Description	<p>The function <code>write-serial-port-char</code> writes the character <i>char</i> to the serial port associated with <i>serial-port</i>, and returns <i>char</i>.</p> <p>A timeout will occur if the character cannot be written before the write timeout (as specified by values given when the serial port was opened by <code>open-serial-port</code>).</p>	

When a timeout occurs, if *timeout-error-p* is non-*nil*, then an error of type **serial-port-timeout** is signalled, otherwise *nil* is returned. The default value of *timeout-error-p* is *t*.

See also

	<i>Function</i>
write-serial-port-string	
Summary	Writes a string to a serial port.
Package	serial-port
Signature	write-serial-port-string <i>string</i> <i>serial-port</i> &optional <i>timeout-error-p</i> &key <i>start end</i> => <i>nwritten</i>
Arguments	<i>string</i> A string. <i>serial-port</i> A serial-port object. <i>timeout-error-p</i> A boolean. <i>start, end</i> Bounding index designators for <i>string</i> .
Values	<i>result</i> The string <i>string</i> or <i>nil</i> .
Description	The function write-serial-port-string writes characters from the subsequence of <i>string</i> bounded by <i>start</i> and <i>end</i> to the serial port associated with <i>serial-port</i> . The default values of <i>start</i> and <i>end</i> are 0 and <i>nil</i> (interpreted as the length of <i>string</i>) respectively. If the characters are successfully written then <i>string</i> is returned. A timeout will occur if the characters cannot be written before the write timeout (as specified by values given when the serial port was opened by open-serial-port).

When a timeout occurs, if *timeout-error-p* is non-`nil`, then an error of type `serial-port-timeout` is signalled, otherwise `nil` is returned. The default value of *timeout-error-p* is `t`.

See also

`write-serial-port-char`

14

The SQL Package

This chapter describes the symbols available in the `sql` package which implements Common SQL. You should use this chapter in conjunction with the corresponding chapter in the *LispWorks User Guide*. In particular that chapter contains more information about the Oracle LOB interface (that is, those functions with names beginning `sql:ora-lob-`).

On Microsoft Windows, Linux and Mac OS X, Common SQL is included only in LispWorks Enterprise Edition.

		<i>Function</i>
Summary	Adds a stream to the broadcast list for SQL commands or results traffic.	
Package	<code>sql</code>	
Signature	<code>add-sql-stream stream &key type database => added-stream</code>	
Arguments	<code>stream</code>	A stream, or <code>t</code> .
	<code>type</code>	A keyword.

	<i>database</i>	A database.
Values	<i>added-stream</i>	The argument <i>stream</i> .
Description		<p>The <code>add-sql-stream</code> function adds the stream <i>stream</i> to the list of streams which receive SQL commands traffic or results traffic.</p> <p>To add <code>*standard-output*</code> to the list, pass <i>stream</i> <code>t</code>.</p> <p>The argument <i>type</i> is one of <code>:commands</code>, <code>:results</code> or <code>:both</code>, and determines whether a stream for commands traffic, results traffic, or both is added.</p> <p>The argument <i>type</i> has a default value of <code>:commands</code>. The <i>database</i> is the value of <code>*default-database*</code> by default.</p>
See also		<ul style="list-style-type: none"> <code>*default-database*</code> <code>delete-sql-stream</code> <code>list-sql-streams</code> <code>sql-recording-p</code> <code>sql-stream</code> <code>start-sql-recording</code> <code>stop-sql-recording</code>

		<i>Function</i>
Summary		Returns the type of an attribute.
Package		<code>sql</code>
Signature		<code>attribute-type attribute table &key database owner => datatype</code>
Arguments	<i>table</i>	A table.
	<i>attribute</i>	An attribute from <i>table</i> .
	<i>database</i>	A database.

This chapter applies to the Enterprise Edition only

	<i>owner</i>	<code>nil</code> , <code>:all</code> or a string.
Values	<i>datatype</i>	A keyword or list denoting a vendor-specific type.
Description		<p>The function <code>attribute-type</code> returns the type of the attribute specified by <i>attribute</i> in the table given by <i>table</i>. The database, in which <i>table</i> is found, has a default value of <code>*default-database*</code>.</p> <p>If <i>owner</i> is <code>nil</code>, only user-owned attributes are considered. This is the default.</p> <p>If <i>owner</i> is <code>:all</code>, all attributes are considered.</p> <p>If <i>owner</i> is a string, this denotes a username and only attributes owned by <i>owner</i> are considered.</p> <p><i>datatype</i> demotes a vendor-specific type. Examples in a MS Access database are <code>:integer</code>, <code>:longchar</code> and <code>:datetime</code>. When <i>datatype</i> is a list, the second element is the length of the type, for example <code>(:varchar 255)</code>.</p>
Example		To print the type of every attribute in the database, do
		<pre>(loop for tab in (sql:list-tables) do (loop for att in (sql:list-attributes tab) do (format t "~&Table ~S Attribute ~S Type ~S" tab att (sql:attribute-type att tab))))</pre>
See also		<code>*default-database*</code> <code>list-attribute-types</code> <code>list-attributes</code>

cache-table-queries	Function						
Summary	Controls the caching of attribute type information.						
Package	sql						
Signature	cache-table-queries <i>table</i> &key <i>database</i> <i>action</i>						
Arguments	<table border="0"> <tr> <td style="vertical-align: top;"> <i>table</i></td><td>A string naming a table, :default or t.</td></tr> <tr> <td style="vertical-align: top;"> <i>database</i></td><td>A database.</td></tr> <tr> <td style="vertical-align: top;"> <i>action</i></td><td>t, nil or :flush.</td></tr> </table>	<i>table</i>	A string naming a table, :default or t .	<i>database</i>	A database.	<i>action</i>	t , nil or :flush .
<i>table</i>	A string naming a table, :default or t .						
<i>database</i>	A database.						
<i>action</i>	t , nil or :flush .						
Description	<p>The function cache-table-queries provides per-table control on the caching in a particular database connection of attribute type information using during update operations.</p> <p>If <i>table</i> is a string, it is the name of the table for which caching is to be altered. If <i>table</i> is t, then the <i>action</i> applies to all tables. If <i>table</i> is :default, then the default caching action is set for those tables which do not have an explicit setting.</p> <p><i>database</i> specifies the database connection, its default value is the value of *default-database*.</p> <p><i>action</i> specifies the caching action. The value t means cache the attribute type information. The value nil means do not cache the attribute type information. If <i>table</i> is :default, the setting applies to all tables which do not have an explicit setup.</p> <p>The value :flush means remove any existing cache for <i>table</i> in <i>database</i>, but continue to cache.</p> <p>cache-table-queries should be called with <i>action</i> :flush when the attribute specifications in <i>table</i> have changed.</p>						
See also	<p style="margin-left: 2em;">*cache-table-queries-default*</p> <p style="margin-left: 2em;">*default-database*</p>						

This chapter applies to the Enterprise Edition only

cache-table-queries-default		<i>Variable</i>
Package	<code>sql</code>	
Initial Value	<code>nil</code>	
Description	The variable <code>*cache-table-queries-default*</code> provides the default attribute type caching behaviour. It allowed values are as described for the <i>action</i> argument of <code>cache-table-queries</code> .	
See also	<code>cache-table-queries</code>	

commit		<i>Function</i>
Summary	Commits changes made to a database.	
Package	<code>sql</code>	
Signature	<code>commit &key database => nil</code>	
Arguments	<code>database</code>	A database.
Values	<code>nil</code>	
Description	The <code>commit</code> function commits changes made to the database specified by <i>database</i> , which is <code>*default-database*</code> by default.	
Example	This example changes records in a database, and uses <code>commit</code> to make those changes permanent.	

```
(insert-records :into [emp]
                      :attributes '(x y z)
                      :values '(a b c))
(update-records [emp]
                :attributes [dept]
                :values 50
                :where [= [dept] 40])
(delete-records :from [emp]
                 :where [> [salary] 300000])
(commit)
```

See also *default-database*
 rollback
 with-transaction

	<i>Function</i>
connect	
Summary	Opens a connection to a database.
Package	sql
Signature	<i>connect connection-spec &key if-exists database-type interface name encoding signal-rollback-errors default-table-type default-table-extra-options date-string-format sql-mode prefetch-rows-number prefetch-memory => database</i>
Arguments	<p><i>connection-spec</i> The connection specifications.</p> <p><i>if-exists</i> A keyword.</p> <p><i>database-type</i> A database type.</p> <p><i>interface</i> A displayed CAPI element, or nil.</p> <p><i>name</i> A Lisp object.</p> <p><i>encoding</i> A keyword naming an encoding.</p> <p><i>signal-rollback-errors</i></p> <p style="padding-left: 20px;"><i>nil</i>, the keyword :default, or a function designator.</p>

This chapter applies to the Enterprise Edition only

default-table-type A string, the keyword
:`:support-transactions`, or `nil`.

default-table-extra-options

A string or `nil`.

date-string-format A string, or the keyword `:standard`, or `nil`.

sql-mode A string or `nil`.

prefetch-rows-number

An integer or the keyword `:default`.

prefetch-memory An integer or the keyword `:default`.

Values *database* A database.

Description The `connect` function opens a connection to a database of type *database-type*.

The allowed values for *database-type* are `:odbc`, `:odbc-driver`, `:mysql`, `:postgresql`, `:oracle8` and `:oracle`, though not all of these are supported on some platforms. See the section "Supported databases" in the *LispWorks User Guide* for details of per-platform database support.

The default for *database-type* is the value of `*default-database-type*`.

`connect` sets the variable `*default-database*` to an instance of the database opened, and returns that instance.

If *connection-spec* is a list it is interpreted as a plist of keywords and values. Some of the keywords are *database-type* specific, see the documentation for each database. General keywords are:

`:username` User name

`:password` Password

:connection A specification of the connection. In general, this is supposed to be sufficient information (other than the username and password) to open a connection. The precise meaning varies according to the *database-type*.

If *connection-spec* is a string, it is interpreted canonically as:

username/password@connection

where *connection* can be omitted along with the '@' in cases when there is a default connection, *password* can be omitted along with the preceding '/', and *username* can be omitted if there is a default user. For example, if you have an Oracle user matching the current Unix username and that does not need a password to connect, you can call

```
(connect "/")
```

Specific *database-types* may allow more elaborate syntax, but conforming to the pattern above. See the section "Initialization" in the *LispWorks User Guide* for details.

Additionally for *database-types* :odbc and :odbc-driver, if *connection-spec* does not include the '@' character then the string is interpreted in a special way, for backward compatibility with LispWorks 4.4 and earlier versions. See the section "Connecting to ODBC" in the *LispWorks User Guide* for details.

The argument *if-exists* modifies the behavior of `connect` as follows:

:new Makes a new connection even if connections to the same database already exist.

:warn-new Makes a new connection but warns about existing connections.

:error Makes a new connection but signals an error for existing connections.

:warn-old Selects old connection if one exists (and warns) or makes a new one.

This chapter applies to the Enterprise Edition only

:old Selects old connection if one exists or makes a new one.

The default value of *if-exists* is the value of ***connect-if-exists***.

interface is used if **connect** needs to display a dialog to ask the user for username and password. If *interface* is a CAPI element, this is used. If *interface* is any other value (the default value is **ni1**), and **connect** is called in a process which is associated with a CAPI interface, then this CAPI interface is used. *interface* has been added because dialogs asking for passwords can fail otherwise. This depends on the driver that the datasource uses: the problem has only been observed using MS SQL on Microsoft Windows.

name can be passed to explicitly specify the name of the connection. If *name* is supplied then it is used as-is for the connection name. Therefore it can be found by another call to **connect** and calls to **find-database**. Connection names are compared with **equalp**. If *name* is not supplied, then a unique database name is constructed from *connection-spec* and a counter.

Note: all the Common SQL functions that accept the keyword argument **:database** use **find-database** to find the database if the given value is not a database. Therefore these functions can now find only databases that were opened with an explicit *name*:

(connect ... :name *name* ...)

encoding specifies the encoding to use in the connection. The value should be a keyword naming an acceptable encoding, or **ni1** (the default). The value **:unicode** is accepted for all *database-types*, and this will try to make a connection that can support sending and retrieving double-byte string values. Other values are *database-type* specific:

<code>:mysql</code>	If <i>encoding</i> is <code>nil</code> or <code>:default</code> then the encoding is chosen according to the default character set of the connection (if available) and if that fails the encoding <code>:utf-8</code> is used. The other recognised values of <i>encoding</i> are <code>:unicode</code> , <code>:utf-8</code> , <code>:ascii</code> , <code>:latin-1</code> , <code>:euc</code> and <code>:sjis</code> . <code>:unicode</code> uses <code>:utf-8</code> internally.
<code>:postgresql</code>	<i>encoding</i> is ignored.
<code>:oracle</code>	The only recognised values of <i>encoding</i> are <code>nil</code> and <code>:unicode</code> .
<code>:oracle8</code>	<i>encoding</i> is ignored.
<code>:odbc</code>	<i>encoding</i> is ignored.
<code>:odbc-driver</code>	<i>encoding</i> is ignored.

signal-rollback-errors controls what happens when an attempted `rollback` causes an error, for databases that do not support rollback properly (for example MySQL with the default settings). For *database-types* other than `:mysql` *signal-rollback-errors* is ignored and such an error is always signalled. For *database-type* `:mysql` *signal-rollback-errors* is interpreted as follows:

<code>nil</code>	Ignore the error.
<code>:default</code>	If <i>default-table-type</i> is <code>:support-transactions</code> , "innodb" or "bdb", then rollback errors are signalled. Otherwise rollback errors are not signalled.

Function designator

The function *signal-rollback-errors* should take two arguments: the database object and a string (for an error message). The function is called when a rollback signalled an error.

The default value of *signal-rollback-errors* is `:default`.

default-table-type specifies the default value of the `:type` argument to `create-table`. See `create-table` for details. The default value of *default-table-type* is `nil`.

default-table-extra-options specifies the default value of the `:extra-options` argument to `create-table`. See `create-table` for details. The default value of *default-table-extra-options* is `nil`.

date-string-format specifies which format to use to represent dates. If the value is a string, it should be appropriate for the *database-type*. The value `:standard` means that the standard SQL date format is used. If the value is `nil` (the default), then the date format is not changed. Currently only *database-type* `:oracle` uses the value of *date-string-format*, and in this case it must be a valid date format string for Oracle.

sql-mode specifies the mode of the SQL connection for *database-type* `:mysql`. By default (that is, when *sql-mode* is not supplied) `connect` sets the mode of the connection to ANSI, by executing this statement:

```
"set sql_mode='ansi'"
```

sql-mode can be supplied as `nil`, in which case no statement is executed. Otherwise it should be a string which is a valid setting for `sql_mode`, and then `connect` executes the statement:

```
set sql_mode='sql-mode'
```

When *database-type* is not `:mysql`, *sql-mode* is ignored.

prefetch-rows-number and *prefetch-memory* are used when *database-type* is `:oracle`, and specify the amount of data to prefetch when performing queries. *prefetch-rows-number* is the number of rows to prefetch, with default value 100. *prefetch-memory* is the maximum number of bytes to prefetch, with default value `#x100000`. *prefetch-rows-number* and *prefetch-memory* can both also have the value `:default`, which allows the database to choose the amount to prefetch.

Compatibility Note	LispWorks 4.4 (and previous versions) use <i>connection-spec</i> passed to <code>connect</code> as the database name. <code>connect</code> checks if a connection with this name already exists (according to the value of <i>if-exists</i>). <code>find-database</code> can be used to find a database using this name. LispWorks 5.0 (and later versions) does not use <i>connection-spec</i> as the name. Instead, by default it generates a name from the <i>connection-spec</i> . The name is intended to be unique (by including a counter). Thus normally <code>connect</code> will not find an existing connection even if it is called again with identical value of <i>connection-spec</i> .
Example	The following example connects LispWorks to the <code>info</code> database. <code>(connect "info")</code> The next example connects to the ODBC database <code>personnel</code> using the username "admin" and the password "secret". <code>(connect "personnel/admin/secret" :database-type :odbc)</code> The next example opens a connection to MySQL which treats quotes as in ANSI but does not set other ANSI features: <code>(sql:connect "me/mypassword/mydb" :sql-mode "ANSI_QUOTES")</code>
See also	<code>*default-database*</code> <code>*default-database-type*</code> <code>connected-databases</code> <code>*connect-if-exists*</code> <code>database-name</code> <code>disconnect</code> <code>find-database</code> <code>reconnect</code> <code>status</code>

This chapter applies to the Enterprise Edition only

connect-if-exists		<i>Variable</i>
Summary	The default value for the <i>if-exists</i> keyword of the <code>connect</code> function.	
Package	<code>sql</code>	
Initial Value	<code>:error</code>	
Description	The variable <code>*connect-if-exists*</code> is the default value for the <i>if-exists</i> keyword of the <code>connect</code> function. It can take the following values:	
<code>:new</code>	Instructs <code>connect</code> to make a new connection even if connections to the same database already exist.	
<code>:warn-new</code>	Instructs <code>connect</code> to make a new connection but warn about existing connections.	
<code>:error</code>	Instructs <code>connect</code> to make a new connection but signal an error for existing connections.	
<code>:warn-old</code>	Instructs <code>connect</code> to select an old connection if one exists (and warns) or make a new one.	
<code>:old</code>	Instructs <code>connect</code> to select an old connection if one exists or make a new one.	
See also	<code>connect</code>	

connected-databases		<i>Function</i>
Summary	Returns a list of connected databases.	
Package	<code>sql</code>	
Signature	<code>connected-databases => database-list</code>	

Arguments	None.	
Values	<code>database-list</code>	A list of connected databases.
Description	The function <code>connected-databases</code> returns a list of the databases LispWorks is connected to.	
See also	connect disconnect status find-database database-name	

create-index *Function*

Summary	Creates an index for a table.	
Package	<code>sql</code>	
Signature	<code>create-index name &key on unique attributes database =></code>	
Arguments	<code>name</code>	The name of the index.
	<code>on</code>	The name of a table.
	<code>unique</code>	A boolean.
	<code>attributes</code>	A list of attributes.
	<code>database</code>	A database.
Values	None.	
Description	The function <code>create-index</code> creates an index called <code>name</code> on the table specified by <code>on</code> . The attributes of the table to index are given by <code>attributes</code> . Setting <code>unique</code> to <code>t</code> includes <code>UNIQUE</code> in the SQL index command, specifying that the columns indexed must contain unique values.	

This chapter applies to the Enterprise Edition only

The default value of *unique* is `nil`. The default value of *database* is `*default-database*`.

Example	<code>(create-index [manager] :on [emp] :unique t :attributes '([ename] [sal]))</code>
See also	<code>*default-database*</code> <code>drop-index</code> <code>create-table</code>

create-table

Function

Summary	Creates a table.										
Package	<code>sql</code>										
Signature	<code>create-table name description &key database type extra-options</code>										
Arguments	<table><tr><td><i>name</i></td><td>The name of the table.</td></tr><tr><td><i>description</i></td><td>The table properties.</td></tr><tr><td><i>database</i></td><td>A database.</td></tr><tr><td><i>type</i></td><td>A string or the keyword <code>:support-transactions</code>, or <code>nil</code>.</td></tr><tr><td><i>extra-options</i></td><td>A string or <code>nil</code>.</td></tr></table>	<i>name</i>	The name of the table.	<i>description</i>	The table properties.	<i>database</i>	A database.	<i>type</i>	A string or the keyword <code>:support-transactions</code> , or <code>nil</code> .	<i>extra-options</i>	A string or <code>nil</code> .
<i>name</i>	The name of the table.										
<i>description</i>	The table properties.										
<i>database</i>	A database.										
<i>type</i>	A string or the keyword <code>:support-transactions</code> , or <code>nil</code> .										
<i>extra-options</i>	A string or <code>nil</code> .										
Values	None.										
Description	The function <code>create-table</code> creates a table called <i>name</i> and defines its columns and other properties with <i>description</i> . The argument <i>description</i> is a list containing lists of attribute-name and type information pairs. The default value of <i>database</i> is <code>*default-database*</code> .										

type and *extra-options* are treated in a *database-type* specific way. Currently only *database-type* :`mysql` uses these options, as follows.

If *type* is not supplied, it defaults to the value (if any) of *default-table-type* that was supplied to `connect`. If *extra-options* is not supplied, it defaults to the value (if any) of *default-table-extra-options* that was supplied to `connect`.

type, if non-`nil`, is used as argument to `TYPE` in the SQL statement:

```
create table MyTable (column-specs) TYPE = type
```

except that if *type* is :`support-transactions` then `createtable` will attempt to make tables that support transactions, by using the type `innodb`.

extra-options (if non-`nil`) is appended in the end of this SQL statement.

When *database-type* is not :`mysql`, *type* and *extra-options* are ignored.

Example

The following code:

```
(create-table [manager]
  '(([id] (char 10) not-null)
    ([salary] (number 8 2))))
```

is equivalent to the following SQL:

```
CREATE TABLE MANAGER
  (ID CHAR(10) NOT NULL, SALARY NUMBER(8,2))
```

See also

```
connect
*default-database*
drop-table
```

create-view

Function

Summary

Creates a view using a specified query.

This chapter applies to the Enterprise Edition only

Package	<code>sql</code>						
Signature	<code>create-view name &key as column-list with-check-option database =></code>						
Arguments	<table><tr><td><code>name</code></td><td>The view to be created.</td></tr><tr><td><code>as</code></td><td>An SQL query statement.</td></tr><tr><td><code>column-list</code></td><td>A list.</td></tr></table>	<code>name</code>	The view to be created.	<code>as</code>	An SQL query statement.	<code>column-list</code>	A list.
<code>name</code>	The view to be created.						
<code>as</code>	An SQL query statement.						
<code>column-list</code>	A list.						
	<i>with-check-option</i>						
	A boolean.						
	<code>database</code>						
	A database.						
Values	None.						
Description	<p>The <code>create-view</code> function creates a view called <code>name</code> using the <code>as</code> query and the optional <code>column-list</code> and <code>with-check-option</code>. The <code>column-list</code> argument is a list of columns to add to the view. The <code>with-check-option</code> adds <code>WITH CHECK OPTION</code> to the resulting SQL.</p> <p>The default value of <code>with-check-option</code> is <code>nil</code>. The default value of <code>database</code> is <code>*default-database*</code>.</p>						
Example	<p>This example creates the view <code>manager</code> with the records in the employee table whose department is 50.</p> <pre>(create-view [manager] :as [select [*] :from [emp] :where [= [dept] 50]])</pre>						
See also	<code>create-index</code> <code>create-table</code> <code>*default-database*</code> <code>drop-view</code>						

		<i>Function</i>
Summary		Creates a view in a database based on a class that defines the view.
Package		<code>sql</code>
Signature		<code>create-view-from-class class &key database =></code>
Arguments	<i>class</i>	A class.
	<i>database</i>	A database.
Values		None.
Description		The function <code>create-view-from-class</code> creates a view in <i>database</i> based on <i>class</i> which defines the view. The argument <i>database</i> has a default value of <code>*default-database*</code> .
See also		<code>*default-database*</code> <code>drop-view-from-class</code> <code>create-view</code>

		<i>Function</i>
Summary		Returns the name of a database.
Package		<code>sql</code>
Signature		<code>database-name database => connection</code>
Arguments	<i>database</i>	A database.
Values	<i>connection</i>	A string.
Description		The function <code>database-name</code> returns the name of the database specified by <i>database</i> .

This chapter applies to the Enterprise Edition only

See also

```
connect  
disconnect  
connected-databases  
find-database  
status
```

default-database

Variable

Summary The default database in database operations.

Package sq1

Initial Value nil

Description The variable ***default-database*** is set by `connect` and specifies the default database to be used for database operations.

See also connect

default-database-type

Variable

Summary Specifies the default type of database.

Package sq1

Initial Value nil

Description The variable `*default-database-type*` specifies the default type of database. You can set this or it is initialized by the `initialize-database-type` function.

LispWorks supports the values shown in the section "Supported databases" in the *LispWorks User Guide*.

See also [initialize-database-type](#)

default-update-objects-max-len *Variable*

Summary The default maximum number of objects supplying data for a query when updating remote joins.

Package `sql`

Initial Value `nil`

Description The variable `*default-update-objects-max-len*` provides the default value of the `max-len` argument in the function `update-objects-joins`.

See also [update-objects-joins](#)

def-view-class *Macro*

Summary Extends the syntax of `defclass` to allow specified slots to be mapped onto the attributes of database views.

Package `sql`

Signature `def-view-class name superclasses slots &rest class-options => class`

Arguments `name` A class name.

`superclasses` The superclasses of the class to be created.

`slots` The slot definitions of the new class.

`class-options` The class options of the new class.

Values `class` The defined class.

Slot Options	The slot options for <code>def-view-class</code> are <code>:db-kind</code> and <code>:db-info</code> . In addition the slot option <code>:type</code> is treated specially for View Classes. <code>:db-kind</code> may be one of <code>:base</code> , <code>:key</code> , <code>:join</code> , or <code>:virtual</code> . The default is <code>:base</code> . Each value is described below:
<code>:base</code>	This indicates that this slot corresponds to an ordinary attribute of the database view. You can name the database attribute by using the keyword <code>:column</code> . By default, the database attribute is named by the slot.
<code>:key</code>	This indicates that this slot corresponds to part of the unique key for this view. A <code>:key</code> slot is also a <code>:base</code> slot. All View Classes must have <code>:key</code> fields that uniquely distinguish the instances, to maintain object identity. To specify a key which spans multiple slots, each of the slots should have <code>:db-kind :key</code> . The underlying requirement is that tuples of the form (key1 ... keyN) are unique. The <code>:db-kind :key</code> slots do not need to be keys in the table.
<code>:join</code>	This indicates that this slot corresponds to a join. A slot of this type will contain View Class objects.
<code>:virtual</code>	This indicates that this slot is an ordinary CLOS slot not associated with a database column.

A join is defined by the slot option `:db-info`, which takes a list. Items in the list may be:

`:join-class class-name`

This is the class to join on.

:home-key slot-name

This is the slot of the defining class to be a subject for the join. The argument *slot-name* may be an element or a list of elements, where elements can be symbols, `nil`, strings, integers or floats.

:foreign-key slot-name

This is the name of the slot of the `:join-class` to be a subject for the join. The *slot-name* may be an element or a list of elements, where elements can be symbols, `nil`, strings, integers or floats.

:target-slot target-slot

This is the name of a `:join` slot in `:join-class`. This is optional and is only specified if you want the defining slot to contain instances of this target slot as opposed to those of `:join-class`. The actual behavior depends on the value of *set*. An example of its usage is when the `:join-class` is an intermediate class and you are really only interested in it as a route to the `:target-slot`.

:retrieval retrieval-time

retrieval-time can be `:deferred`, which defers filling this slot from the database until the slot itself is accessed. This is the default value.

retrieval-time can alternatively be `:immediate` which generates the join SQL for this slot whenever a query is generated on the class. In other words, this is an intermediate class only, which is present for the

purpose of joining two entities of other classes together. When *retrieval-time* is **:immediate**, then *set* is **nil**.

:set set

When *set* is **t** and *target-slot* is defined, the slot will contain a list of pairs (*target-value join-instance*) where *target-value* is the value of the target slot and *join-instance* is the corresponding instance of the join class.

When *set* is **t** and *target-slot* is undefined, the slot will contain a list of instances of the join class.

When *set* is **nil** the slot will contain a single instance.

The default value of *set* is **t**.

The syntax for **:home-key** and **:foreign-key** means that an object from a join class will only be included in the join slot if the values from *home-key* are **equal** to the values in *foreign-key*, in order. These values are calculated as follows: if the element in the list is a symbol it is taken to be a slot name and the value of the slot is used, otherwise the element is taken to be the value. See the second example below.

The **:type** slot option is treated specially for View Classes. There is a need for stringent type-checking in View Classes because of the translation into database data, and therefore **:type** is mandatory for slots with **:db-kind :base** or **:key**. Some methods are provided for type checking and type conversion. For example, a **:type** specifier of **(string 10)** in SQL terms means allow a character type value with length of less than or equal to 10. The following Lisp types are accepted for *type*, and correspond to the SQL type shown:

(string n)	CHAR(n)
integer	INTEGER
(integer n)	INTEGER(n)

<code>float</code>	<code>FLOAT</code>
<code>(float n)</code>	<code>FLOAT(n)</code>
<code>sql:universal-time</code>	<code>TIMESTAMP</code>

Class Options `def-view-class` recognizes the following class options in addition to the standard class options defined for `defclass`:

`(:base-table table-name)`

The slots of the class *name* will be read from the table *table-name*. If you do not specify the `:base-table` option, then *table-name* defaults to the name of the class.

Description The macro `def-view-class` creates a class called *name* which maps onto a database view. Such a class is called a View Class.

The macro `def-view-class` extends the syntax of `defclass` to allow special *base slots* to be mapped onto the attributes of database views (presently single tables). When a `select` query that names a View Class is submitted, then the corresponding database view is queried, and the slots in the resulting View Class instances are filled with attribute values from the database.

If *superclasses* is `nil` then `standard-db-object` automatically becomes the superclass of the newly-defined View Class. If *superclasses* is `nil`, it must include `standard-db-object`.

Examples The following example shows a class corresponding to the traditional employees table, with the employee's department given by a join with the departments table.

This chapter applies to the Enterprise Edition only

```
(def-view-class employee (standard-db-object)
  ((employee-number :db-kind :key
                    :column empno
                    :type integer)
   (employee-name :db-kind :base
                  :column ename
                  :type (string 20)
                  :accessor employee-name)
   (employee-department :db-kind :base
                         :column deptno
                         :type integer
                         :accessor employee-department)
   (employee-job :db-kind :base
                 :column job
                 :type (string 9))
   (employee-manager :db-kind :base
                      :column mgr
                      :type integer)
   (employee-location :db-kind :join
                       :db-info (:join-class department
                                         :retrieval :deferred
                                         :set nil
                                         :home-key
                                         employee-department
                                         :foreign-key
                                         department-number
                                         :target-slot
                                         department-loc)
                       :accessor employee-location))
   (:base-table emp))
```

The following example illustrates how elements or lists of elements can follow :**home-key** and :**foreign-key** in the :**db-info** slot option.

```
(def-view-class flex-schema ()
  ((name      :type (string 8) :db-kind :key)
   (description :type (string 256)))
  (classes :db-kind :join
            :db-info (:home-key name
                                :foreign-key schema-name
                                :join-class flex-class
                                :retrieval :deferred)))
  (:base-table flex_schema))
```

```

(def-view-class flex-class ()
  ((schema-name :type (string 8) :db-kind :key
   :column schema_name)
   (name :type (string 32) :db-kind :key)
   (base-name :type (string 64) :column base_name)
   (super-classes :db-kind :join
     :db-info (:home-key
               (schema-name name)
               :foreign-key
               (schema-name class-name)
               :join-class
               flex-superclass
               :retrieval :deferred))
   (schema :db-kind :join
     :db-info (:home-key schema-name
                           :foreign-key name
                           :join-class flex-schema
                           :set nil))
   (properties :db-kind :join
     :db-info (:home-key (schema-name name ""))
               :foreign-key
               (schema-name class-name slot-name)
               :join-class flex-property
               :retrieval :deferred)))
  (:base-table flex_class))

(def-view-class flex-slot ()
  ((schema-name :type (string 8) :db-kind :key
   :column schema_name)
   (class-name :type (string 32) :db-kind :key
   :column class_name)
   (name :type (string 32) :db-kind :key)
   (class :db-kind :join
     :db-info (:home-key (schema-name class-name)
                           :foreign-key (schema-name name)
                           :join-class flex-class
                           :set nil))
   (properties :db-kind :join
     :db-info (:home-key
               (schema-name class-name name)
               :foreign-key
               (schema-name class-name slot-name)
               :join-class flex-property
               :retrieval :deferred)))
  (:base-table flex_slot)))

```

This chapter applies to the Enterprise Edition only

```
(def-view-class flex-property ()
  ((schema-name :type (string 8) :db-kind :key
     :column schema_name)
   (class-name :type (string 32) :db-kind :key
     :column class_name)
   (slot-name :type (string 32) :db-kind :key
     :column slot_name)
   (property :type (string 32) :db-kind :key)
   (values :db-kind :join
     :db-info (:home-key
       (schema-name class-name
         slot-name property)
     :foreign-key
       (schema-name class-name
         slot-name property)
       :join-class flex-property-value
       :retrieval :deferred)))
  (:base-table flex_property))

(def-view-class flex-property-value ()
  ((schema-name :type (string 8) :db-kind :key
     :column schema_name)
   (class-name :type (string 32) :db-kind :key
     :column class_name)
   (slot-name :type (string 32) :column slot_name)
   (property :type (string 32) :db-kind :key)
   (order :type integer)
   (value :type (string 128)))
  (:base-table flex_property_value))
```

See also

[create-view-from-class](#)
[delete-instance-records](#)
[drop-view-from-class](#)
[standard-db-object](#)
[update-record-from-slot](#)
[update-records-from-instance](#)

delete-instance-records

Generic Function

Summary Deletes records corresponding to View Class instances.

Package `sql`

Signature	<code>delete-instance-records <i>instance</i> =></code>
Arguments	<i>instance</i> An instance of a View Class.
Values	None.
Description	The <code>delete-instance-records</code> function deletes the records represented by <i>instance</i> from the database associated with it. If <i>instance</i> has no associated database, <code>delete-instance-records</code> signals an error.
See also	<code>update-records</code> <code>update-records-from-instance</code>

	<i>Function</i>
Summary	Deletes rows from a database table.
Package	<code>sql</code>
Signature	<code>delete-records &key <i>from where database</i> =></code>
Arguments	<i>from</i> A database table. <i>where</i> An SQL conditional statement. <i>database</i> A database.
Values	None.
Description	The <code>delete-records</code> function deletes rows from a table specified by <i>from</i> in which the <i>where</i> condition is true. The argument <i>database</i> specifies a database from which the records are to be removed, and defaults to <code>*default-database*</code> .

This chapter applies to the Enterprise Edition only

See also ***default-database***
 insert-records
 update-records

delete-sql-stream		Function
Summary	Deletes a stream from the broadcast list for SQL commands or results traffic.	
Package	sql	
Signature	<code>delete-sql-stream stream &key type database => deleted-stream</code>	
Arguments	<code>stream</code>	A stream or t.
	<code>type</code>	A keyword.
	<code>database</code>	A database.
Values	<code>deleted-stream</code>	The argument <i>stream</i> .
Description	<p>The function <code>delete-sql-stream</code> deletes the stream <i>stream</i> from the list of streams which receive SQL commands or results traffic.</p> <p>To remove <code>*standard-output*</code> from the list, pass <i>stream</i> t.</p> <p>The keyword <i>type</i> is <code>:commands</code>, <code>:results</code> or <code>:both</code>. It determines whether a stream for SQL commands traffic, results traffic, or both is deleted.</p> <p>The default value of <i>type</i> is <code>:commands</code>. The default value for <i>database</i> is the value of <code>*default-database*</code>.</p>	

See also **add-sql-stream**
default-database
list-sql-streams
sql-recording-p
sql-stream
start-sql-recording
stop-sql-recording

disable-sql-reader-syntax *Function*

Summary Turns off square bracket syntax.

Package **sql**

Signature **disable-sql-reader-syntax =>**

Arguments None.

Values None.

Description The function **disable-sql-reader-syntax** turns off square bracket syntax and sets state so that **restore-sql-reader-syntax-state** will make the syntax disabled if it is consequently enabled.

See also **enable-sql-reader-syntax**
locally-disable-sql-reader-syntax
locally-enable-sql-reader-syntax
restore-sql-reader-syntax-state

disconnect *Function*

Summary Closes a connection to a database.

Package **sql**

This chapter applies to the Enterprise Edition only

Signature	<code>disconnect &key database error => success</code>	
Arguments	<code>database</code>	A database.
	<code>error</code>	A boolean.
Values	<code>success</code>	A boolean.
Description	<p>The function <code>disconnect</code> closes a connection to a database specified by <code>database</code>. If successful, <code>success</code> is <code>t</code> and if only one other connection exists, <code>*default-database*</code> is reset.</p> <p>The default value for <code>database</code> is <code>*default-database*</code>. If <code>database</code> is a database object, then it is used directly. Otherwise, the list of connected databases is searched to find one with <code>database</code> as its connection specifications (see <code>connect</code>). If no such database is found, then if <code>error</code> and <code>database</code> are both non-<code>nil</code> an error is signaled, otherwise <code>disconnect</code> returns <code>nil</code>.</p>	
Example	<code>(disconnect :database "test")</code>	
See also	<code>connect</code> <code>connected-databases</code> <code>database-name</code> <code>*default-database*</code> <code>find-database</code> <code>reconnect</code> <code>status</code>	

do-query	<i>Macro</i>
Summary	Repeatedly binds a set of variables to the results of a query, and executes a body of code using the bound variables.
Package	<code>sql</code>

Signature	<code>do-query ((&rest args) query &key database not-inside-transaction get-all) &body body =></code>	
Arguments	<code>args</code>	A set of variables.
	<code>query</code>	A database query.
	<code>database</code>	A database.
	<i>not-inside-transaction</i>	A generalized boolean.
	<code>get-all</code>	A generalized boolean.
	<code>body</code>	A Lisp code body.
Values	None.	
Description	<p>The macro <code>do-query</code> repeatedly executes <code>body</code> within a binding of <code>args</code> on the attributes of each record resulting from <code>query</code>. <code>do-query</code> returns no values.</p> <p>The default value of <code>database</code> is <code>*default-database*</code>.</p> <p><i>not-inside-transaction</i> and <code>get-all</code> may be useful when fetching many records through a connection with <code>database-type :mysql</code>. Both of these arguments have default value <code>nil</code>. See the section "Special considerations for iteration functions and macros" in the <i>LispWorks User Guide</i> for details.</p>	
Example	<p>The following code repeatedly binds the result of selecting an entry in <code>ename</code> from the table <code>emp</code> to the variable <code>name</code>, and then prints <code>name</code> using the Lisp function <code>print</code>.</p> <pre>(do-query ((name) [select [ename] :from [emp]]) (print name))</pre>	
See also	<code>loop</code> <code>map-query</code> <code>query</code> <code>select</code> <code>simple-do-query</code>	

This chapter applies to the Enterprise Edition only

drop-index	<i>Function</i>				
Summary	Deletes an index from a database.				
Package	<code>sql</code>				
Signature	<code>drop-index index &key database =></code>				
Arguments	<table><tr><td><code>index</code></td><td>The name of an index.</td></tr><tr><td><code>database</code></td><td>A database.</td></tr></table>	<code>index</code>	The name of an index.	<code>database</code>	A database.
<code>index</code>	The name of an index.				
<code>database</code>	A database.				
Values	None.				
Description	<p>The function <code>drop-index</code> deletes <i>index</i> from <i>database</i>.</p> <p>The default value of <i>database</i> is <code>*default-database*</code>.</p>				
See also	<code>create-index</code> <code>drop-table</code>				
drop-table	<i>Function</i>				
Summary	Deletes a table from a database.				
Package	<code>sql</code>				
Signature	<code>drop-table table &key database =></code>				
Arguments	<table><tr><td><code>table</code></td><td>The name of a table.</td></tr><tr><td><code>database</code></td><td>A database.</td></tr></table>	<code>table</code>	The name of a table.	<code>database</code>	A database.
<code>table</code>	The name of a table.				
<code>database</code>	A database.				
Values	None.				
Description	<p>The function <code>drop-table</code> deletes <i>table</i> from a <i>database</i>.</p> <p>The default value of <i>database</i> is <code>*default-database*</code>.</p>				

See also **`create-table`**
 `default-database`

drop-view

Function

Summary Deletes a view from a database.

Package **sql**

Signature **`drop-view view &key database =>`**

Arguments **`view`** A view.

`database` A database.

Values None.

Description The function **drop-view** deletes *view* from *database*.

The default value of *database* is ***`default-database`***.

Note: `DROP VIEW` is not implemented in MS Access SQL, so **drop-view** does not work with that database. Use **drop-table** instead.

See also **`create-view`**
 `default-database`
 `drop-index`
 `drop-table`

drop-view-from-class

Function

Summary Deletes a view from a database based on a class defining the view.

Package **sql**

This chapter applies to the Enterprise Edition only

Signature	<code>drop-view-from-class class &key database =></code>
Arguments	<code>class</code> A class. <code>database</code> A database.
Values	None.
Description	The function <code>drop-view-from-class</code> deletes a view or base table from <code>database</code> based on <code>class</code> which defines that view. The argument <code>database</code> has a default value of <code>*default-database*</code> .
See also	<code>create-view-from-class</code> <code>*default-database*</code> <code>drop-view</code>

enable-sql-reader-syntax *Function*

Summary	Turns on square bracket SQL syntax.
Package	<code>sql</code>
Signature	<code>enable-sql-reader-syntax =></code>
Arguments	None.
Values	None.
Description	The function <code>enable-sql-reader-syntax</code> turns on square bracket syntax and sets the state so that <code>restore-sql-reader-syntax-state</code> will make the syntax enabled if it is subsequently disabled.
See also	<code>disable-sql-reader-syntax</code> <code>locally-disable-sql-reader-syntax</code>

```
locally-enable-sql-reader-syntax
restore-sql-reader-syntax-state
```

execute-command *Function*

Summary	Executes an SQL expression.				
Package	<code>sql</code>				
Signature	<code>execute-command sql-exp &key database =></code>				
Arguments	<table> <tr> <td><i>sql-exp</i></td><td>Any SQL statement other than a query.</td></tr> <tr> <td><i>database</i></td><td>A database.</td></tr> </table>	<i>sql-exp</i>	Any SQL statement other than a query.	<i>database</i>	A database.
<i>sql-exp</i>	Any SQL statement other than a query.				
<i>database</i>	A database.				
Values	None.				
Description	<p>The function <code>execute-command</code> executes the SQL command specified by <i>sql-exp</i> for the database specified by <i>database</i>, which has a default value of <code>*default-database*</code>. The argument <i>sql-exp</i> may be any SQL statement other than a query.</p> <p>To run a stored procedure, pass an appropriate string. The call to the procedure needs to be wrapped in a PL/SQL <code>BEGIN</code> <code>END</code> pair, for example:</p> <pre>(sql:execute-command "BEGIN my_procedure(1, 'foo'); END;")</pre>				
See also	<code>*default-database*</code> <code>query</code>				

find-database *Function*

Summary	Returns a database, given a database or database name.
Package	<code>sql</code>

This chapter applies to the Enterprise Edition only

Signature	<code>find-database database &optional errorp => database, count</code>	
Arguments	<code>database</code>	A string or a database.
	<code>errorp</code>	A boolean. Default value: <code>t</code> .
Values	<code>database</code>	A database.
	<code>count</code>	An integer.
Description	<p>The function <code>find-database</code>, given a string <code>database</code>, searches amongst the connected databases for one matching the name <code>database</code>. If there is exactly one such database, it is returned and the second return value <code>count</code> is 1. If more than one databases match and <code>errorp</code> is <code>nil</code>, then the most recently connected of the matching databases is returned and <code>count</code> is the number of matches. If no matching database is found and <code>errorp</code> is <code>nil</code>, then <code>nil</code> is returned. If none, or more than one, matching databases are found and <code>errorp</code> is true, then an error is signalled.</p>	
	<p>If the argument <code>database</code> is a database, it is simply returned.</p>	
See also	<code>connect</code> <code>connected-databases</code> <code>database-name</code> <code>disconnect</code> <code>status</code>	

initialize-database-type *Function*

Summary	Initializes a database type.
Package	<code>sql</code>
Signature	<code>initialize-database-type &key database-type => type</code>

Arguments	<i>database-type</i>	A database type.
Values	<i>type</i>	A database type.
Description	The function <code>initialize-database-type</code> initializes a database type by loading code and appropriate database libraries according to the value of <i>database-type</i> . If <code>*default-database-type*</code> is not initialized, this function initializes it. It adds <i>database-type</i> to the list of initialized types. The initialized database type is returned.	
Example	The following example shows how to use <code>initialize-database-type</code> to initialize the <code>:odbc</code> database type.	
	<pre>(require "odbc") (in-package sql) (setf *default-database-type* :odbc) (initialize-database-type) (print *initialized-database-types*)</pre>	
	The ODBC database type is now initialized, and connections can be made to ODBC databases.	
See also	database-name *initialized-database-types* *default-database-type*	

initialized-database-types *Variable*

Summary	A list of initialized database types.
Package	<code>sql</code>
Initial Value	<code>nil</code>
Description	The variable <code>*initialized-database-types*</code> contains a list of database types that have been initialized by calls to <code>initialize-database-type</code> .

This chapter applies to the Enterprise Edition only

See also [initialize-database-type](#)

insert-records *Function*

Summary Inserts a set of values into a table.

Package `sql`

Signature `insert-records &key into attributes values av-pairs query database`

Arguments `into` A database table.

`values` A list of values, or `nil`

`attributes` A list of attributes, or `nil`

`av-pairs` A list of two-element lists, or `nil`.

`query` A query expression, or `nil`.

`database` A database.

Values None.

Description The function `insert-records` inserts records into the table `into`.

The records created contain `values` for `attributes` (or `av-pairs`).

The argument `values` is a list of values. If `attributes` is supplied then `values` must be a corresponding list of values for each of the listed attribute names.

If `av-pairs` is non-`nil`, then both `attributes` and `values` must be `nil`.

If `query` is non-`nil`, then neither `values` nor `av-pairs` should be. `query` should be a query expression, and the attribute names in it must also exist in the table `into`.

The default value of `database` is `*default-database*`.

Example In the first example, the Lisp expression

```
(insert-records :into [person]
  :values '("abc" "Joe" "Bloggs" 10000 3000 nil
           "plumber"))
```

is equivalent to the following SQL:

```
INSERT INTO PERSON
  VALUES ('abc','Joe',
          'Bloggs',10000,3000,NULL,'plumber')
```

In the second example, the LispWorks expression

```
(insert-records :into [person]
  :attributes '(person_id income surname occupation)
  :values '("aaa" 10 "jim" "plumb"))
```

is equivalent to the following SQL:

```
INSERT INTO PERSON
  (PERSON_ID,INCOME,SURNAME,OCCUPATION)
  VALUES ('aaa',10,'jim','plumb')
```

The following example demonstrates how to use `:av-pairs`.

```
(insert-records :into [person] :av-pairs
  '((person_id "bbb") (surname "Jones")))
```

See also

- `*default-database*`
- `delete-records`
- `update-records`

instance-refreshed

Generic Function

Summary Provides a hook for user code on View Class instance updates.

Package `sql`

Signature `instance-refreshed instance`

Arguments `instance` An instance of a View Class.

This chapter applies to the Enterprise Edition only

Values	None.
Description	<p>The function <code>instance-refreshed</code> is called inside <code>select</code> when its <code>refresh</code> argument is true and the instance <code>instance</code> has just been updated.</p> <p>The supplied method on <code>standard-db-object</code> does nothing. If your application needs to take action when a View Class instance has been updated by</p> <pre>(select ... :refresh t)</pre> <p>then add an <code>instance-refresh</code> method specializing on your subclass of <code>standard-db-object</code>.</p>
See also	<code>def-view-class</code> <code>select</code>

list-attribute-types *Function*

Summary	Returns type information for a table's attributes.	
Package	<code>sql</code>	
Signature	<code>list-attribute-types table &key database owner => result</code>	
Arguments	<code>table</code>	A table.
	<code>database</code>	A database.
	<code>owner</code>	<code>nil</code> , <code>:all</code> or a string.
Values	<code>result</code>	A list.
Description	<p>The function <code>list-attribute-types</code> returns type information for the attributes in the table given by <code>table</code>. <code>database</code> has a default value of <code>*default-database*</code>.</p>	

If *owner* is `nil`, only user-owned attributes are considered.
This is the default.

If *owner* is `:all`, all attributes are considered.

If *owner* is a string, this denotes a username and only attributes owned by *owner* are considered.

result is a list in which each element is a list (*attribute datatype precision scale nullable*). *attribute* is a string denoting the attribute name. *datatype* is the vendor-specific type as described in `attribute-type`. *nullable* is 1 if the attribute accepts the value `NULL`, and 0 otherwise.

Example To print the type of every attribute in the database, do

```
(loop for tab in
      (sql:list-tables)
      do
        (loop for type-info in
              (sql:list-attribute-types tab)
              do
                (format t "~&Table ~S Attribute ~S Type ~S"
                        tab
                        (first type-info)
                        (second type-info))))
```

See also `attribute-type`
`list-attributes`

list-attributes *Function*

Summary Returns a list of attributes from a table in a database.

Package `sql`

Signature `list-attributes table &key database owner => result`

Arguments `table` A table in the database.

`database` A database.

This chapter applies to the Enterprise Edition only

	<i>owner</i>	<code>nil</code> , <code>:all</code> or a string.
Values	<i>result</i>	A list of attributes.
Description		The function <code>list-attributes</code> returns a list of attributes from <i>table</i> in <i>database</i> , which has a default value of <code>*default-database*</code> . If <i>owner</i> is <code>nil</code> , only user-owned attributes are considered. This is the default. If <i>owner</i> is <code>:all</code> , all attributes are considered. If <i>owner</i> is a string, this denotes a username and only attributes owned by <i>owner</i> are considered.
See also		<code>attribute-type</code> <code>list-attribute-types</code> <code>list-tables</code>

list-classes *Function*

Summary	Returns a list of View Classes connected to a given database.	
Package	<code>sql</code>	
Signature	<code>list-classes &key database root-class test => result-list</code>	
Arguments	<i>database</i>	A database.
	<i>root-class</i>	A class.
	<i>test</i>	A test function.
Values	<i>result-list</i>	A list of class objects.
Description	The function <code>list-classes</code> collects all the classes below <i>root-class</i> (which defaults to <code>standard-db-object</code>) that are connected to the given database specified by <i>database</i> , and which	

satisfy the *test* function. The default for the *test* argument is **identity**.

By default, **list-classes** returns a list of all the classes connected to the default database, ***default-database***.

	<i>Function</i>				
Summary	Returns the broadcast list of streams recording SQL commands or results traffic.				
Package	sql				
Signature	list-sql-streams &key type database => streams				
Arguments	<table> <tr> <td><i>type</i></td><td>A keyword.</td></tr> <tr> <td><i>database</i></td><td>A database.</td></tr> </table>	<i>type</i>	A keyword.	<i>database</i>	A database.
<i>type</i>	A keyword.				
<i>database</i>	A database.				
Values	<i>streams</i> A list.				
Description	<p>The function list-sql-streams returns the broadcast list of streams recording SQL commands or results traffic.</p> <p>Each element of <i>streams</i> is a stream or the symbol t, denoting *standard-output*.</p> <p>The keyword <i>type</i> is one of :commands or :results, and determines whether to return a list of streams for SQL commands or results traffic.</p> <p>The default value of <i>type</i> is :commands. The default value for <i>database</i> is the value of *default-database*.</p>				
See also	add-sql-stream delete-sql-stream sql-recording-p sql-stream				

This chapter applies to the Enterprise Edition only

```
start-sql-recording  
stop-sql-recording
```

		<i>Function</i>
list-tables		
Summary		Returns a list of the table names in a database.
Package		<code>sql</code>
Signature		<code>list-tables &key database owner => table-list</code>
Arguments	<i>database</i>	A database.
	<i>owner</i>	<code>nil</code> , <code>:all</code> or a string.
Values	<i>table-list</i>	A list of table names.
Description		The function <code>list-tables</code> returns the list of table names in <i>database</i> , which has a default value of <code>*default-database*</code> . If <i>owner</i> is <code>nil</code> , only user-owned tables are considered. This is the default. If <i>owner</i> is <code>:all</code> , all tables are considered. If <i>owner</i> is a string, this denotes a username and only tables owned by <i>owner</i> are considered.
See also		<code>create-table</code> <code>drop-table</code> <code>list-attributes</code> <code>table-exists-p</code>
lob-stream		<i>Class</i>
Summary		The LOB stream class.

Superclasses	<code>buffered-stream</code>
Initargs	<code>:lob-locator</code> A LOB locator. <code>:direction</code> One of <code>:input</code> or <code>:output</code> . <code>:free-lob-locator-on-close</code> A generalized boolean.
Accessors	<code>lob-stream-lob-locator</code>
Description	<p>The <code>lob-stream</code> class implements LOB streams in the Oracle LOB interface.</p> <p>A <code>lob-stream</code> for input can be returned from <code>select</code> or <code>query</code> by specifying <code>:input-stream</code> as the type to return for the LOB column.</p> <p>A <code>lob-stream</code> for output can be returned from <code>select</code> or <code>query</code> by specifying <code>:output-stream</code> as the type to return for the LOB column.</p> <p>A <code>lob-stream</code> can be attached to an existing LOB locator by creating the stream explicitly.</p> <p><i>direction</i> specifies whether the stream is for input or output. The default value of <i>direction</i> is <code>:input</code>.</p> <p>By default, if the stream is closed the LOB locator is freed, unless <code>free-lob-locator-on-close</code> is passed as <code>nil</code>. The default value of <code>free-lob-locator-on-close</code> is <code>t</code>.</p>
Example	<p>This creates an input stream connected to the LOB locator <i>lob-locator</i>:</p> <pre>(make-instance 'lob-stream :lob-locator <i>lob-locator</i>)</pre>
See also	<code>query</code> <code>select</code>

This chapter applies to the Enterprise Edition only

locally-disable-sql-reader-syntax *Function*

Summary	Turns off square bracket syntax and does not change syntax state.
Package	<code>sql</code>
Signature	<code>locally-disable-sql-reader-syntax =></code>
Arguments	None.
Values	None.
Description	The function <code>locally-disable-sql-reader-syntax</code> turns off square bracket syntax and does not change syntax state. This ensures that <code>restore-sql-reader-syntax-state</code> restores the current enable/disable state.
Example	The intended use of <code>locally-disable-sql-reader-syntax</code> is in a file: <code>#.(locally-disable-sql-reader-syntax) <Lisp code not using [...] syntax> #.(restore-sql-reader-syntax-state)</code>
See also	<code>disable-sql-reader-syntax</code> <code>enable-sql-reader-syntax</code> <code>locally-enable-sql-reader-syntax</code> <code>restore-sql-reader-syntax-state</code>

locally-enable-sql-reader-syntax *Function*

Summary	Turns on square bracket syntax and does not change syntax state.
Package	<code>sql</code>

Signature	<code>locally-enable-sql-reader-syntax</code>
Arguments	None.
Values	None.
Description	The function <code>locally-enable-sql-reader-syntax</code> turns on square bracket syntax and does not change the syntax state. This ensures that <code>restore-sql-reader-syntax-state</code> restores the current enable/disable state.
Example	The intended use of <code>locally-enable-sql-reader-syntax</code> is in a file:
	<pre>#.(locally-enable-sql-reader-syntax) <code using [...] syntax> #.(restore-sql-reader-syntax-state)</pre>
See also	<code>disable-sql-reader-syntax</code> <code>enable-sql-reader-syntax</code> <code>locally-disable-sql-reader-syntax</code> <code>restore-sql-reader-syntax-state</code>

	loop	<i>Macro</i>				
Summary	Extends <code>loop</code> to provide a clause for iterating over query results.					
Package	<code>common-lisp</code>					
Signature	<code>loop {for as} var [type-spec] being {the each} {records record} {in of} query-expression [not-inside-transaction <i>not-inside-transaction</i>] [get-all get-all] => result</code>					
Arguments	<table> <tr> <td><code>var</code></td> <td>A variable.</td> </tr> <tr> <td><code>query-expression</code></td> <td>An SQL query statement.</td> </tr> </table>	<code>var</code>	A variable.	<code>query-expression</code>	An SQL query statement.	
<code>var</code>	A variable.					
<code>query-expression</code>	An SQL query statement.					

	<i>not-inside-transaction</i>	A generalised boolean.
	<i>get-all</i>	A generalised boolean.
Values	<i>result</i>	A <code>loop</code> return value.
Description		The Common Lisp <code>loop</code> macro has been extended with a clause for iterating over query results. This extension is available only when Common SQL has been loaded. For a full description of the rest of the Common Lisp <code>loop</code> facility, including the possible return values, see the ANSI Common Lisp specification. Each iteration of the loop assigns the next record of the table to the variable <code>var</code> . The record is represented in Lisp as a list. Destructuring can be used in <code>var</code> to bind variables to specific attributes of the records resulting from <i>query-expression</i> . In conjunction with the panoply of existing clauses available from the <code>loop</code> macro, the new iteration clause provides an integrated report generation facility.
		The additional loop keywords <code>not-inside-transaction</code> and <code>get-all</code> may be useful when fetching many records through a connection with <code>database-type :mysql</code> . See the section "Special considerations for iteration functions and macros" in the <i>LispWorks User Guide</i> for details.
Example		This extended <code>loop</code> example performs the following on each record returned as a result of a query: bind <code>name</code> to the query result, find the salary (if any) from an associated hash-table, increment a count for salaries greater than 20000, accumulate the salary, and print the details. Finally, it prints the average salary.

```
(loop
    for (name) being each record in
        [select [ename] :from [emp]]
        as salary = (gethash name *salary-table*)
        initially (format t "~&~20A~10D" 'name 'salary)
        when (and salary (> salary 20000))
            count salary into salaries
            and sum salary into total
            and do (format t "~&~20A~10D" name salary)
        else
            do (format t "~&~20A~10A" name "N/A")
        finally
            (format t "~2&Av Salary: ~10D" (/ total salaries)))
```

See also

[do-query](#)
[map-query](#)
[query](#)
[select](#)
[simple-do-query](#)

map-query*Function*

Summary Returns the results of mapping a function across an SQL query statement.

Package `sql`

Signature `map-query output-type-spec function query-exp &key database not-inside-transaction get-all => result`

Arguments `output-type-spec` The output type specification.

`result-type` The result sequence type.

`function` A function.

`query-exp` An SQL query.

`database` A database.

`not-inside-transaction`

A generalized boolean.

This chapter applies to the Enterprise Edition only

	<code>get-all</code>	A generalized boolean.
Values	<code>result</code>	A sequence of type <i>output-type-spec</i> containing the results of the map function.
Description	The function <code>map-query</code> returns the result of mapping <i>function</i> across the results of <i>query-exp</i> . The <i>output-type-spec</i> argument specifies the type of the result sequence as per the Common Lisp <code>map</code> function.	<p>The default value of <i>database</i> is <code>*default-database*</code>.</p> <p><i>not-inside-transaction</i> and <code>get-all</code> may be useful when fetching many records through a connection with <i>database-type</i> <code>:mysql</code>. Both of these arguments have default value <code>nil</code>. See the section "Special considerations for iteration functions and macros" in the <i>LispWorks User Guide</i> for details.</p>
Example	This example binds <code>name</code> to each name in the employee table and prints it.	<pre>(map-query nil #'(lambda (name) (print name)) [select [ename] :from [emp] :flatp t])</pre>
See also	<code>do-query</code> <code>loop</code> <code>print-query</code> <code>query</code> <code>select</code> <code>simple-do-query</code>	

mysql-library-directories

Variable

Package	<code>sql</code>
Initial Value	<code>"C:\\Program Files\\MySQL\\MySQL*\\bin"</code>

Description	The variable *mysql-library-directories* helps Lisp-Works for Windows to locate the MySQL library for use with <i>database-type :mysql</i> . It specifies a directory or a list of directories in which to search for the MySQL library. If the value is a directory pathname specifier then it is passed to directory . If the value is a list of directory pathname specifiers then each item is passed to directory . The collected results are the list of directories to search in. The default value matches the default MySQL installation. Note that this default will match any MySQL release, so if you need to be sure to match a specific MySQL release, you need to change the value of *mysql-library-directories* such that it matches only that particular release.
See also	*mysql-library-path*

mysql-library-path *Variable*

Package	sql
Initial Value	On Microsoft Windows: <code>"libmysql.dll"</code> On other platforms with pthreads: <code>"-lmysqlclient_r"</code> On other platforms without pthreads: <code>"-lmysqlclient"</code>
Description	The variable *mysql-library-path* helps the system to locate the MySQL library for use with <i>database-type :mysql</i> . It specifies the library name, and can also be set to a full path. If it is not a name, the system searches the standard library locations.

This chapter applies to the Enterprise Edition only

You can override the value of `*mysql-library-path*` by setting the environment variable `LW_MYSQL_LIBRARY`.

See also `*mysql-library-directories*`

ora-lob-append *Function*

Summary Appends two internal LOBs together.

Package `sql`

Signature `ora-lob-append src-lob-locator dest-lob-locator &key errorp`

Arguments `src-lob-locator` A LOB locator.

`dest-lob-locator` A LOB locator.

`errorp` A generalized boolean.

Description The function `ora-lob-append` appends the contents of the LOB pointed to by `src-lob-locator` to the end of LOB pointed by `dest-lob-locator`. The source and destination LOBs must be of the same internal LOB type, that is, either both BLOB or both CLOB/NCLOB.

If an error occurs and `errorp` is true, an error is signaled. If `errorp` is false, the function returns an object of type `sql-database-error`. The default value of `errorp` is `nil`.

`ora-lob-append` is applicable to internal LOBs only.

Note: This is a direct call OCILobAppend.

Note: this function is available only when the "oracle" module is loaded. See the section on the Oracle LOB interface in the *LispWorks User Guide* for more information.

		<i>Function</i>
ora-lob-assign		
Summary	Assigns a LOB to another LOB locator.	
Package	<code>sql</code>	
Signature	<code>ora-lob-assign src-lob-locator &key dest-lob-locator errorp => lob--locator</code>	
Arguments	<p><code>src-lob-locator</code> A LOB locator.</p> <p><code>dest-lob-locator</code> A LOB locator.</p> <p><code>errorp</code> A generalized boolean.</p>	
Values	<code>lob-locator</code> A LOB locator.	
Description	<p>The function <code>ora-lob-assign</code> assigns the underlying LOB for <code>src-lob-locator</code> to another LOB locator.</p> <p>If <code>dest-lob-locator</code> is <code>nil</code> then a new LOB locator is created and returned. Otherwise <code>dest-lob-locator</code> should be an existing LOB locator which is modified and returned. The default value of <code>dest-lob-locator</code> is <code>nil</code>.</p> <p>If an error occurs and <code>errorp</code> is true, an error is signaled. If <code>errorp</code> is false, the function returns an object of type <code>sql-database-error</code>. The default value of <code>errorp</code> is <code>nil</code>.</p> <p>Note: This is a direct call to <code>OCILobAssign</code>.</p> <p>Note: this function is available only when the "oracle" module is loaded. See the section on the Oracle LOB interface in the <i>LispWorks User Guide</i> for more information.</p>	

		<i>Function</i>
ora-lob-char-set-form		
Summary	Returns the character set form of a LOB.	
Package	<code>sql</code>	

This chapter applies to the Enterprise Edition only

Signature	<code>ora-lob-char-set-form lob-locator &key errorp => charset</code>	
Arguments	<i>lob-locator</i>	A LOB locator.
	<i>errorp</i>	A generalized boolean.
Values	<i>charset</i>	A non-negative integer.
Description	<p>The function <code>ora-lob-char-set-form</code> returns the char set form of the LOB underlying <i>lob-locator</i>. <i>charset</i> is 0 for a binary LOB (BLOB or BFILE), SQLCS_IMPLICIT (1) for a character LOB (CFILE or CLOB) and SQLCS_NCHAR (2) for a NCLOB.</p> <p>If an error occurs and <i>errorp</i> is true, an error is signaled. If <i>errorp</i> is false, the function returns an object of type <code>sql-database-error</code>. The default value of <i>errorp</i> is <code>nil</code>.</p> <p>Note: This is a direct call to <code>OCILobCharSetForm</code>.</p> <p>Note: this function is available only when the "oracle" module is loaded. See the section on the Oracle LOB interface in the <i>LispWorks User Guide</i> for more information.</p>	

		<i>Function</i>
Summary	Returns the database character set identifier of a LOB.	
Package	<code>sql</code>	
Signature	<code>ora-lob-char-set-id lob-locator &key errorp => db-charset-id</code>	
Arguments	<i>lob-locator</i>	A LOB locator.
	<i>errorp</i>	A generalized boolean.
Values	<i>db-charset-id</i>	A non-negative number.

Description	<p>The function <code>ora-lob-char-set-id</code> returns the database character set identifier of the LOB underlying <i>lob-locator</i>. <i>db-charset-id</i> is 0 for a binary LOB.</p> <p>If an error occurs and <i>errorp</i> is true, an error is signaled. If <i>errorp</i> is false, the function returns an object of type <code>sql-database-error</code>. The default value of <i>errorp</i> is <code>nil</code>.</p> <p>Note: This is a direct call to <code>OCILobCharSetID</code>.</p> <p>Note: this function is available only when the "oracle" module is loaded. See the section on the Oracle LOB interface in the <i>LispWorks User Guide</i> for more information.</p>
-------------	--

	<i>Function</i>				
Summary	Closes an opened LOB.				
Package	<code>sql</code>				
Signature	<code>ora-lob-close lob-locator &key errorp</code>				
Arguments	<table> <tr> <td><i>lob-locator</i></td><td>A LOB locator.</td></tr> <tr> <td><i>errorp</i></td><td>A generalized boolean.</td></tr> </table>	<i>lob-locator</i>	A LOB locator.	<i>errorp</i>	A generalized boolean.
<i>lob-locator</i>	A LOB locator.				
<i>errorp</i>	A generalized boolean.				
Description	<p>The function <code>ora-lob-close</code> closes a LOB which has been opened by <code>ora-lob-open</code>.</p> <p>For more information see <code>ora-lob-open</code>.</p> <p>If an error occurs and <i>errorp</i> is true, an error is signaled. If <i>errorp</i> is false, the function returns an object of type <code>sql-database-error</code>. The default value of <i>errorp</i> is <code>nil</code>.</p> <p>Note: This is a direct call to <code>OCILobClose</code>.</p> <p>Note: this function is available only when the "oracle" module is loaded. See the section on the Oracle LOB interface in the <i>LispWorks User Guide</i> for more information.</p>				

This chapter applies to the Enterprise Edition only

See also [ora-lob-open](#)

ora-lob-copy *Function*

Summary Copies part of an internal LOB.

Package `sql`

Signature `ora-lob-copy dest-lob-locator src-lob-locator amount &key dest-offset src-offset errorp`

Arguments *dest-lob-locator* A LOB locator.

src-lob-locator A LOB locator.

amount A non-negative integer.

dest-offset A non-negative integer.

src-offset A non-negative integer.

errorp A generalized boolean.

Description The function `ora-lob-copy` copies part of the LOB pointed to by *src-lob-locator* into the LOB pointed to by *dest-lob-locator*.

The details of the operation are determined by *amount*, *src-offset* and *dest-offset*. These numbers are in characters for CLOB/NCLOB and bytes for BLOB, and the offsets start from 1. The part of the source LOB from offset *src-offset* of length *amount* is copied into the destination LOB at offset *dest-offset*. The default value of *dest-offset* is 1 and the default value of *src-offset* is 1.

The destination LOB is extended if needed. If the *dest-offset* is beyond the end of the destination LOB, the gap between the end and *dest-offset* is erased, that is, filled with 0 for BLOBs or spaces for CLOBs.

Both LOBs must be internal LOBs, and they must be of the same type, that is, either both BLOB or both CLOB/NCLOB.

`ora-lob-append` is applicable to internal LOBs only.

If an error occurs and `errorp` is true, an error is signaled. If `errorp` is false, the function returns an object of type `sql-database-error`. The default value of `errorp` is `nil`.

Note: This is a direct call OCILobCopy.

Note: this function is available only when the "oracle" module is loaded. See the section on the Oracle LOB interface in the *LispWorks User Guide* for more information.

See also `ora-lob-load-from-file`

ora-lob-create-empty *Function*

Summary Creates an empty LOB.

Package `sql`

Signature `ora-lob-create-empty &key db type => lob--locator`

Arguments `db` A database.

`type` A Lisp object.

Values `lob--locator` A LOB locator.

Description The function `ora-lob-create-empty` creates an empty LOB object and returns a LOB locator for it.

If `type` is `:lob` then `ora-lob-create-empty` creates a LOB of type BLOB/CLOB. If `type` is any other value, it creates a file LOB. The default value of `type` is `:lob`.

Empty LOBs can be put in the database by passing them to `insert-records` or `update-records`. However, the preferred approach is to use the Oracle SQL function `EMPTY_BLOB` as

described in the section "Inserting empty LOBs" in the *Lisp-Works User Guide*.

The default value of *db* is the value of `*default-database*`.

Note: this function is available only when the "oracle" module is loaded. See the section on the Oracle LOB interface in the *LispWorks User Guide* for more information.

ora-lob-create-temporary	<i>Function</i>
Summary	Creates a temporary LOB.
Package	<code>sql</code>
Signature	<code>ora-lob-create-temporary db-or-lob-locator &key errorp cache session-duration clob-p => lob-locator</code>
Arguments	<p><i>db-or-lob-locator</i> A database or a LOB locator.</p> <p><i>errorp</i> A generalized boolean.</p> <p><i>cache</i> A generalized boolean.</p> <p><i>session-duration</i> A generalized boolean.</p> <p><i>clob-p</i> A generalized boolean.</p>
Values	<i>lob-locator</i> A LOB locator.
Description	<p>The function <code>ora-lob-create-temporary</code> creates a temporary LOB.</p> <p><i>db-or-lob-locator</i> specifies the database to associate the new LOB with. If it is a LOB locator the database from which the LOB locator came is used.</p> <p>If an error occurs and <i>errorp</i> is true, an error is signaled. If <i>errorp</i> is false, the function returns an object of type <code>sql-database-error</code>. The default value of <i>errorp</i> is <code>nil</code>.</p>

cache specifies whether to use a cache or not. The default value of *cache* is `nil`.

session-duration specifies the lifetime: if it is true then it uses `OCI_DURATION_SESSION`, otherwise it uses `OCI_DURATION_CALL`. The default value of *session-duration* is `t`.

If *clob-p* is true then the new LOB is a CLOB, otherwise it is a BLOB. The default value of *clob-p* is `nil`.

The new temporary LOB locator is returned.

Note: This is a direct call to `OCILobCreateTemporary`.

Note: this function is available only when the "oracle" module is loaded. See the section on the Oracle LOB interface in the *LispWorks User Guide* for more information.

See also	<code>ora-lob-free-temporary</code>
	<code>ora-lob-is-temporary</code>

ora-lob-disable-buffering

Function

Summary	Disables the buffering of the Oracle client.	
Package	<code>sql</code>	
Signature	<code>ora-lob-disable-buffering lob-locator &key errorp</code>	
Arguments	<i>lob-locator</i>	A LOB locator.
	<i>errorp</i>	A generalized boolean.
Description	<p>The function <code>ora-lob-disable-buffering</code> disables the buffering of the Oracle client. This function does not flush the buffers.</p> <p>This function is applicable to internal LOBs only.</p>	

If an error occurs and *errorp* is true, an error is signaled. If *errorp* is false, the function returns an object of type **sql-database-error**. The default value of *errorp* is **nil**.

Note: This is a direct call to OCILobDisableBuffering.

Note: this function is available only when the "oracle" module is loaded. See the section on the Oracle LOB interface in the *LispWorks User Guide* for more information.

See also **ora-lob-enable-buffering**
 ora-lob-flush-buffer

ora-lob-element-type *Function*

Summary Returns the Lisp element type corresponding to that of a LOB locator.

Package **sql**

Signature **ora-lob-element-type lob-locator => type**

Arguments *lob-locator* A LOB locator.

Values *type* A Lisp type descriptor.

Description The function **ora-lob-element-type** returns the Lisp element type that best corresponds to the charset of the LOB locator *lob-locator*.

For BLOB and BFILE *type* is (**unsigned-byte 8**). For CLOB, NCLOB and CFILER *type* is either **base-char** or **simple-char**, depending on the charset.

Note: this function is available only when the "oracle" module is loaded. See the section on the Oracle LOB interface in the *LispWorks User Guide* for more information.

		<i>Function</i>
ora-lob-enable-buffering		
Summary	Enables the buffering of the Oracle client.	
Package	<code>sql</code>	
Signature	<code>ora-lob-enable-buffering lob-locator &key errorp</code>	
Arguments	<p><i>lob-locator</i> A LOB locator.</p> <p><i>errorp</i> A generalized boolean.</p>	
Description	<p>The function <code>ora-lob-enable-buffering</code> enables the buffering of the Oracle client. This function does not flush the buffers.</p> <p>This function is applicable to internal LOBs only.</p> <p>If an error occurs and <i>errorp</i> is true, an error is signaled. If <i>errorp</i> is false, the function returns an object of type <code>sql-database-error</code>. The default value of <i>errorp</i> is <code>nil</code>.</p> <p>Note: This is a direct call to <code>OCILobEnableBuffering</code>.</p> <p>Note: this function is available only when the "oracle" module is loaded. See the section on the Oracle LOB interface in the <i>LispWorks User Guide</i> for more information.</p>	
See also	<p><code>ora-lob-disable-buffering</code></p> <p><code>ora-lob-flush-buffer</code></p>	

		<i>Function</i>
ora-lob-env-handle		
Summary	Returns a foreign pointer to the environment handle of a LOB.	
Package	<code>sql</code>	
Signature	<code>ora-lob-env-handle lob-locator => pointer</code>	

This chapter applies to the Enterprise Edition only

Arguments	<i>lob-locator</i>	A LOB locator.
Values	<i>pointer</i>	A foreign pointer of type <code>sql:p-oci-env</code> .
Description	The function <code>ora-lob-env-handle</code> returns a foreign pointer to the environment handle of the LOB underlying <i>lob-locator</i> . Note: this function is available only when the "oracle" module is loaded. See the section on the Oracle LOB interface in the <i>LispWorks User Guide</i> for more information.	

ora-lob-erase *Function*

Summary	Erases part of an internal LOB.	
Package	<code>sql</code>	
Signature	<code>ora-lob-erase lob-locator offset amount &key errorp => erased</code>	
Arguments	<i>lob-locator</i>	A LOB locator.
	<i>offset</i>	A non-negative integer.
	<i>amount</i>	A non-negative integer.
	<i>errorp</i>	A generalized boolean.
Values	<i>erased</i>	A non-negative integer.
Description	<p>The function <code>ora-lob-erase</code> erases part of the LOB pointed to by <i>lob-locator</i>. That is, it fills part of the LOB with 0 for BLOBs or spaces for CLOBs.</p> <p>The operation starts from offset <i>offset</i> into the LOB and erases <i>amount</i> of data in the LOB, or to the end of the LOB. Note that the offset starts from 1, and that <i>offset</i> and <i>amount</i> are in characters for CLOBs and bytes for BLOB.</p>	

Erasing does not extend beyond the end of the LOB. The return value *erased* is the number of characters or bytes erased. *erased* will be smaller than *amount* if the sum of *offset* and *amount* is greater than the length of the LOB.

ora-lob-erase is applicable to internal LOBs only.

If an error occurs and *errorp* is true, an error is signaled. If *errorp* is false, the function returns an object of type **sql-database-error**. The default value of *errorp* is **nil**.

Note: This is a direct call to OCILobErase.

Note: this function is available only when the "oracle" module is loaded. See the section on the Oracle LOB interface in the *LispWorks User Guide* for more information.

	<i>Function</i>
Summary	Closes a file LOB.
Package	sql
Signature	ora-lob-file-close <i>file-lob-locator</i> & <i>key</i> <i>errorp</i>
Arguments	<i>file-lob-locator</i> A file LOB locator. <i>errorp</i> A generalized boolean.
Description	<p>The function ora-lob-file-close closes the file that <i>file-lob-locator</i> is associated with.</p> <p>If an error occurs and <i>errorp</i> is true, an error is signaled. If <i>errorp</i> is false, the function returns an object of type sql-database-error. The default value of <i>errorp</i> is nil.</p> <p>Note: This is a direct call to OCILobFileClose.</p> <p>Note: this function is available only when the "oracle" module is loaded. See the section on the Oracle LOB interface in the <i>LispWorks User Guide</i> for more information.</p>

This chapter applies to the Enterprise Edition only

See also **ora-lob-file-open**

ora-lob-file-close-all *Function*

Summary Closes all the file LOBs.

Package **sql**

Signature **ora-lob-file-close-all &key db errorp**

Arguments **db** A database.

errorp A generalized boolean.

Description The function **ora-lob-file-close** closes the files that are associated with all the file LOB locators that are opened through the database connection specified by *database*.

The default value of *db* is the value of ***default-database***.

If an error occurs and *errorp* is true, an error is signaled. If *errorp* is false, the function returns an object of type **sql-database-error**. The default value of *errorp* is **nil**.

Note: This is a direct call to OCILobFileCloseAll.

Note: this function is available only when the "oracle" module is loaded. See the section on the Oracle LOB interface in the *LispWorks User Guide* for more information.

See also **ora-lob-file-close**

ora-lob-file-exists *Function*

Summary The predicate for whether a LOB file exists.

Package **sql**

Signature	<code>ora-lob-file-exists lob-locator &key errorp => result</code>	
Arguments	<i>lob-locator</i>	A LOB locator.
	<i>errorp</i>	A generalized boolean.
Values	<i>result</i>	A boolean.
Description	<p>The function <code>ora-lob-file-exists</code> returns <code>t</code> if the file associated with the LOB exists. This function is applicable only to file LOBs (CFILE or BFILE).</p> <p>If an error occurs and <i>errorp</i> is true, an error is signaled. If <i>errorp</i> is false, the function returns an object of type <code>sql-database-error</code>. The default value of <i>errorp</i> is <code>nil</code>.</p>	
	<p>Note: This is a direct call to <code>OCILobFileExists</code>.</p> <p>Note: this function is available only when the "oracle" module is loaded. See the section on the Oracle LOB interface in the <i>LispWorks User Guide</i> for more information.</p>	

		<i>Function</i>
Summary	Returns the directory and name for the file associated with a file LOB.	
Package	<code>sql</code>	
Signature	<code>ora-lob-file-get-name lob-locator &key errorp => dir, filename</code>	
Arguments	<i>lob-locator</i>	A LOB locator.
	<i>errorp</i>	A generalized boolean.
Values	<i>dir</i>	A string of length no greater than 30.
	<i>filename</i>	A string of length no greater than 255.

This chapter applies to the Enterprise Edition only

Description	The function <code>ora-lob-file-get-name</code> returns as multiple values the directory alias <i>dir</i> and the filename <i>filename</i> associated with the LOB denoted by <i>lob-locator</i> . The function is applicable only to file LOBs (CFILE or BFILE). If an error occurs and <i>errorp</i> is true, an error is signaled. If <i>errorp</i> is false, the function returns an object of type <code>sql-database-error</code> . The default value of <i>errorp</i> is <code>nil</code> .
	Note: This is a direct call to <code>OCILobFileGetName</code> . Note: this function is available only when the "oracle" module is loaded. See the section on the Oracle LOB interface in the <i>LispWorks User Guide</i> for more information.

	<i>Function</i>				
ora-lob-file-is-open					
Summary	The predicate for whether a LOB file is open.				
Package	<code>sql</code>				
Signature	<code>ora-lob-file-is-open lob-locator &key errorp => result</code>				
Arguments	<table><tr><td><i>lob-locator</i></td><td>A LOB locator.</td></tr><tr><td><i>errorp</i></td><td>A generalized boolean.</td></tr></table>	<i>lob-locator</i>	A LOB locator.	<i>errorp</i>	A generalized boolean.
<i>lob-locator</i>	A LOB locator.				
<i>errorp</i>	A generalized boolean.				
Values	<i>result</i> A boolean.				
Description	The function <code>ora-lob-file-is-open</code> returns <code>t</code> if the file associated with the LOB is open. This function is applicable only to file LOBs (CFILE or BFILE). If an error occurs and <i>errorp</i> is true, an error is signaled. If <i>errorp</i> is false, the function returns an object of type <code>sql-database-error</code> . The default value of <i>errorp</i> is <code>nil</code> .				
	Note: This is a direct call to <code>OCILobFileIsOpen</code> .				

Note: this function is available only when the "oracle" module is loaded. See the section on the Oracle LOB interface in the *LispWorks User Guide* for more information.

ora-lob-file-open *Function*

Summary	Opens a file LOB.				
Package	<code>sql</code>				
Signature	<code>ora-lob-file-open file-lob-locator &key errorp</code>				
Arguments	<table> <tr> <td><code>file-lob-locator</code></td><td>A file LOB locator.</td></tr> <tr> <td><code>errorp</code></td><td>A generalized boolean.</td></tr> </table>	<code>file-lob-locator</code>	A file LOB locator.	<code>errorp</code>	A generalized boolean.
<code>file-lob-locator</code>	A file LOB locator.				
<code>errorp</code>	A generalized boolean.				
Description	<p>The function <code>ora-lob-file-open</code> opens the file that <code>file-lob-locator</code> is associated with.</p> <p>If an error occurs and <code>errorp</code> is true, an error is signaled. If <code>errorp</code> is false, the function returns an object of type <code>sql-database-error</code>. The default value of <code>errorp</code> is <code>nil</code>.</p> <p>Note: This is a direct call to <code>OCILobFileOpen</code>.</p> <p>Note: this function is available only when the "oracle" module is loaded. See the section on the Oracle LOB interface in the <i>LispWorks User Guide</i> for more information.</p>				
See also	<code>ora-lob-file-close</code>				

ora-lob-file-set-name *Function*

Summary	Sets the name of a file LOB.
Package	<code>sql</code>

This chapter applies to the Enterprise Edition only

Signature	<code>ora-lob-file-set-name <i>file-lob-locator</i> <i>dir-alias</i> <i>name</i> &key <i>errorp</i></code>	
Arguments	<i>file-lob-locator</i>	A file LOB locator.
	<i>dir-alias</i>	A string or <code>nil</code> .
	<i>name</i>	A string or <code>nil</code> .
	<i>errorp</i>	A generalized boolean.
Description	<p>The function <code>ora-lob-file-set-name</code> sets the directory alias and the name of the file LOB pointed to by <i>file-lob-locator</i>. If <i>dir-alias</i> is a string it should be of length no greater than 30. If it is <code>nil</code> then the directory alias of the file LOB is not changed. If <i>name</i> is a string it should be of length no greater than 255. If it is <code>nil</code> then the name of the file LOB is not changed. If an error occurs and <i>errorp</i> is true, an error is signaled. If <i>errorp</i> is false, the function returns an object of type <code>sql-database-error</code>. The default value of <i>errorp</i> is <code>nil</code>.</p>	

Note: This is a direct call to `OCILobFileSetAlias`.

Note: this function is available only when the "oracle" module is loaded. See the section on the Oracle LOB interface in the *LispWorks User Guide* for more information.

		<i>Function</i>
Summary	Flushes the buffer of the Oracle client.	
Package	<code>sql</code>	
Signature	<code>ora-lob-flush-buffer <i>lob-locator</i> &key <i>free-buffer</i> <i>errorp</i></code>	
Arguments	<i>lob-locator</i>	A LOB locator.

	<i>free-buffer</i>	A generalized boolean.
	<i>errorp</i>	A generalized boolean.
Description	<p>The function <code>ora-lob-flush-buffer</code> flushes the buffer that is used by the Oracle client.</p> <p>If <i>free-buffer</i> is true, it also frees the buffer. The default value of <i>free-buffer</i> is <code>nil</code>.</p> <p>If an error occurs and <i>errorp</i> is true, an error is signaled. If <i>errorp</i> is false, the function returns an object of type <code>sql-database-error</code>. The default value of <i>errorp</i> is <code>nil</code>.</p> <p>Note: This is a direct call to <code>OCILobFlushBuffer</code>.</p> <p>Note: this function is available only when the "oracle" module is loaded. See the section on the Oracle LOB interface in the <i>LispWorks User Guide</i> for more information.</p>	
See also	<code>ora-lob-enable-buffering</code>	

		<i>Function</i>
Summary	Frees a LOB locator.	
Package	<code>sql</code>	
Signature	<code>ora-lob-free lob-locator</code>	
Arguments	<i>lob-locator</i>	A LOB locator.
Description	<p>The function <code>ora-lob-free</code> frees the LOB locator <i>lob-locator</i>.</p> <p>If <i>lob-locator</i> was retrieved inside an iteration macro or function (that is, one of <code>map-query</code>, <code>do-query</code>, <code>simple-do-query</code> and <code>loop</code>), it is freed before the next record is fetched, or when terminating the iteration for the last record.</p>	

This chapter applies to the Enterprise Edition only

LOB locators which were retrieved by `select` or `query`, or were created by the user by `ora-lob-assign` or `ora-lob-create-empty` are freed automatically when the database connection is closed by a call to `disconnect`.

If you create many LOB locators without closing the connection, it is useful to free them by calling `ora-lob-free`, to free the resources that are associated with them.

Freeing a LOB locator does not affect the underlying LOB. In particular, after modifications to the LOB there is no `roll-back` even if there was not yet a `commit`.

Note: this function is available only when the "oracle" module is loaded. See the section on the Oracle LOB interface in the *LispWorks User Guide* for more information.

	<i>Function</i>
ora-lob-free-temporary	
Summary	Frees a temporary LOB locator.
Package	<code>sql</code>
Signature	<code>ora-lob-free-temporary temp-lob-locator &key errorp</code>
Arguments	<code>temp-lob-locator</code> A temporary LOB locator. <code>errorp</code> A generalized boolean.
Description	The function <code>ora-lob-free-temporary</code> frees a temporary LOB locator. <code>temp-lob-locator</code> should be a temporary LOB locator as created by <code>ora-lob-create-temporary</code> . If an error occurs and <code>errorp</code> is true, an error is signaled. If <code>errorp</code> is false, the function returns an object of type <code>sql-database-error</code> . The default value of <code>errorp</code> is <code>nil</code> .

Note: temporary LOB locators are freed automatically when the database connection is closed by `disconnect`.

Note: This is a direct call to OCILobFreeTemporary.

Note: this function is available only when the "oracle" module is loaded. See the section on the Oracle LOB interface in the *LispWorks User Guide* for more information.

See also `ora-lob-create-temporary`
 `ora-lob-is-temporary`

ora-lob-get-buffer *Function*

Summary Gets a buffer for efficient I/O with foreign functions.

Package `sql`

Signature `ora-lob-get-buffer lob-locator &key for-writing offset => amount/size, foreign-buffer, eof-or-error-p`

Arguments *lob-locator* A LOB locator.

for-writing A generalized boolean.

offset A non-negative integer or `nil`.

Values *amount/size* A non-negative integer.

foreign-buffer A FLI pointer.

eof-or-error-p A boolean or an error object.

Description The function `ora-lob-get-buffer` gets a buffer for efficient I/O with foreign functions.

If *for-writing* is `nil`, then `ora-lob-get-buffer` fills an internal buffer and returns three values: *amount/size* is how much it filled, *foreign-buffer* points to the actual buffer, and *eof-or-error-p* is the return value from the call to `ora-lob-read-foreign-`

buffer. The offset *offset* is passed directly **ora-lob-read-foreign-buffer**.

If *for-writing* is true, then **ora-lob-get-buffer** returns two values: *amount/size* is the size of the foreign buffer and *foreign-buffer* points to the actual buffer, which then can be passed to **ora-lob-write-foreign-buffer**.

The default value of *for-writing* is **nil**.

The buffer that is used by **ora-lob-get-buffer** is always the same for the LOB locator, it is used by **ora-lob-read-buffer** and **ora-lob-write-buffer**, and is freed automatically when the LOB locator is freed. Thus until you finish with the buffer, you cannot use **ora-lob-read-buffer** or **ora-lob-write-buffer** or call **ora-lob-get-buffer** again or free the LOB locator.

Note: this function is available only when the "oracle" module is loaded. See the section on the Oracle LOB interface in the *LispWorks User Guide* for more information.

Example This first example illustrates reading using the buffer obtained by **ora-lob-get-buffer**. You have a foreign function

```
my_chunk_processor(char *data, int size)  
with this FLI definition  
  
(fli:define-foreign-function my_chunk_processor  
  ((data :pointer)  
   (size :int)))
```

You can pass all the data from the LOB locator to this function. Assuming no other function reads from it, it will start from the beginning.

```
(loop  
  (multiple-value-bind (amount buffer eof-or-error-p)  
      (ora-lob-get-buffer lob)  
    (when (zerop amount) (return))  
    (my_chunk_processor buffer amount )))
```

This second example illustrates writing with the buffer obtained by `ora-lob-get-buffer`. You have a foreign function that fills a buffer with data, and you want to write it to a LOB. First you should lock the record, and if required trim the LOB locator.

```
(multiple-value-bind (size buffer)
  (ora-lob-get-buffer lob-locator
    :for-writing t
    ;; start at the beginning
    :offset 1)
  (loop (let ((amount (my-fill-buffer buffer size)))
    (when (zerop amount) (return))
    (ora-lob-write-foreign-buffer
      lob-locator nil
      amount buffer size))))
```

See also

`ora-lob-read-buffer`
`ora-lob-read-foreign-buffer`
`ora-lob-write-buffer`
`ora-lob-write-foreign-buffer`

ora-lob-get-chunk-size *Function*

Summary Returns the chunk size of a LOB.

Package `sql`

Signature `ora-lob-get-chunk-size lob-locator &key errorp => size`

Arguments `lob-locator` A LOB locator.

`errorp` A generalized boolean.

Values `size` A non-negative integer.

Description The function `ora-lob-get-chunk-size` returns the chunk size of the LOB locator `lob-locator`, which is the best value for the size of a buffer.

If an error occurs and *errorp* is true, an error is signaled. If *errorp* is false, the function returns an object of type **sql-database-error**. The default value of *errorp* is **nil**.

Note: This is a direct call to OCILobGetChunkSize.

Note: this function is available only when the "oracle" module is loaded. See the section on the Oracle LOB interface in the *LispWorks User Guide* for more information.

ora-lob-get-length

Function

Summary Returns the length of a LOB.

Package **sql**

Signature **ora-lob-get-length lob-locator &key errorp => length**

Arguments *lob-locator* A LOB locator.

errorp A generalized boolean.

Values *length* A non-negative integer.

Description The function **ora-lob-get-length** returns the current length of the LOB underlying *lob-locator*.

If an error occurs and *errorp* is true, an error is signaled. If *errorp* is false, the function returns an object of type **sql-database-error**. The default value of *errorp* is **nil**.

Note: This is a direct call to OCILobGetLength.

Note: this function is available only when the "oracle" module is loaded. See the section on the Oracle LOB interface in the *LispWorks User Guide* for more information.

		<i>Function</i>
ora-lob-internal-lob-p		
Summary	The predicate for internal LOBs.	
Package	<code>sql</code>	
Signature	<code>ora-lob-internal-lob-p <i>lob-locator</i> => <i>result</i></code>	
Arguments	<i>lob-locator</i>	A LOB locator.
Values	<i>result</i>	A boolean.
Description	The function <code>ora-lob-internal-lob-p</code> returns <code>t</code> if <i>lob-locator</i> is internal (BLOB, CLOB, or NCLOB). Otherwise it returns <code>nil</code> .	
	Note: this function is available only when the "oracle" module is loaded. See the section on the Oracle LOB interface in the <i>LispWorks User Guide</i> for more information.	

		<i>Function</i>
ora-lob-is-equal		
Summary	The comparison function for LOB locators.	
Package	<code>sql</code>	
Signature	<code>ora-lob-is-equal <i>lob-locator1</i> <i>lob-locator2</i> => <i>result</i></code>	
Arguments	<i>lob-locator1</i>	A LOB locator.
	<i>lob-locator2</i>	A LOB locator.
Values	<i>result</i>	A boolean.
Description	The function <code>ora-lob-is-equal</code> returns <code>t</code> if <i>lob-locator1</i> and <i>lob-locator2</i> point to the same LOB object.	
	Note: This is a direct call to OCILobIsEqual.	

This chapter applies to the Enterprise Edition only

Note: this function is available only when the "oracle" module is loaded. See the section on the Oracle LOB interface in the *LispWorks User Guide* for more information.

ora-lob-is-open *Function*

Summary The predicate for whether a LOB locator is opened.

Package `sql`

Signature `ora-lob-is-open lob-locator &key errorp => result`

Arguments `lob-locator` A LOB locator.

`errorp` A generalized boolean.

Values `result` A boolean.

Description The function `ora-lob-is-open` returns `t` if the LOB pointed to by `lob-locator` is opened (by `ora-lob-open`).

`ora-lob-is-open` is applicable to internal LOBs only.

If an error occurs and `errorp` is true, an error is signaled. If `errorp` is false, the function returns an object of type `sql-database-error`. The default value of `errorp` is `nil`.

Note: This is a direct call to OCILobIsOpen.

Note: this function is available only when the "oracle" module is loaded. See the section on the Oracle LOB interface in the *LispWorks User Guide* for more information.

See also `ora-lob-open`

ora-lob-is-temporary *Function*

Summary The predicate for whether a LOB is temporary.

Package	<code>sql</code>
Signature	<code>ora-lob-is-temporary lob-locator &key errorp => result</code>
Arguments	<code>lob-locator</code> A LOB locator.
	<code>errorp</code> A generalized boolean.
Values	<code>result</code> A boolean.
Description	<p>The function <code>ora-lob-is-temporary</code> returns <code>t</code> if the LOB underlying <code>lob-locator</code> is temporary, that is, it was created by <code>ora-lob-create-temporary</code>.</p> <p>If an error occurs and <code>errorp</code> is true, an error is signaled. If <code>errorp</code> is false, the function returns an object of type <code>sql-database-error</code>. The default value of <code>errorp</code> is <code>nil</code>.</p>
Note: This is a direct call to <code>OCILobIsTemporary</code> .	
Note: this function is available only when the "oracle" module is loaded. See the section on the Oracle LOB interface in the <i>LispWorks User Guide</i> for more information.	
See also	<code>ora-lob-create-temporary</code>

	<i>Function</i>
Summary	Loads data from a file LOB into a LOB.
Package	<code>sql</code>
Signature	<code>ora-lob-load-from-file dest-lob-locator src-lob-file amount &key src-offset dest-offset errorp</code>
Arguments	<code>dest-lob-locator</code> An internal LOB locator.
	<code>src-lob-file</code> A file LOB locator.
	<code>amount</code> A non-negative integer.

This chapter applies to the Enterprise Edition only

<i>src-offset</i>	A non-negative integer.
<i>dest-offset</i>	A non-negative integer.
<i>errorp</i>	A generalized boolean.
Description	<p>The function <code>ora-lob-load-from-file</code> loads the data from the <i>src-lob-file</i> into the destination LOB pointed to by <i>dest-lob-locator</i>.</p> <p>The source LOB must be a BFILE and the destination must be an internal LOB.</p> <p>The details of the operation are determined by <i>amount</i>, <i>src-offset</i> and <i>dest-offset</i>. <i>amount</i> and <i>dest-offset</i> are in characters for CLOB/NCLOB and are in bytes for BLOB. <i>src-offset</i> is in bytes. The offsets start from 1. The default value of <i>dest-offset</i> is 1 and the default value of <i>src-offset</i> is 1.</p> <p>No conversion is performed by <code>ora-lob-load-from-file</code>, so if the destination is a CLOB/NCLOB, the source must already be in the right format.</p> <p>If an error occurs and <i>errorp</i> is true, an error is signaled. If <i>errorp</i> is false, the function returns an object of type <code>sql-database-error</code>. The default value of <i>errorp</i> is <code>nil</code>.</p> <p>Note: This is a direct call to OCILobReadFromFile. The Oracle documentation is ambiguous on whether it is mandatory to open the source LOB before calling this function.</p> <p>Note: this function is available only when the "oracle" module is loaded. See the section on the Oracle LOB interface in the <i>LispWorks User Guide</i> for more information.</p>

See also `ora-lob-copy`

Function
<code>ora-lob-lob-locator</code>
Summary Returns a foreign pointer to the underlying LOB locator.

Package	<code>sql</code>	
Signature	<code>ora-lob-lob-locator lob-locator => pointer</code>	
Arguments	<i>lob-locator</i>	A LOB locator.
Values	<i>pointer</i>	A foreign pointer.
Description	<p>The function <code>ora-lob-lob-locator</code> returns a foreign pointer to the OCI LOB locator underlying <i>lob-locator</i>. <i>pointer</i> is of type <code>sql:p-oci-lob-locator</code> or <code>sql:p-oci-file</code>.</p> <p>Note: this function is available only when the "oracle" module is loaded. See the section on the Oracle LOB interface in the <i>LispWorks User Guide</i> for more information.</p>	

		<i>Function</i>
Summary	The predicate for whether a LOB is initialized.	
Package	<code>sql</code>	
Signature	<code>ora-lob-locator-is-init lob-locator &key errorp => result</code>	
Arguments	<i>lob-locator</i>	A LOB locator.
	<i>errorp</i>	A generalized boolean.
Values	<i>result</i>	A boolean.
Description	<p>The function <code>ora-lob-locator-is-init</code> returns <code>t</code> if the LOB locator <i>lob-locator</i> is initialized. If an error occurs and <i>errorp</i> is true, an error is signaled. If <i>errorp</i> is false, the function returns an object of type <code>sql-database-error</code>. The default value of <i>errorp</i> is <code>nil</code>.</p>	

This chapter applies to the Enterprise Edition only

Note: This is a direct call to OCILobIsInit.

Note: this function is available only when the "oracle" module is loaded. See the section on the Oracle LOB interface in the *LispWorks User Guide* for more information.

ora-lob-open	<i>Function</i>				
Summary	Opens a LOB.				
Package	<code>sql</code>				
Signature	<code>ora-lob-open lob-locator &key errorp</code>				
Arguments	<table><tr><td><i>lob-locator</i></td><td>A LOB locator.</td></tr><tr><td><i>errorp</i></td><td>A generalized boolean.</td></tr></table>	<i>lob-locator</i>	A LOB locator.	<i>errorp</i>	A generalized boolean.
<i>lob-locator</i>	A LOB locator.				
<i>errorp</i>	A generalized boolean.				
Description	<p>The function <code>ora-lob-open</code> opens the LOB pointed to by <i>lob-locator</i>, which can be an internal LOB or a file LOB.</p> <p>Opening the LOB creates a transaction, so any updates associated with modifying the LOB are delayed until the <code>ora-lob-close</code> call. This saves round-trips and avoids extra work on the server side. However it is not mandatory to use <code>ora-lob-open</code>.</p> <p>Calls to <code>ora-lob-open</code> must be strictly paired to calls to <code>ora-lob-close</code>, and the latter must be called before a call to <code>commit</code>. It is also an error to call <code>ora-lob-open</code> on a server LOB object that is already open, even if it has been opened via a different LOB locator.</p> <p>If an error occurs and <i>errorp</i> is true, an error is signaled. If <i>errorp</i> is false, the function returns an object of type <code>sql-database-error</code>. The default value of <i>errorp</i> is <code>nil</code>.</p> <p>Note: This is a direct call to OCILobOpen.</p>				

Note: this function is available only when the "oracle" module is loaded. See the section on the Oracle LOB interface in the *LispWorks User Guide* for more information.

See also

`ora-lob-close`
`ora-lob-is-open`

		<i>Function</i>
Summary	Reads from a LOB into a buffer.	
Package	<code>sql</code>	
Signature		<code>ora-lob-read-buffer lob-locator offset amount buffer &key buffer-offset csid => amount-read, eof-or-error-p</code>
Arguments	<p><i>lob-locator</i></p> <p><i>offset</i></p> <p><i>amount</i></p> <p><i>buffer</i></p> <p><i>buffer-offset</i></p> <p><i>csid</i></p>	<p>A LOB locator.</p> <p>A non-negative integer or <code>nil</code>.</p> <p>A non-negative integer.</p> <p>A string, or a vector of element type <code>(unsigned-byte 8)</code>.</p> <p>A non-negative integer.</p> <p>A Character Set ID</p>
Values	<p><i>amount-read</i></p> <p><i>eof-or-error-p</i></p>	<p>A non-negative integer.</p> <p>A boolean or an error object.</p>
Description	<p>The function <code>ora-lob-read-buffer</code> reads into <i>buffer</i> from the LOB pointed to by <i>lob-locator</i>.</p> <p><i>offset</i> specifies the offset to start reading from. It starts with 1, and specifies characters for CLOB/NCLOB/CFILE and bytes for BLOB/BFILE. If <i>offset</i> is <code>nil</code> then the offset after the end of the previous read operation is used (write operations are</p>	

ignored). This is especially useful for reading linearly from the LOB.

amount is the amount to read, in characters for CLOB/NCLOB/CFILE and bytes for BLOB/BFILE.

The element type of *buffer* should match the element type of the LOB locator (see `ora-lob-element-type`). For this comparison (`unsigned-byte 8`) and `base-char` are considered as the same.

If the buffer *buffer* is not static, there is some additional overhead. For small amounts of data, this is probably insignificant.

buffer-offset specifies where to put the data. It is an offset in bytes from the beginning of the buffer. The default value of *buffer-offset* is 0.

csid specifies what Character Set ID the data in the target buffer should be. It defaults to the CSID of the LOB pointed to by *lob-locator*.

The return value *amount-read* is the number of elements (characters or bytes) that were read.

If the return value *eof-or-error-p* is `nil` then there is still more to read. If *eof-or-error-p* is `t` then it read to the end of the LOB. If an error occurred then *eof-or-error-p* is an error object.

Note: This is a direct call to OCILobRead, without callback.

Note: this function is available only when the "oracle" module is loaded. See the section on the Oracle LOB interface in the *LispWorks User Guide* for more information.

Example

This example sequentially reads the LOB data into a string, starting from offset 10000. It calls a processing function on each chunk of data and then reads in the next chunk starting from where the previous read ended.

```
(let ((my-buffer (make-string 1000
                               :element-type 'base-char))
      (offset 10000))
  (loop
    (let ((nread
           (ora-lob-read-buffer lob-locator
                                 offset
                                 1000
                                 my-buffer)))
      (when (zerop nread) ; end of the LOB
        (return))
      (my-processing-function my-buffer nread))
    (setq offset nil))) ; so next time it continues
                         ; from where it finished
```

See also **ora-lob-element-type**
ora-lob-read-foreign-buffer

ora-lob-read-into-plain-file *Function*

Summary	Writes the contents of a LOB into a file.	
Package	sql	
Signature	ora-lob-read-into-plain-file <i>lob-locator</i> <i>file-name</i> & <i>key</i> <i>offset</i> <i>file-offset</i> <i>if-exists</i>	
Arguments	<i>lob-locator</i>	A LOB locator.
	<i>file-name</i>	A pathname designator.
	<i>offset</i>	A non-negative integer, or nil .
	<i>file-offset</i>	A non-negative integer, or nil .
	<i>if-exists</i>	A keyword or nil .
Description	<p>The function ora-lob-read-into-plain-file writes the contents of a LOB into a file.</p> <p><i>file-name</i> specifies the file to write, which should be a standard file. The file is always opened in a binary mode, so if the</p>	

This chapter applies to the Enterprise Edition only

LOB is a CLOB, the file will be generated in the right format when reading it from the LOB.

offset is the offset into the LOB from where to start reading. It starts from 1, counts characters in a CLOB, and if it is `nil` then the operation starts from the end of the previous read operation. The default value of *offset* is `nil`.

file-offset specifies the offset into the file to start the operation from. If *file-offset* is `nil` then it starts writing at the start of the file. The default value of *file-offset* is `nil`.

if-exists is passed to `open` when opening the file, with the standard Common Lisp meaning. The default value of *if-exists* is `:error`.

Note: this function is available only when the "oracle" module is loaded. See the section on the Oracle LOB interface in the *LispWorks User Guide* for more information.

See also

`ora-lob-write-from-plain-file`

ora-lob-read-foreign-buffer *Function*

Summary Reads from a LOB into a foreign buffer.

Package `sql`

Signature `ora-lob-read-foreign-buffer lob-locator offset amount foreign-buffer buffer-length &key buffer-offset csid => amount-read, eof-or-error-p`

Arguments *lob-locator* A LOB locator.

offset A non-negative integer or `nil`.

amount A non-negative integer.

foreign-buffer A FLI pointer.

buffer-length A non-negative integer.

<i>buffer-offset</i>	A non-negative integer.
<i>csid</i>	A Character Set ID
Values	<p><i>amount-read</i> A non-negative integer.</p> <p><i>eof-or-error-p</i> A boolean or an error object.</p>
Description	<p>The function ora-lob-read-foreign-buffer reads from the LOB pointed to by <i>lob-locator</i> into the foreign buffer <i>foreign-buffer</i>.</p> <p>This is just like ora-lob-read-buffer except that it reads from the LOB locator into a foreign buffer.</p> <p><i>foreign-buffer</i> is a FLI pointer to a buffer, which must be of size at least <i>buffer-length</i>.</p> <p>Note: This is a direct call to OCILobRead, without callback.</p> <p>Note: this function is available only when the "oracle" module is loaded. See the section on the Oracle LOB interface in the <i>LispWorks User Guide</i> for more information.</p>
See also	<p>ora-lob-get-buffer</p> <p>ora-lob-read-buffer</p>

ora-lob-svc-ctx-handle *Function*

Summary	Returns a foreign pointer to the context handle of a LOB.
Package	<code>sql</code>
Signature	<code>ora-lob-svc-ctx-handle lob-locator => pointer</code>
Arguments	<i>lob-locator</i> A LOB locator.
Values	<i>pointer</i> A foreign pointer of type <code>sql:p-oci-svc-ctx</code> .

This chapter applies to the Enterprise Edition only

Description	The function <code>ora-lob-svc-ctx-handle</code> returns a foreign pointer to the context handle of the LOB underlying <i>lob-locator</i> . Note: this function is available only when the "oracle" module is loaded. See the section on the Oracle LOB interface in the <i>LispWorks User Guide</i> for more information.
-------------	--

	<i>Function</i>						
ora-lob-trim							
Summary	Trims an internal LOB.						
Package	<code>sql</code>						
Signature	<code>ora-lob-trim lob-locator new-size &key errorp</code>						
Arguments	<table><tr><td><i>lob-locator</i></td><td>A LOB locator.</td></tr><tr><td><i>new-size</i></td><td>A non-negative integer.</td></tr><tr><td><i>errorp</i></td><td>A generalized boolean.</td></tr></table>	<i>lob-locator</i>	A LOB locator.	<i>new-size</i>	A non-negative integer.	<i>errorp</i>	A generalized boolean.
<i>lob-locator</i>	A LOB locator.						
<i>new-size</i>	A non-negative integer.						
<i>errorp</i>	A generalized boolean.						
Description	<p>The function <code>ora-lob-trim</code> trims the LOB pointed to by <i>lob-locator</i> to a new size <i>new-size</i>, which must be smaller than its current size.</p> <p>Note that <i>new-size</i> is in characters for CLOBs and bytes for BLOBs.</p> <p><code>ora-lob-trim</code> is applicable to internal LOBs only.</p> <p>If an error occurs and <i>errorp</i> is true, an error is signaled. If <i>errorp</i> is false, the function returns an object of type <code>sql-database-error</code>. The default value of <i>errorp</i> is <code>nil</code>.</p> <p>Note: This is a direct call to OCILobTrim.</p> <p>Note: this function is available only when the "oracle" module is loaded. See the section on the Oracle LOB interface in the <i>LispWorks User Guide</i> for more information.</p>						

ora-lob-write-buffer	Function
Summary	Writes a buffer to a LOB.
Package	<code>sql</code>
Signature	<code>ora-lob-write-buffer lob-locator offset amount buffer &key buffer-offset csid => amount-written, eof-or-error-p</code>
Arguments	<p><i>lob-locator</i> A LOB locator.</p> <p><i>offset</i> A non-negative integer or <code>nil</code>.</p> <p><i>amount</i> A non-negative integer.</p> <p><i>buffer</i> A string, or a vector of element type <code>(unsigned-byte 8)</code>.</p> <p><i>buffer-offset</i> A non-negative integer.</p> <p><i>csid</i> A Character Set ID</p>
Values	<p><i>amount-written</i> A non-negative integer.</p> <p><i>eof-or-error-p</i> A boolean or an error object.</p>
Description	<p>The function <code>ora-lob-write-buffer</code> writes to the LOB pointed to by <i>lob-locator</i> from <i>buffer</i>.</p> <p><i>offset</i> specifies the offset to start writing to. It starts with 1, and specifies characters for CLOB/NCLOB/CFILE and bytes for BLOB/BFILE. If <i>offset</i> is <code>nil</code> then the offset after the end of the previous write operation is used (read operations are ignored). This is especially useful for writing linearly to the LOB.</p> <p><i>amount</i> is the amount to write, in characters for CLOB/NCLOB/CFILE and bytes for BLOB/BFILE.</p> <p>The element type of <i>buffer</i> should match the element type of the LOB locator (see <code>ora-lob-element-type</code>). For this comparison (<code>unsigned-byte 8</code>) and <code>base-char</code> are considered as the same.</p>

This chapter applies to the Enterprise Edition only

If the buffer *buffer* is not static, there is some additional overhead. For small amounts of data, this is probably insignificant.

buffer-offset specifies where in the buffer to start writing data from. It is an offset in bytes from the beginning of the buffer. The default value of *buffer-offset* is 0.

csid specifies what Character Set ID the data in the source buffer should be. It defaults to the CSID of the LOB pointed to by *lob-locator*.

The return value *amount-written* is the number of elements (characters or bytes) that were written

The LOB is extended as required.

If the return value *eof-or-error-p* is `nil` then there is still more to write. If *eof-or-error-p* is `t` then it wrote to the end of the LOB. If an error occurred then *eof-or-error-p* is an error object.

Note: The record from which the LOB came must be locked. See the section "Locking" in the *LispWorks User Guide*.

Note: This is a direct call to OCILobWrite, without callback.

Note: this function is available only when the "oracle" module is loaded. See the section on the Oracle LOB interface in the *LispWorks User Guide* for more information.

See also	<code>ora-lob-element-type</code> <code>ora-lob-write-foreign-buffer</code>
----------	--

	<i>Function</i>
Summary	Writes the contents of a file into a LOB.
Package	<code>sql</code>
Signature	<code>ora-lob-write-from-plain-file lob-locator file-name &key offset file-offset if-does-not-exist</code>

Arguments	<i>lob-locator</i>	A LOB locator.
	<i>file-name</i>	A pathname designator.
	<i>offset</i>	A non-negative integer, or <code>nil</code> .
	<i>file-offset</i>	A non-negative integer, or <code>nil</code> .
	<i>if-does-not-exist</i>	A keyword or <code>nil</code> .
Description		<p>The function <code>ora-lob-write-from-plain-file</code> writes the contents of a file into a LOB.</p> <p><i>file-name</i> specifies the file to read, which should be a standard file. The file is always opened in a binary mode, so if the LOB is a CLOB, the file must be in the right format when writing it into the LOB.</p> <p><i>offset</i> is the offset into the LOB from where to start writing. It starts from 1, counts characters in a CLOB, and if it is <code>nil</code> then the operation starts from the end of the previous write operation. The default value of <i>offset</i> is <code>nil</code>.</p> <p><i>file-offset</i> specifies the offset into the file to start the operation from. If <i>file-offset</i> is <code>nil</code> then it starts reading at the start of the file. The default value of <i>file-offset</i> is <code>nil</code>.</p> <p><i>if-does-not-exist</i> is passed to <code>open</code> when opening the file, with the standard Common Lisp meaning. The default value of <i>if-does-not-exist</i> is <code>:error</code>.</p> <p>Note: this function is available only when the "oracle" module is loaded. See the section on the Oracle LOB interface in the <i>LispWorks User Guide</i> for more information.</p>
See also		<code>ora-lob-read-into-plain-file</code>

		<i>Function</i>
Summary	Writes a foreign buffer to a LOB.	

This chapter applies to the Enterprise Edition only

Package	sql														
Signature	<code>ora-lob-write-foreign-buffer lob-locator offset amount foreign-buffer buffer-length &key buffer-offset csid => amount- written, eof-or-error-p</code>														
Arguments	<table><tr><td><i>lob-locator</i></td><td>A LOB locator.</td></tr><tr><td><i>offset</i></td><td>A non-negative integer or <code>nil</code>.</td></tr><tr><td><i>amount</i></td><td>A non-negative integer.</td></tr><tr><td><i>foreign-buffer</i></td><td>A FLI pointer.</td></tr><tr><td><i>buffer-length</i></td><td>A non-negative integer.</td></tr><tr><td><i>buffer-offset</i></td><td>A non-negative integer.</td></tr><tr><td><i>csid</i></td><td>A Character Set ID</td></tr></table>	<i>lob-locator</i>	A LOB locator.	<i>offset</i>	A non-negative integer or <code>nil</code> .	<i>amount</i>	A non-negative integer.	<i>foreign-buffer</i>	A FLI pointer.	<i>buffer-length</i>	A non-negative integer.	<i>buffer-offset</i>	A non-negative integer.	<i>csid</i>	A Character Set ID
<i>lob-locator</i>	A LOB locator.														
<i>offset</i>	A non-negative integer or <code>nil</code> .														
<i>amount</i>	A non-negative integer.														
<i>foreign-buffer</i>	A FLI pointer.														
<i>buffer-length</i>	A non-negative integer.														
<i>buffer-offset</i>	A non-negative integer.														
<i>csid</i>	A Character Set ID														
Values	<table><tr><td><i>amount-written</i></td><td>A non-negative integer.</td></tr><tr><td><i>eof-or-error-p</i></td><td>A boolean or an error object.</td></tr></table>	<i>amount-written</i>	A non-negative integer.	<i>eof-or-error-p</i>	A boolean or an error object.										
<i>amount-written</i>	A non-negative integer.														
<i>eof-or-error-p</i>	A boolean or an error object.														
Description	<p>The function <code>ora-lob-write-foreign-buffer</code> writes to the LOB pointed to by <i>lob-locator</i> from <i>buffer</i>. This is just like <code>ora-lob-write-buffer</code> except that it writes the LOB locator from a foreign buffer. <i>foreign-buffer</i> is a FLI pointer to a buffer, which must be of size at least <i>buffer-length</i>.</p> <p>Note: this function is available only when the "oracle" module is loaded. See the section on the Oracle LOB interface in the <i>LispWorks User Guide</i> for more information.</p>														
See also	<code>ora-lob-get-buffer</code> <code>ora-lob-write-buffer</code>														

print-query	Function												
Summary	Prints a tabulated version of records resulting from a query.												
Package	<code>sql</code>												
Signature	<code>print-query query-exp &key titles formats sizes stream database</code> =>												
Arguments	<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 15%;"><i>query-exp</i></td><td>An SQL query expression.</td></tr> <tr> <td><i>titles</i></td><td>A list of strings.</td></tr> <tr> <td><i>formats</i></td><td>A list of strings.</td></tr> <tr> <td><i>sizes</i></td><td>A list.</td></tr> <tr> <td><i>stream</i></td><td>An output stream.</td></tr> <tr> <td><i>database</i></td><td>A database.</td></tr> </table>	<i>query-exp</i>	An SQL query expression.	<i>titles</i>	A list of strings.	<i>formats</i>	A list of strings.	<i>sizes</i>	A list.	<i>stream</i>	An output stream.	<i>database</i>	A database.
<i>query-exp</i>	An SQL query expression.												
<i>titles</i>	A list of strings.												
<i>formats</i>	A list of strings.												
<i>sizes</i>	A list.												
<i>stream</i>	An output stream.												
<i>database</i>	A database.												
Values	None.												
Description	<p>The <code>print-query</code> function takes a symbolic SQL query expression and formatting information and prints onto <i>stream</i> a table containing the results of the query.</p> <p>A list of strings to use as column headings is given by <i>titles</i>, which has a default value of <code>nil</code>.</p> <p>The <i>formats</i> argument is a list of format strings used to print each attribute, and has a default value of <code>t</code>, which means that <code>~A</code> or <code>~va</code> are used if sizes are provided or computed.</p> <p>The field sizes are given by <i>sizes</i>. It has a default value of <code>t</code>, which specifies that minimum sizes are computed.</p> <p>The output stream is given by <i>stream</i>, which has a default value of <code>t</code>. This specifies that <code>*standard-output*</code> is used.</p>												
Examples	The following call prints out two even columns of names and salaries:												

This chapter applies to the Enterprise Edition only

```
(print-query [select [surname] [income] :from [person]]  
           :titles '("NAME" "SALARY"))
```

See also

[map-query](#)
[print-query](#)
[select](#)

	<i>Function</i>								
query									
Summary	Queries a database and returns a list of values.								
Package	<code>sql</code>								
Signature	<code>query sql-exp &key database result-types flatp => result-list, field-names</code>								
Arguments	<table><tr><td><code>sql-exp</code></td><td>An SQL query statement to be performed.</td></tr><tr><td><code>database</code></td><td>A database.</td></tr><tr><td><code>result-types</code></td><td>A list of symbols.</td></tr><tr><td><code>flatp</code></td><td>A boolean.</td></tr></table>	<code>sql-exp</code>	An SQL query statement to be performed.	<code>database</code>	A database.	<code>result-types</code>	A list of symbols.	<code>flatp</code>	A boolean.
<code>sql-exp</code>	An SQL query statement to be performed.								
<code>database</code>	A database.								
<code>result-types</code>	A list of symbols.								
<code>flatp</code>	A boolean.								
Values	<table><tr><td><code>result-list</code></td><td>A list of values.</td></tr><tr><td><code>field-names</code></td><td>A list of strings.</td></tr></table>	<code>result-list</code>	A list of values.	<code>field-names</code>	A list of strings.				
<code>result-list</code>	A list of values.								
<code>field-names</code>	A list of strings.								
Description	<p>The function <code>query</code> is the basic SQL query function. It queries the database specified by <code>database</code> with an SQL query statement given by <code>sql-exp</code>.</p> <p>The argument <code>database</code> defaults to <code>*default-database*</code>. <code>result-types</code> is a list of symbols such as <code>:string</code> and <code>:integer</code>, one for each field in the query, which are used to specify the types to return.</p> <p><code>flatp</code> is used as in <code>select</code>.</p>								

result-list is a list of values as per `select`, and *field-names* is a list of field names selected in *sql-exp*.

Example The following two queries, on a table whose second column contains dates that we want to return as strings, are equivalent:

```
(sql:query "select * from some_table"
           :result-types '(nil :string))

(sql:query [select [*]
            :from [some_table]
            :result-types '(nil :string)])
```

See also

- [do-query](#)
- [execute-command](#)
- [lob-stream](#)
- [loop](#)
- [map-query](#)
- [select](#)
- [simple-do-query](#)

reconnect	Function						
Summary	Reconnects a database to its underlying RDBMS.						
Package	<code>sql</code>						
Signature	<code>reconnect &key database error force => success</code>						
Arguments	<table> <tr> <td><i>database</i></td><td>The database to be reconnected.</td></tr> <tr> <td><i>error</i></td><td>A boolean.</td></tr> <tr> <td><i>force</i></td><td>A boolean.</td></tr> </table>	<i>database</i>	The database to be reconnected.	<i>error</i>	A boolean.	<i>force</i>	A boolean.
<i>database</i>	The database to be reconnected.						
<i>error</i>	A boolean.						
<i>force</i>	A boolean.						
Values	<i>success</i> A boolean.						

This chapter applies to the Enterprise Edition only

Description	<p>The <code>reconnect</code> function reconnects <i>database</i> to its underlying RDBMS. If successful, <i>success</i> is <code>t</code> and the variable <code>*default-database*</code> is set to the newly reconnected database.</p> <p>The default value for <i>database</i> is <code>*default-database*</code>. If <i>database</i> is a database object, then it is used directly.</p> <p>Otherwise, the list of connected databases is searched to find one with <i>database</i> as its connection specifications (see <code>connect</code>). If no such database is found, then if <i>error</i> and <i>database</i> are both non-<code>nil</code> an error is signaled, otherwise <code>reconnect</code> returns <code>nil</code>.</p> <p><i>force</i> controls whether an error should be signaled if the existing database connection cannot be closed. When non-<code>nil</code> (this is the default value) the connection is closed without error checking. When <i>force</i> is <code>nil</code>, an error is signaled if the database connection has been lost.</p> <p>Note: <i>force non-nil</i> might result in a memory leak if the database driver fails to release its memory (some drivers do not allow the connection to be closed if the underlying RDBMS is not responding).</p>
See also	<code>connect</code> <code>connected-databases</code> <code>*default-database*</code>

restore-sql-reader-syntax-state *Function*

Summary	Sets the enable/disable square bracket syntax state to reflect the last call to either <code>disable-sql-reader-syntax</code> or <code>enable-sql-reader-syntax</code> .
Package	<code>sql</code>
Signature	<code>restore-sql-reader-syntax-state</code>
Arguments	None.

Values	None.
Description	The function <code>restore-sql-reader-syntax-state</code> sets the enable/disable state of the square bracket syntax to reflect the last call to either <code>enable-sql-reader-syntax</code> or <code>disable-sql-reader-syntax</code> . The default state of the square bracket syntax is disabled.
See also	<code>disable-sql-reader-syntax</code> <code>enable-sql-reader-syntax</code> <code>locally-disable-sql-reader-syntax</code> <code>locally-enable-sql-reader-syntax</code>

	<i>Function</i>
rollback	
Summary	Rolls back changes made to a database since the last commit.
Package	<code>sql</code>
Signature	<code>rollback &key database => nil</code>
Arguments	<code>database</code> A database.
Values	<code>nil</code>
Description	The function <code>rollback</code> rolls back changes made in <code>database</code> since the last commit, that is, changes made since the last commit are not recorded. The argument <code>database</code> defaults to <code>*default-database*</code> .
See also	<code>commit</code> <code>with-transaction</code>

This chapter applies to the Enterprise Edition only

	select	<i>Function</i>
Summary		Selects data from a database given a number of specified constraints.
Package		<code>sql</code>
Signature		<pre>select &rest <i>selections</i> &key <i>all</i> <i>set-operation</i> <i>distinct</i> from <i>result-types</i> <i>flatp</i> where <i>group-by</i> having <i>database</i> order-by <i>refresh</i> for-update => <i>result-list</i></pre>
Arguments	<i>selections</i>	A set of database identifiers or strings.
	<i>all</i>	A boolean.
	<i>set-operation</i>	An SQL operation.
	<i>distinct</i>	A boolean.
	<i>from</i>	An SQL table.
	<i>result-types</i>	A list of symbols.
	<i>flatp</i>	A boolean.
	<i>where</i>	An SQL condition.
	<i>group-by</i>	An SQL condition.
	<i>having</i>	An SQL condition.
	<i>database</i>	A database.
	<i>order-by</i>	An SQL condition.
	<i>refresh</i>	A boolean.
	<i>for-update</i>	<code>t</code> , <code>:nowait</code> , a string or a list.
Values	<i>result-list</i>	A list of selections.
Description		The function <code>select</code> selects data from <i>database</i> , which has a default value of <code>*default-database*</code> , given the constraints specified by the rest of the arguments. It returns a list of

objects as specified by *selections*. By default, the objects will each be represented as lists of attribute values.

The argument *selections* consists either of database identifiers, type-modified database identifiers or literal strings.

A type-modified database identifier is an expression such as `[foo :string]` which means that the values in column `foo` are returned as Lisp strings. This syntax can be used to force values in time/date fields to be returned as strings (see below for an example). It can also be used to affect the value returned from MySQL, using the keywords mentioned in the section "Using MySQL" in the *LispWorks User Guide*. It can also be used to return `lob-stream` objects for queries on Oracle LOB columns, using an expression like `[foo :input-stream]` or `[foo :output-stream]`.

result-types is used when *selections* is `*` or `[*]`. It should be a list of symbols such as `:string` and `:integer`, one for each field in the table being selected in order to specify the types to return. Note that, for specific selections, the result type can be specified by using a type-modified identifier as described above. However, you cannot use *result-types* to modify the type returned from a time/date field.

The *flatp* argument, which has a default value of `nil`, specifies if full bracketed results should be returned for each matched entry. If *flatp* is `nil`, the results are returned as a list of lists. If *flatp* is `t`, the results are returned as elements of a list, only if there is only one result per row. See the examples section for an example of the use of *flatp*.

The arguments *all*, *set-operation*, *distinct*, *from*, *where*, *group-by*, *having* and *order-by* have the same function as the equivalent SQL expression.

for-update is used to specify the FOR UPDATE clause in a select statement which is used by Oracle to lock the selected records. If *for-update* is `t` then a plain "FOR UPDATE" clause is generated. This locks all retrieved records, waiting for the

locks to become available. If *for-update* is `:nowait` then a "FOR UPDATE NOWAIT" clause is generated. This locks all the retrieved records, or otherwise returns with error ora-00054 which causes Lisp to signal a `sql-temporary-error`. If *for-update* is a string then it should specify a column to be locked and a clause "FOR UPDATE OF *for-update*" is generated. If *for-update* is a list then the elements of the list should be strings each specifying a column to be locked, except that the last element of the list may be `:nowait`. A clause locking multiple columns is generated, waiting for the locks according to whether `:nowait` was supplied. For an example see the "Locking" section in the LOB section of the *LispWorks User Guide*.

The `select` function is common across both the functional and object-oriented SQL interfaces. If *selections* refers to View Classes then the select operation becomes object-oriented. This means that `select` returns a list of View Class instances, and `slot-value` becomes a valid SQL operator for use within the *where* clause.

In the View Class case, a second equivalent `select` call will return the same View Class instance objects. If *refresh* is true, then existing instances are updated if necessary, and in this case you might need to extend the hook `instance-refreshed`. The default value of *refresh* is `nil`.

SQL expressions used in the `select` function are specified using the square bracket syntax, once this syntax has been enabled using `enable-sql-reader-syntax`.

Examples

The following is a potential query and result:

```
(select [person_id] [surname] :from [person])
=> ((111 "Brown") (112 "Jones") (113 "Smith"))
```

In the next example, the *flatp* argument is set to `t`, and the result is a simple list of surname values:

```
(select [surname] :from [person] :flatp t)
```

```
=> ("Brown" "Jones" "Smith")
```

In this example data in the attribute `largenum`, which is of a vendor-specific large numeric type, is returned to Lisp as strings:

```
(sql:select [largenum :string] :from [my-table])
```

In this example the second column of `some_table` is a date that we want to return as a string:

```
(sql:select [*]
            :from [some_table]
            :result-types '(nil :string))
```

In this example we see that a time/date field value is returned as an integer. We then use Common Lisp to decode that universal time, and finally query the database again, forcing the return value to be a string formatted by the database:

```
CL-USER 219 > (sql:select [MyDate]
                           :from [MyTable]
                           :flatp t)
(3313785600)
("MYDATE")

CL-USER 220 > (decode-universal-time (car *))
0
0
0
4
1
2005
1
NIL
0

CL-USER 221 > (sql:select [MyDate :string]
                           :from [MyTable]
                           :flatp t)
("2005-01-04 00:00:00")
("MYDATE")
```

This chapter applies to the Enterprise Edition only

Finally this code gets the first 1KB of data from the first LOB returned by a query on an Oracle table containing a column of type LOB:

```
(let* ((array
        (make-array 1024
                    :element-type '(unsigned-byte 8)))
       (lobs (sql:select [my-lob-column :input-stream]
                         :from [mytable] :flatp t)))
  (read-sequence array (car lobs)))
```

See also [instance-refreshed](#)
[lob-stream](#)
[print-query](#)

simple-do-query *Macro*

Summary	Repeatedly binds a variable to the results of a query, optionally binds another variable to the column names, and executes a body of code within the scope of these bindings.	
Package	<code>sql</code>	
Signature	<code>simple-do-query (values-list query &key names-list database not-inside-transaction get-all) &body body =></code>	
Arguments	<i>values-list</i>	A variable.
	<i>query</i>	A database query.
	<i>names-list</i>	A variable, or <code>nil</code> .
	<i>database</i>	A database.
	<i>not-inside-transaction</i>	
		A generalized boolean.
	<i>get-all</i>	A generalized boolean.
	<i>body</i>	A Lisp code body.

Values	None.
Description	<p>The macro simple-do-query repeatedly executes <i>body</i> within a binding of <i>values-list</i> to the attributes of each record resulting from <i>query</i>.</p> <p>If a variable <i>names-list</i> is supplied, then it is bound to a list of the column names for the query during the execution of <i>body</i>. The default value of <i>names-list</i> is nil.</p> <p>simple-do-query returns no values.</p> <p>The default value of <i>database</i> is *default-database*.</p> <p><i>not-inside-transaction</i> and <i>get-all</i> may be useful when fetching many records through a connection with <i>database-type</i> :mysql. Both of these arguments have default value nil. See the section "Special considerations for iteration functions and macros" in the <i>LispWorks User Guide</i> for details.</p>
Example	<pre>(sql:simple-do-query (person-details [select [Surname][ID] :from [person]] :names-list xx) (format t "~&~A: ~A, ~A: ~A~%" (first xx) (first person-details) (second xx) (second person-details))) => SURNAME: Brown, ID: 2 SURNAME: Jones, ID: 3 SURNAME: Smith, ID: 4</pre>
See also	do-query loop map-query query select

This chapter applies to the Enterprise Edition only

sql	<i>Function</i>
Summary	Generates SQL from a set of expressions.
Package	<code>sql</code>
Signature	<code>sql &rest args => sql-expression</code>
Arguments	<code>args</code> A set of expressions.
Values	<code>sql-expression</code> An SQL expression.
Description	The function <code>sql</code> generates SQL from a set of expressions given by <code>args</code> . Each argument to <code>sql</code> is translated into SQL and then the <code>args</code> are concatenated with a single space between each pair. The rules for translation into SQL, based on the type of each individual argument <code>x</code> , are as follows: <code>string => (format nil "'~A'" x)</code> <code>nil => "NULL"</code> <code>symbol => (symbol-name x)</code> <code>number => (princ-to-string x)</code> <code>list => (format nil "({~A~^,~})" (mapcar #'sql x))</code> <code>vector => (format nil "~~{~A~^,~}" (map 'list #'sql x))</code> <code>sql-expression => x</code> Any other symbol => error
See also	<code>sql-expression</code> <code>sql-operation</code> <code>sql-operator</code>

sql-connection-error	<i>Condition</i>
Package	<code>sql</code>

Superclasses	<code>sql-database-error</code>
Subclasses	<code>sql-fatal-error</code> <code>sql-timeout-error</code>
Description	The condition class <code>sql-connection-error</code> is used to signal an error with the connection to the database.

sql-database-data-error *Condition*

Package	<code>sql</code>
Superclasses	<code>sql-database-error</code>
Description	The condition class <code>sql-database-data-error</code> is used to signal an error with the data given. This means either a syntax error or things like accessing a non-existent table. It signifies an error that must be fixed for the code to work.

sql-database-error *Condition*

Package	<code>sql</code>
Superclasses	<code>simple-error</code>
Subclasses	<code>sql-connection-error</code> <code>sql-database-data-error</code> <code>sql-temporary-error</code>
Accessors	<code>sql-error-error-id</code> <code>sql-error-secondary-error-id</code> <code>sql-error-database-message</code>
Description	The condition class <code>sql-database-error</code> is used to signal errors in the database interface that Common SQL uses.

This chapter applies to the Enterprise Edition only

sql-error-error-id returns the primary error identifier. On ODBC the value is a string. On Oracle it is some number, the "v2 return code" in the Cursor Data Area.

sql-error-secondary-error-id returns the secondary error identifier. On ODBC this is the error code from the underlying database. On Oracle that is the "v4 return code" (also known as "return code") in the Cursor Data Area, which is the useful code.

sql-error-database-message is a string (maybe `nil`) that came back from the foreign code.

Note: ODBC drivers for Oracle return the "v4 return code" as the underlying database code. Therefore in the event of an error on connection to an Oracle database, **sql-error-secondary-error-id** always returns the "v4 return code" whether the connection is through ODBC.

See also **sql-user-error**

		<i>Function</i>
sql-expression		
Summary		Generates an SQL expression from the given keywords.
Package		<code>sql</code>
Signature		<code>sql-expression &key string table alias attribute type => sql-result</code>
Arguments	<i>string</i>	A string.
	<i>table</i>	A table in a database.
	<i>alias</i>	A table alias.
	<i>attribute</i>	An attribute.
	<i>type</i>	A type.

Values	<i>sql-result</i>	An SQL expression.
Description	The function sql-expression generates an SQL expression from the given keywords. Valid combinations of the arguments are:	
<ul style="list-style-type: none"> • <i>string</i> • <i>table</i> • <i>table</i> and <i>alias</i> • <i>table</i> and <i>attribute</i> • <i>table, attribute, and type</i> • <i>table or alias, and attribute</i> • <i>table or alias, and attribute and type</i> • <i>attribute</i> • <i>attribute and type</i> 		
See also	sql sql-operation sql-operator	

sql-enlarge-static		<i>Variable</i>
Package	sql	
Initial Value	100000	
Description	The amount to enlarge static memory by before loading database code. This is an optimization of static memory fragmentation, useful for some databases. It is ignored when loading Oracle.	
Note: applicable in LispWorks for UNIX only (not LispWorks for Linux or LispWorks for FreeBSD).		

This chapter applies to the Enterprise Edition only

sql-fatal-error *Condition*

Package	<code>sql</code>
Superclasses	<code>sql-connection-error</code>
Description	The condition class <code>sql-fatal-error</code> is used to signal errors that mean the connection can no longer be used.

sql-libraries *Variable*

Package	<code>sql</code>
Initial Value	<code>nil</code>
Description	Holds a pathname or list of libraries to override default database library loading. The value should be a pathname or a list. If its value is a pathname, it is prepended to a list of relative pathnames in the same manner that the supplied environment variable (for example <code>ORACLE_HOME</code>) would be. If its value is a list, then it is assumed to be a complete list of full library names which are loaded verbatim.
Note:	applicable only on Unix/Linux.

sql-loading-verbose *Variable*

Package	<code>sql</code>
Initial Value	<code>nil</code>
Description	The variable <code>*sql-loading-verbose*</code> controls the verbosity of messages while loading the database libraries.
Note:	applicable only on Unix.

sql-operation	<i>Function</i>																
Summary	Generates an SQL statement from an operator and arguments.																
Package	<code>sql</code>																
Signature	$\begin{aligned} \text{sql-operation } & op \&rest\ args \Rightarrow \textit{sql-result} \\ \text{sql-operation } & \text{sql-function } name \&rest\ args \Rightarrow \textit{sql-result} \\ \text{sql-operation } & \text{sql-operator } inop1\ left \&rest\ rights \Rightarrow \textit{sql-result} \\ \text{sql-operation } & \text{sql-boolean-operator } inop2\ left \&rest\ rights \\ & \Rightarrow \textit{sql-result} \end{aligned}$																
Arguments	<table border="0"> <tr> <td><i>op</i></td> <td>An operator.</td> </tr> <tr> <td><i>args</i></td> <td>A set of arguments for <i>op</i>.</td> </tr> <tr> <td><i>name</i></td> <td>An arbitrary function.</td> </tr> <tr> <td><i>args</i></td> <td>A set of arguments for <i>name</i>.</td> </tr> <tr> <td><i>inop1</i></td> <td>An infix operator with non-boolean result.</td> </tr> <tr> <td><i>inop2</i></td> <td>An infix operator that returns a boolean.</td> </tr> <tr> <td><i>left</i></td> <td>Argument to be placed on the left of an infix operator.</td> </tr> <tr> <td><i>rights</i></td> <td>Arguments to be placed on the right of an infix operator.</td> </tr> </table>	<i>op</i>	An operator.	<i>args</i>	A set of arguments for <i>op</i> .	<i>name</i>	An arbitrary function.	<i>args</i>	A set of arguments for <i>name</i> .	<i>inop1</i>	An infix operator with non-boolean result.	<i>inop2</i>	An infix operator that returns a boolean.	<i>left</i>	Argument to be placed on the left of an infix operator.	<i>rights</i>	Arguments to be placed on the right of an infix operator.
<i>op</i>	An operator.																
<i>args</i>	A set of arguments for <i>op</i> .																
<i>name</i>	An arbitrary function.																
<i>args</i>	A set of arguments for <i>name</i> .																
<i>inop1</i>	An infix operator with non-boolean result.																
<i>inop2</i>	An infix operator that returns a boolean.																
<i>left</i>	Argument to be placed on the left of an infix operator.																
<i>rights</i>	Arguments to be placed on the right of an infix operator.																
Values	<i>sql-result</i> An SQL expression.																
Description	<p>The function <code>sql-operation</code> takes an operator and its arguments, and returns an SQL expression.</p> <p><code>(sql-operation op args)</code></p> <p>is shorthand for</p> <p><code>(apply (sql-operator op) args).</code></p>																

This chapter applies to the Enterprise Edition only

The pseudo operator `sql-function` allows an arbitrary function *name* to be passed. In this case, *name* is put in the SQL expression using `princ`, and *args* are given as arguments.

The pseudo operators `sql-boolean-operator` and `sql-operator` generate SQL that calls an infix operator with *left* on the left and *rights* on the right separated by spaces. Use `sql-boolean-operator` for SQL infix operators that return a boolean and use `sql-operator` for any other SQL infix operator.

Note: the pseudo operator `sql-operator` should not be confused with the Common SQL function `sql-operator`.

Example The following code, uses `sql-operation` to produce an SQL expression.

```
(sql-operation 'select
  (sql-expression :table 'foo :attribute 'bar)
  (sql-expression :attribute 'baz)
  :from (list
    (sql-expression :table 'foo)
    (sql-expression :table 'quux))
:where
  (sql-operation 'or
    (sql-operation '>
      (sql-expression :attribute 'baz)
      3)
    (sql-operation 'like
      (sql-expression :table 'foo :attribute 'bar)
      "SU%")))
```

The following SQL expression is produced.

```
#<SQL-QUERY: "(SELECT FOO.BAR,BAZ FROM FOO,QUUX
  WHERE ((BAZ > 3) OR (FOO.BAR LIKE 'SU%')))">
```

The following code illustrates use of the pseudo operator `sql-function`:

```
(sql-operation 'sql-function "TO_DATE" "03/06/99"
  "mm/DD/RR")
```

The following SQL expression is produced.

```
#<SQL-VALUE-EXP "TO_DATE('03/06/99','mm/DD/RR')">
```

See also `sql`
`sql-expression`
`sql-operator`

sql-operator Function

Summary	Returns the symbol for a SQL operator.
Package	<code>sql</code>
Signature	<code>sql-operator symbol => sql-symbol</code>
Arguments	<code>symbol</code> A symbol naming an SQL operator.
Values	<code>sql-symbol</code> A symbol.
Description	The function <code>sql-operator</code> takes an operator as an argument and returns the Lisp symbol for the operator.
See also	<code>sql</code> <code>sql-expression</code> <code>sql-operation</code>

sql-recording-p Function

Summary	A predicate for determining if SQL commands or results traffic is being recorded.
Package	<code>sql</code>
Signature	<code>sql-recording-p &key type database => recording-p</code>
Arguments	<code>type</code> One of <code>:commands</code> or <code>:results</code> .

This chapter applies to the Enterprise Edition only

	<i>database</i>	A database.
Values	<i>recording-p</i>	A boolean.
Description		The function <code>sql-recording-p</code> returns <code>t</code> if <code>type</code> is <code>:commands</code> and SQL commands traffic is being recorded, or if <code>type</code> is <code>:results</code> and SQL results traffic is being recorded. Otherwise it returns <code>nil</code> . The default value of <code>type</code> is <code>:commands</code> . The default value of <code>database</code> is the value of <code>*default-database*</code> .
See also		<code>add-sql-stream</code> <code>delete-sql-stream</code> <code>list-sql-streams</code> <code>sql-stream</code> <code>start-sql-recording</code> <code>stop-sql-recording</code>

		<i>Function</i>
Summary		Returns the broadcast stream used for recording SQL commands or results traffic
Package		<code>sql</code>
Signature		<code>sql-stream &key type database => stream</code>
Arguments	<code>type</code>	One of <code>:commands</code> or <code>:results</code> .
	<code>database</code>	A database.
Values	<code>stream</code>	A broadcast stream.
Description		The function <code>sql-stream</code> returns the broadcast stream used for recording SQL commands or results traffic.

`type` can be either `:commands` or `:results`, and specifies whether to return the broadcast stream for commands or results traffic.

The default value of `type` is `:commands`. The default value of `database` is the value of `*default-database*`.

Note that SQL traffic can appear on `*standard-output*` as well as on `stream`. See `add-sql-stream` for details.

See also

`add-sql-stream`
`delete-sql-stream`
`list-sql-streams`
`sql-recording-p`
`start-sql-recording`
`stop-sql-recording`

sql-temporary-error

Condition

Package `sql`

Superclasses `sql-database-error`

Description The condition class `sql-temporary-error` is used to signal an error that results from other users using the same database. This can be a table lock, but also running out of various resources.

It means the code can work without change, once the other users stop using the database.

sql-timeout-error

Condition

Package `sql`

Superclasses `sql-connection-error`

This chapter applies to the Enterprise Edition only

Description The condition class `sql-timeout-error` is used to signal an error due to the time out of some operation.

sql-user-error *Condition*

Package `sql`

Superclasses `simple-error`

Description The condition class `sql-user-error` is used to signal errors in Lisp code.

See also `sql-database-error`

standard-db-object *Class*

Package `sql`

Superclasses `standard-object`

Description The class `standard-db-object` implements View Classes.

See also `def-view-class`

start-sql-recording *Function*

Summary Starts recording SQL commands or results traffic.

Package `sql`

Signature `start-sql-recording &key type database =>`

Arguments `type` A keyword.

	<i>database</i>	A database.
Values	None.	
Description	The function <code>start-sql-recording</code> starts recording SQL traffic, potentially to multiple streams. The traffic recorded can be the commands, the results, or both commands and results. By default the output appears only <code>*standard-output*</code> . You can modify the broadcast list of recording streams using <code>add-sql-stream</code> and <code>delete-sql-stream</code> . <code>type</code> is one of <code>:commands</code> , <code>:results</code> or <code>:both</code> . It determines whether SQL commands traffic, results traffic or both is recorded.	
		The default value of <code>type</code> is <code>:commands</code> . The default value for <code>database</code> is the value of <code>*default-database*</code> .
See also		<code>add-sql-stream</code> <code>delete-sql-stream</code> <code>list-sql-streams</code> <code>sql-stream</code> <code>sql-recording-p</code> <code>stop-sql-recording</code>

		<i>Function</i>
Summary		Returns status information for the connected databases and initialized database types.
Package	<code>sql</code>	
Signature		<code>status &optional full =></code>
Arguments	<code>full</code>	A boolean.

This chapter applies to the Enterprise Edition only

Values	None.
Description	<p>The function status prints status information to the standard output, for the connected databases and initialized database types.</p> <p>If <i>full</i> is <code>t</code>, detailed status information is printed. The default value of <i>full</i> is <code>nil</code>.</p>
See also	<code>connect</code> <code>connected-databases</code> <code>database-name</code> <code>disconnect</code> <code>find-database</code>

stop-sql-recording *Function*

Summary	Stops recording SQL commands or results traffic.	
Package	<code>sql</code>	
Signature	<code>stop-sql-recording &key type database =></code>	
Arguments	<p><i>type</i> A keyword. <i>database</i> A database.</p>	
Values	None.	
Description	<p>The function stop-sql-recording stops recording SQL commands or results traffic.</p> <p><i>type</i> is one of <code>:commands</code>, <code>:results</code> or <code>:both</code>. It determines whether the recording of SQL commands traffic, results traffic or both is stopped.</p> <p>The default value of <i>type</i> is <code>:commands</code>. The default value for <i>database</i> is <code>*default-database*</code>.</p>	

See also [add-sql-stream](#)
[delete-sql-stream](#)
[list-sql-streams](#)
[sql-recording-p](#)
[sql-stream](#)
[start-sql-recording](#)

	<i>Function</i>						
table-exists-p							
Summary	A predicate for the existence of a table.						
Package	<code>sql</code>						
Signature	<code>table-exists-p <i>table</i> &key <i>database owner</i> => <i>result</i></code>						
Arguments	<table> <tr> <td><i>table</i></td><td>A potential table name.</td></tr> <tr> <td><i>database</i></td><td>A database.</td></tr> <tr> <td><i>owner</i></td><td><code>nil</code>, <code>:all</code> or a string.</td></tr> </table>	<i>table</i>	A potential table name.	<i>database</i>	A database.	<i>owner</i>	<code>nil</code> , <code>:all</code> or a string.
<i>table</i>	A potential table name.						
<i>database</i>	A database.						
<i>owner</i>	<code>nil</code> , <code>:all</code> or a string.						
Values	<i>result</i> A boolean.						
Description	<p>The function <code>table-exists-p</code> determines whether there is a table named <i>table</i> in database <i>database</i>.</p> <p>If <i>owner</i> is <code>nil</code>, only user-owned tables are considered. This is the default.</p> <p>If <i>owner</i> is <code>:all</code>, all tables are considered.</p> <p>If <i>owner</i> is a string, this denotes a username and only tables owned by <i>owner</i> are considered.</p> <p>The default value of <i>database</i> is <code>*default-database*</code>.</p>						
See also	list-tables						

This chapter applies to the Enterprise Edition only

update-instance-from-records		<i>Generic Function</i>
Summary	Updates a View Class instance.	
Package	<code>sql</code>	
Signature	<code>update-instance-from-records <i>instance</i> &key <i>database</i> => <i>instance</i></code>	
Arguments	<code><i>instance</i></code>	An instance of a View Class.
	<code><i>database</i></code>	A database.
Values	<code><i>instance</i></code>	The updated View Class instance.
Description	<p>The generic function <code>update-instance-from-records</code> updates the values in the slots of the View Class instance <code>instance</code> using the data in the database <code>database</code>.</p> <p><code>database</code> defaults to the database that <code>instance</code> is associated with, or the value of <code>*default-database*</code>. If <code>instance</code> is associated with a database, then <code>database</code> must be that same database.</p> <p>The argument <code>slot</code> is the CLOS slot name; the corresponding column names are derived from the View Class definition.</p> <p>The update is not recursive on joins. Join slots (that is, slots with <code>:db-kind :join</code>) are updated, but the joined objects are not updated.</p>	
See also	<code>def-view-class</code> <code>update-slot-from-record</code>	

update-objects-joins		<i>Function</i>
Summary	Updates the remote join slots.	

Signature	<code>update-objects-joins objects &key slots force-p class-name max-len</code>	
Arguments	<i>objects</i>	A list of database objects.
	<i>slots</i>	A list of slot names, or <code>t</code> .
	<i>force-p</i>	A boolean.
	<i>class-name</i>	The class of the objects, or <code>nil</code> .
	<i>max-len</i>	A non-negative integer, or <code>nil</code> .
Description	<p>The function <code>update-objects-joins</code> updates the remote join slots, that is those slots defined without <code>:retrieval :immediate</code>.</p> <p>This is an optimization function which can improve the efficiency of an application by reducing the number of queries of the database. For each slot, it queries the database using the data from all the objects, and then assigns the appropriate value to each object.</p> <p><i>objects</i> is a list of database objects. If <i>class-name</i> is non-<code>nil</code>, then all the database objects are of this class. If <i>class-name</i> is <code>nil</code>, then all the database objects are of the class of the first database object in the list <i>objects</i>.</p> <p>If <i>objects</i> is <code>nil</code>, then <code>update-objects-joins</code> does nothing.</p> <p><i>class-name</i> specifies a class containing all the database objects in the list <i>objects</i>. If <i>class-name</i> is <code>nil</code> (the default) then the class of the first database object is used.</p> <p><i>slots</i> provides a list of the names of slots to update. Each of these slots should be a remote join slot (as defined above).</p> <p><i>slots</i> can also be <code>t</code>, meaning update all the remote join slots. The default value of <i>slots</i> is <code>t</code>.</p> <p><i>force-p</i> controls whether to force the update of all values in the objects. If <i>force-p</i> is <code>nil</code>, then slots which are already are not updated. The default value of <i>force-p</i> is <code>t</code>.</p>	

This chapter applies to the Enterprise Edition only

max-len, if non-*nil*, is a maximum number of objects from which to use data in a single query. If the length of the list *objects* is greater than *max-len* then `update-objects-joins` performs multiple queries using the data from no more than *max-len* objects in each query. This is useful if the DBMS may reject large queries, but it will increase the number of queries and hence reduce overall performance to some extent. The default value of *max-len* is the value of the variable `*default-update-objects-max-len*`.

See also `*default-update-objects-max-len*`
 `def-view-class`

		<i>Function</i>
	update-records	
Summary		Changes the values of fields in a table.
Package		<code>sql</code>
Signature		<code>update-records table &key attributes values av-pairs where database =></code>
Arguments	<i>table</i>	A database table.
	<i>attributes</i>	A set of columns.
	<i>values</i>	A set of values.
	<i>av-pairs</i>	An association list alternative to <i>attributes</i> and <i>values</i> .
	<i>where</i>	A condition.
	<i>database</i>	A database.
Values		None.

Description	The function <code>update-records</code> changes the values of existing fields in <i>table</i> with columns specified by <i>attributes</i> and <i>values</i> (or <i>av-pairs</i>) where the <i>where</i> condition is true.
See also	<code>delete-instance-records</code> <code>delete-records</code> <code>insert-records</code> <code>update-records-from-instance</code>

update-records-from-instance

Generic Function

Summary	Updates a set of specified records in a database.				
Package	<code>sql</code>				
Signature	<code>update-records-from-instance instance &key database =></code>				
Arguments	<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 25%;"><i>instance</i></td><td>An instance of a View Class.</td></tr> <tr> <td><i>database</i></td><td>A database.</td></tr> </table>	<i>instance</i>	An instance of a View Class.	<i>database</i>	A database.
<i>instance</i>	An instance of a View Class.				
<i>database</i>	A database.				
Values	None.				
Description	<p>The generic function <code>update-records-from-instance</code> updates the records in <i>database</i> represented by <i>instance</i>. If the instance is already associated with a database, that database is used, and <i>database</i> is ignored. If <i>instance</i> is not yet associated with a database, a record is created for <i>instance</i> in the appropriate table of <i>database</i> and the instance becomes associated with that database.</p> <p><code>update-records-from-instance</code> only updates the records from the base slots of <i>instance</i> - it doesn't look at the join slots.</p>				
See also	<code>def-view-class</code> <code>delete-instance-records</code> <code>update-records</code>				

This chapter applies to the Enterprise Edition only

update-record-from-slot

Generic Function

Summary	Updates an individual data item from a slot.	
Package	<code>sql</code>	
Signature	<code>update-record-from-slot instance slot &key database</code>	
Arguments	<i>instance</i>	An instance of a View Class.
	<i>slot</i>	A slot.
	<i>database</i>	A database.
Values	None.	
Description	The generic function <code>update-record-from-slot</code> updates an individual data item in the column represented by <i>slot</i> . The <i>database</i> is only used if <i>instance</i> is not yet associated with any database, in which case a record is created in <i>database</i> . Only <i>slot</i> is initialized in this case; other columns in the underlying database receive default values. The argument <i>slot</i> is the CLOS slot name; the corresponding column names are derived from the View Class definition.	
See also	<code>def-view-class</code> <code>update-records-from-instance</code>	

update-slot-from-record

Generic Function

Summary	Updates a slot in a View Class instance.	
Package	<code>sql</code>	
Signature	<code>update-slot-from-record instance slot => instance</code>	
Arguments	<i>instance</i>	An instance of a View Class.

	<i>slot</i>	A slot name.
Values	<i>instance</i>	The updated View Class instance.
Description		<p>The generic function <code>update-slot-from-record</code> updates the value in the slot <i>slot</i> of the View Class instance <i>instance</i> using the records in the database.</p> <p><i>instance</i> must be associated with a database.</p> <p>The argument <i>slot</i> is the CLOS slot name; the corresponding column names are derived from the View Class definition.</p> <p>The update is not recursive on joins. Join slots (that is, slots with <code>:db-kind :join</code>) are updated, but the joined objects are not updated.</p>
See also		<p><code>def-view-class</code></p> <p><code>update-instance-from-records</code></p>

with-transaction

Macro

Summary	Performs a body of code within a transaction for a database.	
Package	<code>sql</code>	
Signature	<code>with-transaction &key database &body body => results</code>	
Arguments	<i>database</i>	A database.
	<i>body</i>	A set of Lisp expressions.
Values	<i>results</i>	The values returned by <i>body</i> .
Description	<p>The macro <code>with-transaction</code> executes <i>body</i> within a transaction for <i>database</i> (which defaults to <code>*default-database*</code>). The transaction is committed if the body finishes successfully.</p>	

This chapter applies to the Enterprise Edition only

(without aborting or throwing), otherwise the database is rolled back.

with-transaction returns the value or multiple values returned from *body*.

Example The following example shows how to use **with-transaction** to insert a new record, updates the department number of employes from 40 to 50, and removes employees whose salary is higher than 300,000. If an error occurs anywhere in the body and an **abort** or **throw** is executed, none of the updates are committed.

```
(with-transaction
  (insert-record :into [emp]
                  :attributes '(x y z)
                  :values '(a b c))
  (update-records [emp]
                  :attributes [dept]
                  :values 50
                  :where [= [dept] 40])
  (delete-records :from [emp]
                  :where [> [salary] 300000]))
```

See also

commit
rollback

15

The STREAM Package

This chapter describes the symbols available in the `stream` package that provide users with the functionality to define their own streams for use by the standard I/O functions.

buffered-stream *Class*

Summary	A stream class giving access to stream buffers.	
Package	<code>stream</code>	
Superclasses	<code>fundamental-stream</code>	
Subclasses	<code>lob-stream</code> <code>string-stream</code> <code>socket-stream</code>	
Initargs	<code>:direction</code>	One of <code>:input</code> , <code>:output</code> or <code>:io</code> . This argument is required.
	<code>:element-type</code>	One of <code>base-char</code> , <code>simple-char</code> or <code>character</code> .

Description	<p>The class <code>buffered-stream</code> provides default methods for the majority of the functions in the User Defined Streams protocol. The default methods implement buffered I/O, requiring the user to define only the methods <code>stream-read-buffer</code>, <code>stream-write-buffer</code> and <code>stream-element-type</code> for each subclass of <code>buffered-stream</code>. You are at liberty to redefine other methods in subclasses as long as they obey the rules outlined here. For example it is usually desirable to implement methods on <code>stream-listen</code>, <code>stream-check-eof-no-hang</code> and <code>close</code> as well.</p> <p>The initargs are handled by the method (<code>method initialize-instance :after (buffered-stream)</code>) as follows:</p> <p>Input and/or output buffers are created based on the value <i>direction</i>. There is no default value, and you must supply a value.</p> <p><i>element-type</i> determines the <code>stream-element-type</code> of the stream. The default is <code>base-char</code>. For binary streams, use <code>base-char</code>.</p> <p>All the methods in the User Defined Streams protocol are defined for <code>buffered-stream</code> as follows:</p> <ul style="list-style-type: none"> • The methods on <code>stream-read-char</code>, <code>stream-read-line</code>, <code>stream-read-sequence</code>, <code>stream-unread-char</code>, <code>stream-read-char-no-hang</code>, <code>stream-clear-input</code> handle input from the buffer. They each call <code>stream-fill-buffer</code> to fill the empty buffer as required. • The methods on <code>stream-write-char</code>, <code>stream-write-string</code>, <code>stream-write-sequence</code>, <code>stream-clear-output</code>, <code>stream-finish-output</code>, <code>stream-force-output</code> and <code>stream-line-column</code> handle output to the buffer. They each call <code>stream-flush-buffer</code> to make the buffer empty as required. • There are <code>:around</code> methods on <code>stream-listen</code> and <code>close</code> which handle the buffer.
-------------	--

- The methods on `input-stream-p`, `output-stream-p` return the appropriate values based on the value of the `:direction` initarg.
- The `open-stream-p` method returns true if `close` has not been called.

Example See the extended example in `examples/streams/buffered-stream.lisp`

See also `close`
`stream-flush-buffer`
`stream-fill-buffer`
`stream-listen`
`stream-read-buffer`
`stream-write-buffer`
`with-stream-input-buffer`

fundamental-binary-input-stream *Class*

Summary A stream class for binary input.

Package `stream`

Superclasses `fundamental-binary-stream`
`fundamental-input-stream`

Subclasses None.

Description The class `fundamental-binary-input-stream` provides a class for generating customized binary input stream classes. A method for `stream-read-byte` should be provided when using this class.

See also `fundamental-binary-stream`
`fundamental-input-stream`
`stream-read-byte`

fundamental-binary-output-stream *Class*

Summary	A stream class for binary output.
Package	<code>stream</code>
Superclasses	<code>fundamental-binary-stream</code> <code>fundamental-output-stream</code>
Description	The class <code>fundamental-binary-output-stream</code> provides a class for generating customized binary output stream classes. A method for <code>stream-write-byte</code> should be provided.
See also	<code>fundamental-binary-stream</code> <code>fundamental-output-stream</code> <code>stream-write-byte</code>

fundamental-binary-stream *Class*

Summary	A class for binary streams.
Package	<code>stream</code>
Superclasses	<code>fundamental-stream</code>
Subclasses	<code>fundamental-binary-input-stream</code> <code>fundamental-binary-output-stream</code>
Description	The class <code>fundamental-binary-stream</code> is the superclass of the binary input and output stream classes. A method for <code>stream-element-type</code> should be provided for instantiable subclasses of this class.
See also	<code>fundamental-binary-input-stream</code> <code>fundamental-binary-output-stream</code> <code>fundamental-stream</code> <code>stream-element-type</code>

fundamental-character-input-stream *Class*

Summary	A class that should be included in stream classes for character input.
Package	<code>stream</code>
Superclasses	<code>fundamental-character-stream</code> <code>fundamental-input-stream</code>
Subclasses	None.
Description	The class <code>fundamental-character-input-stream</code> provides default methods for generic functions used for character input, and should therefore be included by stream classes concerned with character input. The user can provide methods for these generic functions specialized on the user-defined class. Methods for other generic functions must be provided by the user.
See also	<code>fundamental-character-stream</code> <code>fundamental-input-stream</code> <code>stream-clear-input</code> <code>stream-listen</code> <code>stream-peek-char</code> <code>stream-read-char</code> <code>stream-read-char-no-hang</code> <code>stream-read-line</code> <code>stream-read-sequence</code> <code>stream-unread-char</code>

fundamental-character-output-stream *Class*

Summary	A class that should be included in stream classes for character output.
---------	---

Package	<code>stream</code>
Superclasses	<code>fundamental-character-stream</code> <code>fundamental-output-stream</code>
Subclasses	None.
Description	The class <code>fundamental-character-output-stream</code> provides default methods for generic functions used for character output, and should therefore be included by stream classes concerned with character output. The user can provide methods for these generic functions specialized on the user-defined class. Methods for other generic functions must be provided by the user.
See also	<code>fundamental-character-stream</code> <code>fundamental-input-stream</code> <code>stream-clear-output</code> <code>stream-finish-output</code> <code>stream-force-output</code> <code>stream-start-line-p</code> <code>stream-terpri</code> <code>stream-line-column</code> <code>stream-write-char</code> <code>stream-write-sequence</code> <code>stream-write-string</code>

fundamental-character-stream*Class*

Summary	A class whose inclusion provides a method for <code>stream-element-type</code> that returns <code>character</code> .
Package	<code>stream</code>
Superclasses	<code>fundamental-stream</code>

Subclasses	<code>fundamental-character-input-stream</code> <code>fundamental-character-output-stream</code>
Description	The class <code>fundamental-character-stream</code> is a superclass for character streams. Its inclusion provides a method for the generic function <code>stream-element-type</code> that returns the symbol <code>character</code> .
See also	<code>fundamental-character-input-stream</code> <code>fundamental-character-output-stream</code> <code>fundamental-stream</code> <code>stream-element-type</code>

fundamental-input-stream *Class*

Summary	A class whose inclusion causes <code>input-stream-p</code> to return <code>t</code> .
Package	<code>stream</code>
Superclasses	<code>fundamental-stream</code>
Subclasses	<code>fundamental-binary-input-stream</code> <code>fundamental-character-input-stream</code>
Description	The <code>fundamental-input-stream</code> class is a superclass to the binary and character input classes. Its inclusion causes the generic function <code>input-stream-p</code> to return <code>t</code> .
See also	<code>fundamental-binary-input-stream</code> <code>fundamental-character-input-stream</code> <code>fundamental-stream</code> <code>input-stream-p</code>

fundamental-output-stream *Class*

Summary	A class whose inclusion causes <code>output-stream-p</code> to return <code>t</code> .
Package	<code>stream</code>
Superclasses	<code>fundamental-stream</code>
Subclasses	<code>fundamental-binary-output-stream</code> <code>fundamental-character-output-stream</code>
Description	The <code>fundamental-output-stream</code> class is a superclass to the binary and character output classes. Its inclusion causes the generic function <code>output-stream-p</code> to return <code>t</code> .
See also	<code>fundamental-binary-output-stream</code> <code>fundamental-character-output-stream</code> <code>fundamental-stream</code> <code>input-stream-p</code>

fundamental-stream *Class*

Summary	A class whose inclusion causes <code>stream-p</code> to return <code>t</code> .
Package	<code>stream</code>
Superclasses	<code>standard-object</code> <code>stream</code>
Subclasses	<code>fundamental-binary-stream</code> <code>fundamental-character-stream</code> <code>fundamental-input-stream</code> <code>fundamental-output-stream</code>
Description	The class <code>fundamental-stream</code> is a superclass to the fundamental input, output, character and binary streams. Its inclusion causes <code>stream-p</code> to return <code>t</code> .

See also **close**
fundamental-binary-stream
fundamental-character-stream
fundamental-input-stream
fundamental-output-stream
open-stream-p

stream-advance-to-column

Generic Function

Summary	Writes the required number of blank spaces to ensure that the next character will be written in a given column.	
Package	stream	
Signature	stream-advance-to-column <i>stream column => result</i>	
Arguments	<i>stream</i>	A stream.
	<i>column</i>	An integer.
Values	<i>result</i>	A boolean.
Description	The generic function stream-advance-to-column writes enough blank spaces to <i>stream</i> to ensure that the next character is written at <i>column</i> . The generic function returns t if the operation is successful, or nil if it is not supported for this stream. This function is intended for use by print and format ~t . The default method uses stream-line-column and repeated calls to stream-write-char with a #\space character, and returns nil if stream-line-column returns nil .	
See also	stream-line-column	

stream-check-eof-no-hang

Generic Function

Summary	Determines whether a stream is at end of file.	
Package	stream	
Signature	stream-check-eof-no-hang <i>stream</i> => <i>result</i>	
Arguments	<i>stream</i>	An input stream.
Values	<i>result</i>	nil or :eof .
Description	<p>The generic function stream-check-eof-no-hang determines if the data source of the stream is at end of file, without hanging.</p> <p><i>stream</i> should be an instance of a subclass of buffered-stream.</p> <p><i>result</i> is :eof if <i>stream</i> is at end of file and nil otherwise.</p> <p>There is a built-in method specialized on buffered-stream which returns :eof in all cases.</p>	
See also	buffered-stream	

stream-clear-input

Generic Function

Summary	Implements clear-input .	
Package	stream	
Signature	stream-clear-input <i>stream</i> => nil	
Arguments	<i>stream</i>	A stream.
Values	nil	

Description The generic function `stream-clear-input` implements `clear-input`. The default method is defined on `fundamental-input-stream` and does nothing.

See also `fundamental-input-stream`

stream-clear-output

Generic Function

Summary Implements `clear-output`.

Package `stream`

Signature `stream-clear-output stream => nil`

Arguments `stream` A stream.

Values `nil`

Description The generic function `stream-clear-output` implements `clear-output`. The default method is on `fundamental-output-stream` and does nothing.

See also `fundamental-output-stream`

stream-file-position

Generic Function

Summary Returns or changes the current position within a stream.

Package `stream`

Signature `stream-file-position stream => position`

Signature `(setf stream-file-position) position-spec stream => success-p`

Arguments `stream` A stream.

	<i>position-spec</i>	A file position designator.
Values	<i>position</i>	A file position or <code>nil</code> .
	<i>success-p</i>	A generalized boolean.
Description		<p>The generic function <code>stream-file-position</code> implements <code>file-position</code>. <code>stream-file-position</code> is called when <code>file-position</code> is called with one argument. <code>(setf stream:stream-file-position)</code> is called when <code>file-position</code> is called with two arguments.</p> <p>The return value is returned by <code>file-position</code>. For the <code>setf</code> function, this is a slight anomaly because <code>setf</code> functions normally return the new value. However in this case it should return the <i>success-p</i> value mandated by the ANSI Common Lisp standard.</p> <p>The default methods specialized on <code>stream</code> return <code>nil</code>.</p>

		<i>Generic Function</i>
Summary		Fills the stream buffer.
Package		<code>stream</code>
Signature		<code>stream-fill-buffer stream => result</code>
Arguments	<i>stream</i>	An input stream.
Values	<i>result</i>	A generalized boolean.
Description		The generic function <code>stream-fill-buffer</code> is called by the reading functions to fill an empty stream buffer from the underlying data source.

stream should be an instance of a subclass of **buffered-stream**.

stream-fill-buffer should block until some data is available or return false at end of file. If data is available, it should place it in a buffer, set the stream's input buffer, index and limit appropriately and return a true value. The existing stream buffer can be reused if desired but the index and limit must be updated. The buffer must be of type **simple-string**, whose element type matches that given when the stream was constructed.

There is a built-in method specialized on **buffered-stream** which usually suffices. It calls **stream-read-buffer** with the whole buffer and returns false if this call returns 0. If not, the input index is set to 0 and the input limit is set to the value returned by **stream-read-buffer**.

See also

buffered-stream
stream-read-buffer

stream-finish-output

Generic Function

Summary Implements **finish-output**.

Package **stream**

Signature **stream-finish-output stream => nil**

Arguments *stream* A stream.

Values **nil**

Description The generic function **stream-finish-output** implements **finish-output**. The default method is on **fundamental-output-stream** and does nothing.

See also [fundamental-output-stream](#)

stream-flush-buffer		Generic Function
Summary	Flushes a stream's buffer.	
Package	<code>stream</code>	
Signature	<code>stream-flush-buffer stream => result</code>	
Arguments	<code>stream</code>	An output stream.
Values	<code>result</code>	A generalized boolean.
Description	<p>The generic function <code>stream-flush-buffer</code> is called by the writing functions to flush a stream buffer to the underlying data sink.</p> <p><code>stream</code> should be an instance of a subclass of <code>buffered-stream</code>.</p> <p>Before returning, <code>stream-flush-buffer</code> must set the output index of <code>stream</code> so that more characters can be written to the buffer. If desired, the output buffer and limit can be set too.</p> <p>There is a built-in method specialized on <code>buffered-stream</code> which usually suffices. It calls <code>stream-write-buffer</code> with the currently active part of the stream's output buffer and sets the output index to 0.</p> <p><code>result</code> is true if the buffer was flushed.</p>	
See also	<code>buffered-stream</code>	

stream-force-output*Generic Function*

Summary Implements **force-output**.

Package **stream**

Signature **stream-force-output stream => nil**

Arguments *stream* A stream.

Values **nil**

Description The generic function **stream-force-output** implements **force-output**. The default method is on **fundamental-output-stream** and does nothing.

See also **fundamental-output-stream**

stream-fresh-line*Generic Function*

Summary Used by **fresh-line** to start a new line on a given stream.

Package **stream**

Signature **stream-fresh-line stream => bool**

Arguments *stream* A stream.

Values *bool* A generalized boolean.

Description The generic function **stream-fresh-line** is used by **fresh-line** to start a new line on a stream. The default method uses **stream-start-line-p** and **stream-terpri**. The result value is **t** if a new line is output successfully.

See also [stream-start-line-p](#)
[stream-terpri](#)

stream-line-column

Generic Function

Summary Returns the column number where the next character will be written.

Package stream

Signature stream-line-column *stream* \Rightarrow *column*

Arguments *stream* A stream.

Values *column* An integer.

Description	The generic function <code>stream-line-column</code> returns the column number where the next character will be written from <i>stream</i> , or <code>nil</code> if this is not meaningful for the stream. This function is used in the implementation of <code>print</code> and the <code>format ~t</code> directive. A method for this function must be defined for every character output stream class that is defined, although at its simplest it may be defined to always return <code>nil</code> .
-------------	---

See also [fundamental-character-output-stream](#)
[stream-start-line-p](#)

stream-listen

Generic Function

Summary A function used by `listen` that returns `t` if there is input available.

Package stream

Signature	<code>stream-listen stream => result</code>	
Arguments	<code>stream</code>	A stream.
Values	<code>result</code>	A generalized boolean.
Description	<p>The generic function <code>stream-listen</code> is called to determine if there is data immediately available on the stream <code>stream</code>, without hanging.</p> <p><code>result</code> should be true if there is input, and <code>nil</code> otherwise (including at end of file).</p> <p>This method must be implemented for subclasses of <code>buffered-stream</code> that handle input.</p> <p>There is a built-in primary method specialized on <code>buffered-stream</code> which returns <code>nil</code>. There is a built-in <code>:around</code> method specialized on <code>buffered-stream</code> which checks for input in the buffer and calls the next method if the buffer is empty. Thus a primary method specialized on a subclass of <code>buffered-stream</code> need only check the underlying data source.</p> <p>The built-in method on <code>fundamental-input-stream</code> uses <code>stream-read-char-no-hang</code> and <code>stream-unread-char</code>. Most streams should define their own method as this is usually trivial and more efficient than the method provided.</p>	
See also	<code>buffered-stream</code> <code>stream-read-char-no-hang</code> <code>stream-unread-char</code>	

stream-output-width

Generic Function

Summary	Used by the pretty printer to determine the output width when <code>*print-right-margin*</code> is <code>nil</code> .
Package	<code>stream</code>

Signature	<code>stream-output-width stream => result</code>	
Arguments	<code>stream</code>	A stream.
Values	<code>result</code>	An integer or <code>nil</code> .
Description	The generic function <code>stream-output-width</code> is used by the pretty printer to determine the output width when <code>*print-right-margin*</code> is <code>nil</code> . It returns <code>result</code> , the integer width of <code>stream</code> in units of ems, or <code>nil</code> if the width is not known. The default method provided by <code>fundamental-stream</code> returns <code>nil</code> .	
See also	<code>fundamental-stream</code>	

stream-peek-char

Generic Function

Summary	A generic function used by <code>peek-char</code> that returns a character on a given stream without removing it from the stream buffer.	
Package	<code>stream</code>	
Signature	<code>stream-peek-char stream => result</code>	
Arguments	<code>stream</code>	A stream.
Values	<code>result</code>	A character or <code>:EOF</code> symbol.
Description	The generic function <code>stream-peek-char</code> is used to implement <code>peek-char</code> , and corresponds to a peek-type of <code>nil</code> . The default method reads a character from the stream without removing it from the stream buffer, by using <code>stream-read-char</code> and <code>stream-unread-char</code> .	

See also **stream-listen**
 stream-read-char
 stream-unread-char

stream-read-buffer

Generic Function

Summary	Reads data into the stream buffer.	
Package	stream	
Signature	stream-read-buffer <i>stream</i> <i>buffer</i> <i>start</i> <i>end</i> => <i>result</i>	
Arguments	<i>stream</i>	An input stream.
	<i>buffer</i>	A stream buffer.
	<i>start, end</i>	Bounding indexes for a subsequence of <i>buffer</i> .
Values	<i>result</i>	A non-negative integer.
Description	<p>The generic function stream-read-buffer is called by stream-fill-buffer to place characters into the region of the buffer <i>buffer</i> bounded by <i>start</i> and <i>end</i>. <i>stream</i> should be an instance of a subclass of buffered-stream. stream-read-buffer should block until some data is available. <i>result</i> should be the number of characters actually placed in the buffer (0 if at end of file). This method must be implemented for subclasses of buffered-stream that handle input.</p>	
See also	buffered-stream stream-fill-buffer	

stream-read-byte *Generic Function*

Summary	A generic function used by read-byte to read an integer or :eof symbol from a binary stream.	
Package	stream	
Signature	stream-read-byte <i>stream</i> => <i>result</i>	
Arguments	<i>stream</i>	An input stream.
Values	<i>result</i>	An integer or :eof .
Description	<p>The generic function stream-read-byte is used by read-byte, and returns either an integer read from the binary stream specified by <i>stream</i>, or the keyword :eof.</p> <p>A method must be implemented for all binary subclasses of buffered-stream that handle input. A typical implementation will call stream-read-char and convert the character to an integer using char-code.</p> <p>A method should be defined for a subclass of fundamental-binary-input-stream.</p>	
See also	buffered-stream fundamental-binary-input-stream fundamental-binary-stream stream-read-char	

stream-read-char *Generic Function*

Summary	Read one character from a stream.
Package	stream
Signature	stream-read-char <i>stream</i> => <i>character</i>

Arguments	<i>stream</i>	An input stream.
Values	<i>character</i>	A character or the :eof symbol.
Description		The generic function <code>stream-read-char</code> reads one item from <i>stream</i> . The item read is either a character or the end of file symbol :eof if the stream is at the end of a file. Every subclass of <code>fundamental-character-input-stream</code> must define a method for this function.
See also		<code>fundamental-character-input-stream</code> <code>stream-unread-char</code>

stream-read-char-no-hang

Generic Function

Summary	Returns either a character from the stream, an :eof if the end-of-file is reached, or nil if no input is currently available.	
Package	<code>stream</code>	
Signature	<code>stream-read-char-no-hang stream => result</code>	
Arguments	<i>stream</i>	An input stream.
Values	<i>result</i>	Either a character, an :eof symbol, or nil.
Description	The generic function <code>stream-read-char-no-hang</code> implements <code>read-char-no-hang</code> . It returns either a character read from the stream, or :eof if end-of-file is reached, or nil if no input is available. The default method provided by <code>fundamental-character-input-stream</code> simply calls <code>stream-read-char</code> which is sufficient for file streams, but interactive streams should define their own method.	
See also	<code>fundamental-character-input-stream</code> <code>stream-read-char</code>	

stream-read-line		<i>Generic Function</i>
Summary	Returns a string read from a stream.	
Package	<code>stream</code>	
Signature	<code>stream-read-line stream => result terminated</code>	
Arguments	<code>stream</code>	An input stream.
Values	<code>result</code>	A string or <code>:eof</code> .
	<code>terminated</code>	A boolean.
Description	<p>The generic function <code>stream-read-line</code> reads a line of characters from <code>stream</code> and returns this line as a string. If the string is terminated by an end-of-file instead of a newline then <code>terminated</code> is <code>t</code>.</p> <p>The default method uses repeated calls to <code>stream-read-char</code>, and uses <code>stream-element-type</code> to determine the element-type of its result.</p>	
See also	<code>fundamental-character-input-stream</code> <code>stream-element-type</code> <code>stream-read-char</code>	

stream-read-sequence		<i>Generic Function</i>
Summary	Reads a number of items from a stream into a sequence.	
Package	<code>stream</code>	
Signature	<code>stream-read-sequence stream sequence start end => index</code>	
Arguments	<code>stream</code>	A stream.
	<code>sequence</code>	A sequence.

	<i>start</i>	An integer.
	<i>end</i>	An integer.
Values	<i>index</i>	An integer.
Description	The generic function <code>stream-read-sequence</code> reads from <i>stream</i> into <i>sequence</i> . Elements from the <i>start</i> of <i>sequence</i> are replaced by elements from <i>stream</i> until <i>end</i> in <i>sequence</i> or the end-of-file in <i>stream</i> is reached. The index of the first element in <i>sequence</i> that is not replaced is returned.	
	A default method is provided by <code>fundamental-character-input-stream</code> which makes repeated calls to <code>stream-read-char</code> and uses (<code>setf elt</code>) to insert characters into <i>sequence</i> . A default method is provided by <code>fundamental-binary-input-stream</code> that makes repeated calls to <code>stream-read-byte</code> and also uses (<code>setf elt</code>) to insert bytes into <i>sequence</i> . Note that this may lead to error if the sequence is of inappropriate type.	
See also	fundamental-binary-input-stream fundamental-character-input-stream stream-read-byte stream-read-char	

stream-read-timeout

Generic Function

Summary	Accesses the read-timeout property of a socket stream.	
Package	<code>stream</code>	
Signature	<code>stream-read-timeout stream => timeout</code>	
Arguments	<i>stream</i>	A socket stream.
Values	<i>timeout</i>	A positive number or <code>nil</code> .

Description	The generic function <code>stream-read-timeout</code> reads the current <i>read-timeout</i> of an instance of <code>comm:socket-stream</code> . <code>(setf stream-read-timeout)</code> sets the read-timeout of an instance of <code>comm:socket-stream</code> .
See also	<code>socket-stream</code> <code>open-tcp-stream</code>

stream-start-line-p

Generic Function

Summary	A generic function that returns <code>t</code> if the stream is positioned at the beginning of a line.	
Package	<code>stream</code>	
Signature	<code>stream-start-line-p stream => result</code>	
Arguments	<code>stream</code> A stream.	
Values	<code>result</code> A boolean.	
Description	<p>The generic function <code>stream-start-line-p</code> returns <code>t</code> if <code>stream</code> is positioned at the beginning of a line, and <code>nil</code> otherwise. It is permissible to define a method that always returns <code>nil</code>.</p> <p>Note that although a value of <code>0</code> from <code>stream-line-column</code> also indicates the beginning of a line, there are cases where <code>stream-start-line-p</code> can be meaningfully implemented and <code>stream-line-column</code> cannot. For example, for a window using variable-width characters the column number is not very meaningful, whereas the beginning of a line has a clear meaning.</p> <p>The default method for <code>stream-start-line-p</code> on class <code>fundamental-character-output-stream</code> uses <code>stream-line-column</code>. Therefore, if this is defined to return <code>nil</code>, a method</p>	

should be provided for either `stream-start-line-p` or `stream-fresh-line`.

See also `fundamental-character-output-stream`
`stream-fresh-line`
`stream-line-column`

stream-terpri *Generic Function*

Summary Writes an end of line to a stream.

Package `stream`

Signature `stream-terpri stream => nil`

Arguments `stream` A stream.

Values `nil`

Description The generic function `stream-terpri` writes an end of line to `stream`, as for `terpri`. The default method for `stream-terpri` is (`stream-write-char stream #\Newline`).

See also `stream-write-char`

stream-unread-char *Generic Function*

Summary Undoes the last call to `stream-read-char`.

Package `stream`

Signature `stream-unread-char stream character => nil`

Arguments `stream` A stream.
 `character` A character.

Values	<code>nil</code>
Description	The generic function <code>stream-unread-char</code> undoes the last call to <code>stream-read-char</code> , as in <code>unread-char</code> . Every subclass of <code>fundamental-character-input-stream</code> must define a method for this function.
See also	<code>fundamental-character-input-stream</code>

stream-write-buffer

Generic Function

Summary	Writes a part of stream's buffer.						
Package	<code>stream</code>						
Signature	<code>stream-write-buffer stream buffer start end</code>						
Arguments	<table> <tr> <td><code>stream</code></td> <td>An output stream.</td> </tr> <tr> <td><code>buffer</code></td> <td>A stream buffer.</td> </tr> <tr> <td><code>start, end</code></td> <td>Bounding indexes for a subsequence of <code>buffer</code>.</td> </tr> </table>	<code>stream</code>	An output stream.	<code>buffer</code>	A stream buffer.	<code>start, end</code>	Bounding indexes for a subsequence of <code>buffer</code> .
<code>stream</code>	An output stream.						
<code>buffer</code>	A stream buffer.						
<code>start, end</code>	Bounding indexes for a subsequence of <code>buffer</code> .						
Description	<p>The generic function <code>stream-write-buffer</code> is called by <code>stream-flush-buffer</code> to write the region of the buffer bounded by <code>start</code> and <code>end</code> to the stream's underlying data sink.</p> <p><code>stream</code> should be an instance of a subclass of <code>buffered-stream</code>.</p> <p>This method must be implemented for subclasses of <code>buffered-stream</code> that handle output.</p>						
See also	<code>buffered-stream</code> <code>stream-flush-buffer</code>						

stream-write-byte

Generic Function

Summary	A generic function used by <code>write-byte</code> to write an integer to a binary stream.	
Package	<code>stream</code>	
Signature	<code>stream-write-byte stream integer => result</code>	
Arguments	<p><i>stream</i> A stream.</p> <p><i>integer</i> An integer.</p>	
Values	<i>result</i> An integer.	
Description	<p>The generic function <code>stream-write-byte</code> is used by <code>write-byte</code>, and writes the integer <i>integer</i> to the binary stream specified by <i>stream</i>.</p> <p>A method must be implemented for all binary subclasses of <code>buffered-stream</code> that handle output. A typical implementation will convert the integer to a character using <code>code-char</code> and call <code>stream-write-char</code>.</p> <p>A method should be defined for all subclasses of <code>fundamental-binary-output-stream</code>.</p>	

See also

`buffered-stream`
`fundamental-binary-output-stream`
`fundamental-binary-stream`
`stream-write-char`

stream-write-char

Generic Function

Summary	Writes a character to a specified stream.
Package	<code>stream</code>

Signature	<code>stream-write-char stream character => character</code>	
Arguments	<i>stream</i>	A stream.
	<i>character</i>	A character.
Values	<i>character</i>	A character.
Description	The generic function <code>stream-write-char</code> writes <i>character</i> to <i>stream</i> . Every subclass of <code>fundamental-character-output-stream</code> must have a method defined for this function.	
See also	<code>fundamental-character-output-stream</code>	

stream-write-sequence Generic Function

Summary	Writes a subsequence of a sequence to a stream.	
Package	<code>stream</code>	
Signature	<code>stream-write-sequence stream sequence start end => result</code>	
Arguments	<i>stream</i>	A stream.
	<i>sequence</i>	A sequence.
	<i>start</i>	An integer.
	<i>end</i>	An integer.
Values	<i>result</i>	A sequence.
Description	The generic function <code>stream-write-sequence</code> is used by <code>write-sequence</code> to write a subsequence of <i>sequence</i> delimited by <i>start</i> and <i>end</i> to <i>stream</i> . A default method is provided by <code>fundamental-character-output-stream</code> that tests each element of <i>sequence</i> in turn, and then uses <code>stream-write-char</code> or produces an error. A	

default method is provided by `fundamental-binary-output-stream` that tests each element of `sequence` in turn, and then uses `stream-write-byte` or produces an error.

See also `fundamental-binary-output-stream`
 `fundamental-character-output-stream`
 `stream-read-sequence`
 `stream-write-byte`
 `stream-write-char`

stream-write-string

Generic Function

Summary	Used by <code>write-string</code> to write a string to a character output stream.	
Package	<code>stream</code>	
Signature	<code>stream-write-string stream string &optional start end => result</code>	
Arguments	<code>stream</code>	A stream.
	<code>string</code>	A string.
	<code>start</code>	An integer.
	<code>end</code>	An integer.
Values	<code>result</code>	A string.
Description	The generic function <code>stream-write-string</code> is used by <code>write-string</code> to write <code>string</code> to <code>stream</code> . The string can, optionally, be delimited by <code>start</code> and <code>end</code> . The default method provided by <code>fundamental-character-output-stream</code> uses repeated calls to <code>stream-write-char</code> .	
See also	<code>fundamental-character-output-stream</code> <code>stream-write-char</code>	

with-stream-input-buffer	Macro
Summary	Allows access to the input buffer.
Package	<code>stream</code>
Signature	<code>with-stream-input-buffer (buffer index limit) stream &body body => result</code>
Arguments	<p><i>buffer, index, limit</i> Variables.</p> <p><i>stream</i> An input stream.</p> <p><i>body</i> Code.</p>
Values	<i>result</i> The value returned by <i>body</i> .
Description	<p>The macro <code>with-stream-input-buffer</code> allows access to the state of the input buffer for the given buffered stream. <i>stream</i> should be an instance of a subclass of <code>buffered-stream</code>. Within the code <i>body</i>, the variables <i>buffer, index</i> and <i>limit</i> are bound to the buffer of <i>stream</i>, its current index and the limit of the buffer. Setting <i>buffer, index</i> or <i>limit</i> will change the values in the stream <i>stream</i> but note that other changes to these values (for example, by calling other stream functions) will not affect the values bound within the macro. See the example for a typical use which shows how this restriction can be handled.</p> <p>The buffer is always of type <code>simple-string</code>. The <code>stream-element-type</code> of <i>stream</i> depends on how it was constructed. The index is the position of the next element to be read from the buffer and the limit is the position of the element after the end of the buffer. Therefore there is no data in the buffer when <i>index</i> is greater than or equal to <i>length</i>.</p>

Example This example function returns a string with exactly four characters read from a buffered stream. If `end-of-file` is reached before four characters have been read, it returns `nil`.

```
(defun read-4-chars (stream)
  (declare (type stream:buffered-stream stream))
  (let ((res (make-string 4))
        (elt 0))
    (elt 0))
    ;; Outer loop handles buffer filling.
    (loop
      ;; Inner loop handles buffer scanning.
      (loop (stream:with-stream-input-buffer (buf ind
lim) stream
        (when (>= ind lim)
          ;; End of buffer: try to refill.
          (return))
        (setf (schar res elt) (schar buf ind))
        (incf elt)
        (incf ind)
        (when (= elt 4)
          (return-from read-4-chars res))))
      (unless (stream:stream-fill-buffer stream)
        (return-from read-4-chars nil)))))
```

See also `buffered-stream`
`with-stream-output-buffer`

with-stream-output-buffer *Macro*

Summary Allows access to the output buffer.

Package `stream`

Signature `with-stream-output-buffer (buffer index limit) stream &body body => result`

Arguments *buffer, index, limit*

Variables

stream An output stream

body Code

Values	<i>result</i>	The value returned by body.
Description	<p>The macro <code>with-stream-output-buffer</code> allows access to the state of the output buffer for the given buffered stream.</p> <p><i>stream</i> should be an instance of a subclass of <code>buffered-stream</code>.</p> <p>Within the code <i>body</i>, the variable names <i>buffer</i>, <i>index</i> and <i>limit</i> are bound to the buffer of <i>stream</i>, its current index and the limit of the buffer. Setting <i>buffer</i>, <i>index</i> or <i>limit</i> will change the values in the stream <i>stream</i> but note that other changes to these values (for example, by calling other stream functions) will not affect the values bound within the macro. See the example for a typical use which shows how this restriction can be handled.</p> <p>The buffers are always of type <code>simple-string</code>. The <code>stream-element-type</code> of <i>stream</i> depends on how the stream was constructed.</p> <p>The index is the position of the next free element in the buffer and the limit is the position of the element after the end of the buffer. Therefore the buffer is full when <i>index</i> is greater than or equal to <i>length</i>.</p>	
Example	This example function writes a four character string to a buffered stream.	

```
(defun write-4-chars (stream string)
  (declare (type stream:buffered-stream stream))
  (let ((elt 0))
    ;; Outer loop handles buffer flushing.
    (loop
      ;; Inner loop handles buffer updating.
      (loop (stream:with-stream-output-buffer (buf ind
lim) stream
        (when (>= ind lim)
          ;; Buffer full: try to flush.
          (return))
        (setf (schar buf ind) (schar string elt))
        (incf elt)
        (incf ind)
        (when (= elt 4)
          (return-from write-4-chars))))
      (stream:stream-flush-buffer stream))))
```

See also

[buffered-stream](#)
[with-stream-input-buffer](#)

16

The SYSTEM Package

This chapter describes symbols available in the `SYSTEM` package.

apply-with-allocation-in-gen-num		<i>Function</i>
Summary		Allows control over which generation objects are allocated in, in 64-bit LispWorks.
Package		<code>system</code>
Signature		<code>apply-with-allocation-in-gen-num what gen-num func &rest args => results</code>
Arguments	<i>what</i>	One of <code>:cons</code> , <code>:symbol</code> , <code>:function</code> , <code>:non-pointer</code> and <code>:other</code> .
	<i>gen-num</i>	An integer in the inclusive range [0,7], or <code>nil</code> .
	<i>func</i>	A function designator.
	<i>args</i>	The arguments passed to <i>func</i> .

Values	<i>results</i>	The values returned from the call to <i>func</i> with <i>args</i> .
Description	The function <code>apply-with-allocation-in-gen-num</code> applies the function <i>func</i> to <i>args</i> such that objects of allocation type <i>what</i> are allocated in generation <i>gen-num</i> , in 64-bit Lisp-Works.	
	See also the keyword <code>:allocation</code> to <code>make-array</code> , which catches the most common cases.	
		It is probably quite rare that it is useful to use this function, unless the function allocates a lot, and you are certain that every object that is allocated of the allocation type is long-lived, which is normally difficult to tell.
		Note that allocation of interned symbols is controlled separately by <code>*symbol-alloc-gen-num*</code> .
		Note: In 32-bit LispWorks the argument <i>what</i> is ignored and the effect is like that of the macro <code>allocation-in-gen-num</code> .
See also	<code>allocation-in-gen-num</code> <code>make-array</code> <code>*symbol-alloc-gen-num*</code>	

		Type
Summary	The augmented string type.	
Package	<code>system</code>	
Signature	<code>augmented-string length</code>	
Arguments	<i>length</i>	The length of the string (or *, meaning any).
Description	This is the string type that can hold any character. Equivalent to:	

(**vector** **character length**)

augmented-string-p

Function

Summary	Tests if an object is an augmented string.	
Package	system	
Signature	augmented-string-p	<i>object => bool</i>
Arguments	<i>object</i>	The object to be tested.
Values	<i>bool</i>	t if <i>object</i> is an augmented string; nil otherwise.
Description	This is the predicate for augmented strings.	
See also	augmented-string	

call-system

Function

Package	system	
Signature	call-system	<i>command &key current-directory wait shell-type => status</i>
Arguments	<i>command</i>	A string, a list of strings, a simple-vector of strings, or nil .
	<i>current-directory</i>	A string. Implemented only on Microsoft Windows.
	<i>wait</i>	A boolean.
	<i>shell-type</i>	A string or nil .

Values	<code>status</code>	The exit status of the invoked shell or process.
Description		<p><code>call-system</code> allows executables and DOS or Unix shell commands to be called from Lisp code. The output goes to standard output, as the operating system sees it. (This normally means <code>*terminal-io*</code> in LispWorks.)</p> <p>If <i>command</i> is a string then it is passed to the shell the command to run without any other arguments. The type of shell to run is determined by <i>shell-type</i> as described below.</p> <p>If <i>command</i> is a list, then its first element is the command to run directly and the other elements are passed as arguments on the command line (that is, element 0 has its name in <code>argv[0]</code> in C, and so on). If <i>command</i> is a simple-vector of strings, the element at index 0 is the command to run and the other elements are the complete set of arguments seen by the command (that is, element 1 becomes <code>argv[0]</code> in C, and so on). If <i>command</i> is <code>nil</code>, then the shell is run.</p> <p>On Microsoft Windows <i>current-directory</i> is the <i>lpCurrentDirectory</i> argument passed to <code>createProcess</code>. If this is not supplied, the <code>pathname-location</code> of the <code>current-pathname</code> is passed.</p> <p>If <i>wait</i> is true, <code>call-system</code> does not return until the process has exited. The default for <i>wait</i> is <code>t</code>.</p> <p>On Unix/Linux/Mac OS X/FreeBSD, if <i>shell-type</i> is a string it specifies the shell. If <i>shell-type</i> is <code>nil</code> (the default) then the Bourne shell, <code>/bin/sh</code>, is used. The C shell may be obtained by passing <code>" /bin/csh"</code>.</p> <p>On Microsoft Windows if <i>shell-type</i> is <code>nil</code> then <code>cmd.exe</code> is used on Windows Vista, Windows XP and Windows 2000 and <code>command.com</code> on Windows 98 and Windows ME.</p> <p><code>call-system</code> returns the exit status of the shell invoked to execute the command on Unix/Linux/Mac OS X, or the process created on Microsoft Windows.</p>

Compatibility Note The `:shell-type` argument is not implemented in LispWorks for Windows 4.4 and earlier, and `cmd.exe` is not used implicitly.

LispWorks for Windows 5.0 and later use *shell-type cmd.exe* (or `command.com`) by default when *command* is a string. The user may see a DOS command window in this case. To call your command directly *command* should be a list, as in the last example below.

Example On Unix:

```
(call-system (format nil "adb ~a < ~a > ~a"
                      (namestring a)
                      (namestring b)
                      (namestring c)))
```

On Microsoft Windows:

```
(sys:call-system "sleep 3" :wait t)

(sys:call-system '("notepad" "myfile.txt"))
```

See also [open-pipe](#)
[call-system-showing-output](#)
[run-shell-command](#)

call-system-showing-output

Function

Package `system`

Signature `call-system-showing-output command &key current-directory
prefix show-cmd output-stream wait shell-type kill-process-on-abort
=> status`

Arguments *command* A string, a list of strings, a simple-vector of strings, or `nil`.

current-directory A string. Supported only on Microsoft Windows.

	<i>prefix</i>	A string.
	<i>show-cmd</i>	A boolean.
	<i>output-stream</i>	A symbol.
	<i>wait</i>	A boolean.
	<i>shell-type</i>	A string. Supported only on Unix/Linux/Mac OS X.
	<i>kill-process-on-abort</i>	A generalized boolean.
Values	<i>status</i>	The exit status of the invoked shell or process.
Description	<p><code>call-system-showing-output</code> is an extension to <code>call-system</code> which allows output to be redirected. On Unix/Linux/Mac OS X this means it can be redirected to places other than the shell process from which the LispWorks image was invoked. <code>call-system-showing-output</code> therefore allows the user to, for example, invoke a shell command and redirect the output to the current Listener window.</p> <p>The argument <i>command</i> is interpreted as by <code>call-system</code>. <i>prefix</i> is a prefix to be printed at the start of any output line. The default value is " ; ".</p> <p><i>show-cmd</i> specifies whether or not the <i>cmd</i> invoked will be printed as well as the output for that command. If <code>t</code> then <i>cmd</i> will be printed. The default value for <i>show-cmd</i> is <code>t</code>.</p> <p><i>output-stream</i> specifies where the output will be sent to. The default value is <code>*standard-output*</code>.</p> <p>If <i>wait</i> is true, <code>call-system-showing-output</code> does not return until the process has exited. If <code>nil</code>, <code>call-system-showing-output</code> returns immediately and no output is shown. The default for <i>wait</i> is <code>t</code>.</p>	

shell-type is a string naming a UNIX shell. The default is `"/bin/sh"`.

If *kill-process-on-abort* is true, then when `call-system-showing-output` is aborted the process is killed. The default value of *kill-process-on-abort* is `nil`.

`call-system-showing-output` returns the exit status of the shell invoked to execute the command on Unix/Linux/Mac OS X/FreeBSD, or the process created on Microsoft Windows.

Examples

On Linux:

```
CL-USER 1 > (sys:call-system-showing-output "pwd"
  :prefix "***")
***pwd
***/amd/xanfs1-cam/u/ldisk/sp/lispsrc/v42/builds
0
```

```
CL-USER 2 > (sys:call-system-showing-output "pwd"
  :prefix "&&&" :show-cmd nil)
&&&/amd/xanfs1-cam/u/ldisk/sp/lispsrc/v42/builds
0
```

On Microsoft Windows:

```
CL-USER 223 > (sys:call-system-showing-output
  "cmd /c type hello.txt"
  :prefix "***")
***cmd /c type hello.txt
***Hi there
0
```

```
CL-USER 224 > (sys:call-system-showing-output
  "cmd /c type hello.txt"
  :prefix "&&&"
  :show-cmd nil)
&&&Hi there
0
```

See also

`call-system`
`open-pipe`
`run-shell-command`

cdr-assoc	<i>Function</i>												
Summary	A generalized reference for alist elements.												
Package	<code>system</code>												
Signature	<code>cdr-assoc item alist &key test test-not key => result</code> <code>(setf cdr-assoc) value item alist => value</code>												
Arguments	<table> <tr> <td><i>item</i></td> <td>An object.</td> </tr> <tr> <td><i>alist</i></td> <td>An association list.</td> </tr> <tr> <td><i>test</i></td> <td>A function designator.</td> </tr> <tr> <td><i>test-not</i></td> <td>A function designator.</td> </tr> <tr> <td><i>key</i></td> <td>A function designator.</td> </tr> <tr> <td><i>value</i></td> <td>An object.</td> </tr> </table>	<i>item</i>	An object.	<i>alist</i>	An association list.	<i>test</i>	A function designator.	<i>test-not</i>	A function designator.	<i>key</i>	A function designator.	<i>value</i>	An object.
<i>item</i>	An object.												
<i>alist</i>	An association list.												
<i>test</i>	A function designator.												
<i>test-not</i>	A function designator.												
<i>key</i>	A function designator.												
<i>value</i>	An object.												
Values	<i>result</i> An object (from <i>alist</i>) or <code>nil</code> .												
Description	<p>The functions <code>cdr-assoc</code> and <code>(setf cdr-assoc)</code> provide a generalized reference for elements in an association list. The arguments are all as specified for the Common Lisp function <code>assoc</code>. <code>cdr-assoc</code> and <code>(setf cdr-assoc)</code> read and write the <code>cdr</code> of an element in a manner consistent with the Common Lisp notion of places.</p> <p><code>cdr-assoc</code> returns the <code>cdr</code> of the first cons in the alist <i>alist</i> that satisfies the test, or <code>nil</code> if no element of <i>alist</i> matches.</p> <p><code>(setf cdr-assoc)</code> modifies the first cons in <i>alist</i> that satisfies the test, setting its <code>cdr</code> to <i>value</i>. If no element of <i>alist</i> matches, then <code>(setf cdr-assoc)</code> constructs a new cons (<code>cons item value</code>) and inserts it in the head of <i>alist</i>.</p>												

Example	<pre> CL-USER 1 > (defvar *my-alist* (list (cons :foo 1) (cons :bar 2))) *MY-ALIST* CL-USER 2 > (setf (sys:cdr-assoc :bar *my-alist*) 3) 3 CL-USER 3 > *my-alist* ((:FOO . 1) (:BAR . 3)) </pre>
---------	---

check-network-server *Variable*

Summary Indicates the presence of a network license.

Note: LispWorks for UNIX only.

Package `system`

Description This should always be set to `t` for a site (that is, network) license — the licensing mechanism does not work in any other circumstances. Do not set the variable otherwise, as it overrides any useful diagnostics which may accompany keyfile errors. Not applicable to LispWorks for Linux, Windows, FreeBSD or Macintosh.

coerce-to-gesture-spec *Function*

Summary Returns a Gesture Spec object.

Package `system`

Signature `coerce-to-gesture-spec object &optional errorp => gspec`

Arguments `object` A character, keyword, Gesture Spec or string.

	<i>errorp</i>	A boolean.
Values	<i>gspec</i>	A Gesture Spec object
Description	<p>The function <code>coerce-to-gesture-spec</code> returns a Gesture Spec object <i>gspec</i> which can be used to represent the key-stroke indicated by <i>object</i>.</p> <p>If <i>object</i> is a Lisp character, then <i>gspec</i>'s data is one of the known Gesture Spec keywords, or its <code>char-code</code>, and <i>gspec</i>'s modifiers contains its <code>char-bits</code> attribute mapped onto the values <code>gesture-spec-control-bit</code> etc.</p> <p>If <i>object</i> is a keyword, then it must be one of the known Gesture Spec keywords and becomes <i>gspec</i>'s data. <i>gspec</i>'s modifiers is 0.</p> <p>If <i>object</i> is a string, then <code>coerce-to-gesture-spec</code> expects it to be a sequence of modifier key names separated by the - character, followed by a single character or a character name as returned by <code>name-char</code> or the name of one of the known Gesture Spec keywords. Then <i>gspec</i> contains the corresponding Gesture Spec keyword or <code>char-code</code> in its <i>data</i>, and the modifier keys are represented in its <i>modifiers</i>.</p> <p>If <i>object</i> is a Gesture Spec object, it is simply returned.</p> <p><code>coerce-to-gesture-spec</code> does not create wild gesture specs.</p>	

Examples

```
(sys:coerce-to-gesture-spec #\Control-C)
=>
#S(SYSTEM:::GESTURE-SPEC :DATA 67 :MODIFIERS 2)

CL-USER 8 > (sys:coerce-to-gesture-spec #\Control-\c)
=>
#S(SYSTEM:::GESTURE-SPEC :DATA 99 :MODIFIERS 2)

(sys:coerce-to-gesture-spec :F10)
=>
#S(SYSTEM:::GESTURE-SPEC :DATA :F10 :MODIFIERS 0)

(sys:coerce-to-gesture-spec "Ctrl-C")
=>
#S(SYSTEM:::GESTURE-SPEC :DATA 67 :MODIFIERS 2)

(sys:coerce-to-gesture-spec "Shift-F10")
=>
#S(SYSTEM:::GESTURE-SPEC :DATA :F10 :MODIFIERS 1)
```

See also

- `gesture-spec-control-bit`
- `gesture-spec-data`
- `gesture-spec-modifiers`
- `gesture-spec-p`
- `gesture-spec-to-character`
- `make-gesture-spec`
- `print-pretty-gesture-spec`

copy-preferences-from-older-version		<i>Function</i>
Summary	Copies uses preferences.	
Package	<code>system</code>	
Signature		<code>copy-preferences-from-older-version <i>old-path</i> <i>new-path</i></code> <code>&optional <i>flag-name</i></code>
Arguments	<i>old-path</i>	A preference path.
	<i>new-path</i>	A preference path.

<i>flag-name</i>	A string.
Description	<p>The function <code>copy-preferences-from-older-version</code> copies uses preferences from one part of the registry to another. <i>old-path</i> and <i>new-path</i> are the paths of preferences for the old and the new version, corresponding to the paths that were passed to (<code>setf product-registry-path</code>).</p> <p><i>flag-name</i> is a name of the flag to use to record in the registry that the copy is already done. <i>flag-name</i> must be a valid registry value name on Microsoft Windows, and a valid filename on all other platforms. The default value of <i>flag-name</i> is the string "<code>copied-old-preferences</code>".</p> <p><code>copy-preferences-from-older-version</code> performs several checks:</p> <ol style="list-style-type: none"> 1. It checks if it already copied to <i>new-path</i> in the current session, and if so does nothing. 2. It checks if the <i>flag-name</i> entry exists, and if so it does nothing. 3. It checks if another call to <code>copy-preferences-from-older-version</code> is already executing (in another thread), and if so it just waits for the other call to finish. <p>Then if all the checks above indicate that copying is still needed, <code>copy-preferences-from-older-version</code> copies the values from the tree below <i>old-path</i> to a tree below <i>new-path</i>. It traverses the entire tree below <i>old-path</i>, and checks each key to see if it has any values.</p> <p>For a key that has values, it checks if the key exists under <i>new-path</i>, and if the key exists it does not copy any of the values for this key, though it still traverses and maybe copies its subkeys. If the key does not exist under <i>new-path</i>, it creates the key and copies the values.</p>

Because it makes checks before doing any work, `copy-preferences-from-older-version` is an inexpensive call that can be used freely.

See also `product-registry-path`
 `user-preference`

		<i>Function</i>
count-gen-num-allocation		
Summary	Returns the amount of allocated data in a generation in 64-bit LispWorks.	
Package	<code>system</code>	
Signature	<code>count-gen-num-allocation gen-num &optional include-lower-generations</code>	
Arguments	<code>gen-num</code> An integer between 0 and 7, inclusive. <code>include-lower-generations</code> A generalized boolean.	
Values	<code>allocation</code> An integer.	
Description	The function <code>count-gen-num-allocation</code> returns the amount of allocated data in generation <code>gen-num</code> . If <code>include-lower-generations</code> is non- <code>nil</code> , the returned value <code>allocation</code> also includes the data in the younger generations. Note: this function is implemented only in 64-bit LispWorks. It is not relevant to the Memory Management API in 32-bit implementations, where you can use <code>room-values</code> instead.	
See also	<code>room-values</code>	

default-eol-style		<i>Function</i>
Summary	Provides a default end of line style for a file.	
Package	<code>system</code>	
Signature	<code>default-eol-style pathname ef-spec buffer length => new-ef-spec</code>	
Arguments	<p><code>pathname</code> Pathname identifying location of <i>buffer</i>.</p> <p><code>ef-spec</code> An external format spec.</p> <p><code>buffer</code> A buffer whose contents are examined.</p> <p><code>length</code> Length (an integer) up to which <i>buffer</i> should be examined.</p>	
Values	<code>new-ef-spec</code> A new external format spec created by merging <i>ef-spec</i> with the encoding that was found.	
Description	Merge <i>ef-spec</i> with (<code>:default :eol-style :crlf</code>) on Microsoft Windows, (<code>:default :eol-style :lf</code>) on UNIX/Linux/Mac OS X. This is usually used as the last function on its list.	
See also	<code>*file-eol-style-detection-algorithm*</code>	

default-stack-group-list-length		<i>Variable</i>
Summary	The size of the stack cache.	
Package	<code>system</code>	
Initial Value	<code>10</code>	
Description	This variable determines the maximum size of the stack cache.	

Process stacks are cached and reused. When a process dies, its stack is put in the stack cache for future reuse if there are currently less than `*default-stack-group-list-length*` stacks in the cache. Therefore if your application repeatedly creates and discards more than 10 processes you should consider increasing the value of this variable.

Note that stacks are allocated in generation 2, hence a program with a high turnover of processes may need to call `(mark-and-sweep 2)` periodically unless all the stacks of dead processes are reused.

The default stack size is 64KB on all 32-bit LispWorks x86 platforms.

See also [mark-and-sweep](#)

		<i>Macro</i>
Summary	Defines a top level loop command.	
Package	<code>system</code>	
Signature	<code>define-top-loop-command name-and-options lambda-list form*</code> <code>name-and-options ::= name</code> <code> (name option*)</code> <code>option ::= (:aliases alias*)</code> <code> (:result-type result-type)</code>	
Arguments	<code>name</code> <code>alias</code> <code>lambda-list</code> <code>result-type</code>	A keyword naming the command. A keyword naming an alias for the command. A destructuring lambda list. One of the symbols <code>values</code> , <code>eval</code> and <code>nil</code> .

Description	<p>The macro <code>define-top-loop-command</code> defines a top level loop command called <i>name</i> which takes the parameters specified by <i>lambda-list</i>. If <code>&whole</code> is used in <i>lambda-list</i> then the variable will be bound to a list containing the whole command line, including the command name, but the command name is not included in <i>lambda-list</i> otherwise.</p> <p>If any aliases are specified in <i>option</i>, these keywords will also invoke the command.</p> <p>When the command is used, each form is evaluated in sequence with the variables from <i>lambda-list</i> bound to the subsequent forms on the command line.</p> <p>If <i>result-type</i> is <code>values</code> (the default), then the values of the last form will be returned to the top level loop.</p> <p>If <i>result-type</i> is <code>eval</code>, then the value of the last form should be a form and is evaluated by the top level loop as if it had been entered at the prompt.</p> <p>If <i>result-type</i> is <code>nil</code>, then the last form should return two values. If the second value is <code>nil</code> then the first value is treated as a list of values to returned to the top level loop. If the second value is non-<code>nil</code> then the first value should be a form and is evaluated by the top level loop as if it had been entered at the prompt.</p> <p>Note: for details of pre-defined top level loop commands, enter <code>:?</code> at the Listener prompt.</p>
Example	<p>Given this definition:</p> <pre>(define-top-loop-command (:lave (:result-type eval)) (form) (reverse form))</pre> <p>then the command line</p> <pre>:lave (1 2 list)</pre> <p>will evaluate the form <code>(list 2 1)</code>.</p>

Here are definitions for two commands both of which will run `apropos`:

```
(define-top-loop-command (:apropos-eval
                           (:result-type eval))
                           (&rest args)
                           ` (apropos ,@args))

(define-top-loop-command :apropos-noeval (&rest args)
                        (apply 'apropos args))
```

The first one will evaluate the arguments before calling `apropos` whereas the second one will just pass the forms, so

```
:apropos-noeval foo
```

will find all the symbols containing the string `foo`, whereas

```
(setq foo "bar")
```

```
:apropos-eval foo
```

will find all the symbols containing the string `bar`.

detect-eol-style *Function*

Summary Detects the end of line style of a file.

Package `system`

Signature `detect-eol-style pathname ef-spec buffer length => new-ef-spec`

Arguments `pathname` Pathname identifying location of `buffer`.

`ef-spec` An external format spec.

`buffer` A buffer whose contents are examined.

`length` Length (an integer) up to which `buffer` should be examined.

Values	<i>new-ef-spec</i>	A new external format spec created by merging <i>ef-spec</i> with the encoding that was found.
Description		<p>When the encoding in <i>ef-spec</i> has foreign type (<code>unsigned-byte 8</code>), search <i>buffer</i> up to <i>length</i> for the first occurrence of the byte <code>(10)</code>. If found, and it is preceded in <i>buffer</i> by <code>(13)</code>, merge <i>ef-spec</i> with</p> <pre>(:default :eol-style :crlf)</pre> <p>If found and is not preceded by <code>(13)</code>, merge <i>ef-spec</i> with</p> <pre>(:default :eol-style :lf)</pre> <p>Thus a complete external format spec is constructed. Otherwise, return <i>ef-spec</i>.</p> <p>When the encoding in <i>ef-spec</i> has foreign type (<code>unsigned-byte 16</code>), search <i>buffer</i> up to <i>length</i> for the first occurrence of the byte sequence <code>(13 0 10)</code>. If found, merge <i>ef-spec</i> with</p> <pre>(:default :eol-style :crlf)</pre> <p>If <code>(13 0 10)</code> is not found, search <i>buffer</i> up to <i>length</i> for <code>(10 0)</code> or <code>(0 10)</code>. If found, merge <i>ef-spec</i> with</p> <pre>(:default :eol-style :lf)</pre> <p>Thus a complete external format spec is constructed. Otherwise, return <i>ef-spec</i>.</p>
See also	<code>*file-eol-style-detection-algorithm*</code>	

detect-japanese-encoding-in-file		<i>Function</i>
Summary	Determines which type of Japanese encoding is used in a buffer.	
Package	<code>system</code>	

Signature	<code>detect-japanese-encoding-in-file pathname ef-spec buffer length => new-ef-spec</code>	
Arguments	<i>pathname</i>	Pathname identifying location of <i>buffer</i> .
	<i>ef-spec</i>	An external format spec.
	<i>buffer</i>	A buffer whose contents are examined.
	<i>length</i>	Length (an integer) up to which <i>buffer</i> should be examined.
Values	<i>new-ef-spec</i>	A new external format spec created by merging <i>ef-spec</i> with the Japanese encoding that was found.
Description	Assume the encoding is one of <code>:jis</code> , <code>:sjis</code> , <code>:euc</code> , <code>:unicode</code> and <code>:ascii</code> , and try to determine which of these it is, by looking for distinctive byte sequences in <i>buffer</i> up to <i>length</i> . If found, merge <i>ef-spec</i> with that encoding.	
See also	<code>*file-encoding-detection-algorithm*</code>	

detect-unicode-bom

Function

Summary	Looks for the Unicode Byte Order Mark, which if found is assumed to indicate a Unicode UCS-2 encoded file.	
Package	<code>system</code>	
Signature	<code>detect-unicode-bom pathname ef-spec buffer length => new-ef-spec</code>	
Arguments	<i>pathname</i>	Pathname identifying location of <i>buffer</i> .
	<i>ef-spec</i>	An external format spec.
	<i>buffer</i>	A buffer whose contents are examined.
	<i>length</i>	Length (an integer) up to which <i>buffer</i> should be examined.

Values	<code>new-ef-spec</code>	A new external format spec created by merging <code>ef-spec</code> with the encoding that was found.
Description		<p>When called as part of <code>open</code>'s encoding detection routine, if byte pair FE FF is found at the start of the file, it is assumed to be UTF16-BE encoded. This encoding is represented by the ef-spec (<code>:unicode :little-endian nil</code>).</p> <p>If byte pair FF FE is found at the start of the file, it is assumed to be UTF16-LE encoded. This encoding is represented by the ef-spec (<code>:unicode :little-endian t</code>).</p>
See also	<code>*file-encoding-detection-algorithm*</code>	

directory-link-transparency *Variable*

Summary	Controls whether <code>directory</code> returns truenames on Unix-like systems.
Package	<code>system</code>
Initial Value	<code>t</code> on Unix-like systems, <code>nil</code> on Microsoft Windows.
Description	<p>In line with the ANSI Common Lisp standard, <code>directory</code> returns truenames by default.</p> <p>Setting <code>*directory-link-transparency*</code> to <code>nil</code> allows you to get the old behavior of <code>directory</code>, whereby soft links are not resolved in the pathnames returned.</p> <p><code>*directory-link-transparency*</code> is the default value of the <code>link-transparency</code> argument to <code>directory</code>.</p>
See also	<code>directory</code>

extended-spaces

Variable

Summary	Extends the notion of space to include more than just the space character.
Package	<code>system</code>
Initial value	<code>nil</code>
Description	When this variable is true, the concept of “space” is extended from just <code>#\Space</code> to include other appropriate characters. The default is <code>nil</code> , for ANS compliance, but we recommend that you set it to <code>t</code> . This variable controls how the format directives <code>~:c</code> and <code>~:@c</code> output graphic characters which have an empty glyph. When this variable is <code>t</code> , all such characters are output using the name: <pre>(format nil "~:C" #\No-break-space) -> "No-Break- Space" (format nil "~:C" (code-char #x3000)) -> "Ideographic- Space"</pre>
	When false, only one such character is output using the name: <pre>(format nil "~:C" #\Space) -> "Space" (format nil "~:C" #\No-break-space) -> " " (format nil "~:C" (code-char #x3000)) -> " "</pre>

It also affects `whitespace-char-p`.

See also	<code>extended-character-p</code>
----------	-----------------------------------

file-encoding-detection-algorithm

Variable

Summary	List of functions to call to work out an encoding.
Package	<code>system</code>

Initial value	<code>(find-filename-pattern-encoding-match find-encoding-option detect-unicode-bom locale-file-encoding)</code>
Description	Functions on this list take four arguments—the pathname of the file; an external format spec; a vector of element-type (<code>unsigned-byte 8</code>) which contains the first bytes of the file; and a non-negative integer which is the maximum extent of buffer to be searched. This length argument is 0 in the case that the file does not exist, or the direction is <code>:output</code> . They return an external format spec, which normally is either <code>ef-spec</code> unmodified, or the result of merging <code>ef-spec</code> with another external format spec via <code>merge-ef-specs</code> .
Example	If you want to inspect the attribute line and then fall back to a default if not found set the variable to the following:
	<code>(find-encoding-option locale-file-encoding)</code>
See also	<code>find-filename-pattern-encoding-match</code> <code>find-encoding-option</code> <code>detect-unicode-bom</code> <code>detect-japanese-encoding-in-file</code> <code>guess-external-format</code> <code>locale-file-encoding</code>

file-encoding-resolution-error *Condition*

Summary	An error type to signal when an external file format cannot be deduced.	
Package	<code>system</code>	
Superclasses	<code>error</code>	
Initargs	<code>:ef-spec</code>	An external format specification.

Description	An error type signalled when <code>open</code> , <code>load</code> or <code>compile-file</code> fail to detect an external format to use.
	The <code>ef-spec</code> slot contains the incomplete external format specification argument constructed by <code>guess-external-format</code> .
See also	<code>guess-external-format</code>

file-eol-style-detection-algorithm *Variable*

Summary	List of functions for determining the end of line style of a file.
Package	<code>system</code>
Description	Functions on this list satisfy the same specifications as for those in *file-encoding-detection-algorithm*. However they will only be passed an external format spec with the name already determined.
Initial value	<code>(detect-eol-style default-eol-style)</code>
See also	<code>detect-eol-style</code> <code>default-eol-style</code> <code>guess-external-format</code>

filename-pattern-encoding-matches *Variable*

Summary	An association of filename patterns to external format specs.
Package	<code>system</code>
Initial value	<code>((\"TAGS\" . (:latin-1 :eol-style :lf)))</code>
Description	An alist of filename patterns to external format specs.

See also *file-encoding-detection-algorithm*

find-encoding-option		Function
Summary	Examines a buffer for an encoding option.	
Package	<code>system</code>	
Signature	<code>find-encoding-option pathname ef-spec buffer length => result</code>	
Arguments	<i>pathname</i>	Pathname identifying location of <i>buffer</i> .
	<i>ef-spec</i>	An external format spec.
	<i>buffer</i>	A buffer whose contents are examined.
	<i>length</i>	Length (an integer) up to which <i>buffer</i> should be examined.
Values	<i>result</i>	The result of reading the value returned from the <code>encoding</code> or <code>external-format</code> option as a Lisp expression in the <code>keyword</code> package.
Description	<p>Looks in the file options (EMACS-style <code>--</code> line) for an option called <code>encoding</code> or <code>external-format</code>, with value <i>value</i>. If found, it reads <i>value</i> as a Lisp expression in the <code>keyword</code> package and merges <i>ef-spec</i> with <i>value</i> and returns the result as <i>result</i>. Thus it does not override a supplied <i>ef-spec</i>.</p>	
See also	<code>*file-encoding-detection-algorithm*</code>	

find-filename-pattern-encoding-match	<i>Function</i>
Summary Finds the encoding of a file based on the filename.	

Package	<code>system</code>	
Signature	<code>find-filename-pattern-encoding-match pathname ef-spec buffer</code> <code>length => new-ef-spec</code>	
Arguments	<i>pathname</i>	Pathname identifying location of <i>buffer</i> .
	<i>ef-spec</i>	An external format spec.
	<i>buffer</i>	A buffer whose contents are examined.
	<i>length</i>	Length (an integer) up to which <i>buffer</i> should be examined.
Values	<i>new-ef-spec</i>	An external format spec.
Description	<p>Compares <i>pathname</i> (using <code>pathname-match-p</code>) with elements of <code>*filename-pattern-encoding-matches*</code>. If a match is found, merges <i>ef-spec</i> with the corresponding external format spec and returns the result as <i>new-ef-spec</i>. Thus it does not override a supplied <i>ef-spec</i>.</p>	
See also	<code>*file-encoding-detection-algorithm*</code> <code>*filename-pattern-encoding-matches*</code>	

gen-num-segments-fragmentation-state		<i>Function</i>
Summary	Shows the fragmentation state in a generation in 64-bit Lisp-Works.	
Package	<code>system</code>	
Signature	<code>gen-num-segments-fragmentation-state gen-num &optional</code> <code>statics-too => fragmentation-state</code>	
Arguments	<i>gen-num</i>	A number.
	<i>statics-too</i>	A generalized boolean?

Values	<i>fragmentation-state</i>
	A list in which each element is a list of length 3.
Description	<p>The function <code>gen-num-segments-fragmentation-state</code> shows the fragmentation state in a generation in 64-bit LispWorks.</p> <p><code>gen-num-segments-fragmentation-state</code> returns a list, where each element is a sub-list showing the fragmentation state in a segment. The sub-list is of the form</p> $(\mathbf{allocation-type} \; \mathbf{allocated} \; \mathbf{free})$ <p>where <code>allocation-type</code> is the allocation type of the segment, <code>allocated</code> is the amount of allocated data in the segment, and <code>free</code> is the total size of free areas in the segment that cannot be easily used.</p> <p>The ratio <code>free/allocated</code> is the ratio that is compared to the fragmentation threshold to decide whether to copy a segment when doing a marking GC with copying (see <code>set-blocking-gen-num</code> and <code>marking-gc</code>).</p> <p>Allocation types <code>:cons-static</code>, <code>:non-pointer-static</code>, <code>:mixed-static</code>, <code>:other-big</code> and <code>:non-pointer-big</code> are included in the result only if <code>statics-too</code> is non-<code>nil</code>. The default value of <code>statics-too</code> is <code>nil</code>.</p> <p>Note: The implementation of <code>set-blocking-gen-num</code> is intended to solve any fragmentation issues automatically.</p> <p>Note: <code>gen-num-segments-fragmentation-state</code> is implemented only in 64-bit LispWorks. It is not relevant to the Memory Management API in 32-bit implementations, where <code>check-fragmentation</code> is available instead.</p>
See also	<code>check-fragmentation</code> <code>marking-gc</code> <code>set-blocking-gen-num</code>

gesture-spec-accelerator-bit *Constant*

Summary	Used in the representation of a keystroke with the accelerator key.
Package	<code>system</code>
Description	The constant <code>gesture-spec-accelerator-bit</code> is used to represent the accelerator key in a Gesture Spec object. See the entry for <code>capi:output-pane</code> in the <i>LispWorks CAPI Reference Manual</i> for more information about the use of Gesture Specs.
See also	<code>coerce-to-gesture-spec</code> <code>gesture-spec-modifiers</code> <code>make-gesture-spec</code>

gesture-spec-control-bit *Constant*

Summary	Used in the representation of a keystroke with the <code>control</code> key.
Package	<code>system</code>
Description	The constant <code>gesture-spec-control-bit</code> is used to represent the <code>Control</code> modifier key in a Gesture Spec object. See the entry for <code>capi:output-pane</code> in the <i>LispWorks CAPI Reference Manual</i> for more information about the use of Gesture Specs.
See also	<code>coerce-to-gesture-spec</code> <code>gesture-spec-modifiers</code> <code>make-gesture-spec</code>

gesture-spec-data	<i>Function</i>
Summary	Returns the key in a Gesture Spec object.
Package	<code>system</code>
Signature	<code>gesture-spec-data gspec => data</code>
Arguments	<code>gspec</code> A Gesture Spec object
Values	<code>data</code> A non-negative integer or a keyword.
Description	<p>The function <code>gesture-spec-data</code> returns an integer or keyword representing the key in the Gesture Spec object <code>gspec</code>.</p> <p>When <code>data</code> is an integer, it is a non-negative integer less than <code>char-code-limit</code>, and <code>gspec</code> represents a keystroke with the key indicated by the character which is the value of (<code>code-char data</code>).</p> <p><code>data</code> can also be a keyword such as <code>:f6</code>, when <code>gspec</code> represents a keystroke with <code>f6</code> pressed.</p>
See also	<code>gesture-spec-modifiers</code> <code>make-gesture-spec</code>

gesture-spec-hyper-bit	<i>Constant</i>
Summary	Used in the representation of a keystroke with the <code>Hyper</code> key.
Package	<code>system</code>
Description	<p>The constant <code>gesture-spec-hyper-bit</code> is used to represent the <code>Hyper</code> modifier key in a Gesture Spec object.</p> <p>See the entry for <code>capi:output-pane</code> in the <i>LispWorks CAPI Reference Manual</i> for more information about the use of Gesture Specs.</p>

See also `coerce-to-gesture-spec`
`gesture-spec-modifiers`
`make-gesture-spec`

gesture-spec-meta-bit *Constant*

Summary Used in the representation of a keystroke with the `meta` key.

Package `system`

Description The constant `gesture-spec-meta-bit` is used to represent the `meta` modifier key in a Gesture Spec object.

See the entry for `capi:output-pane` in the *LispWorks CAPI Reference Manual* for more information about the use of Gesture Specs.

See also `coerce-to-gesture-spec`
`gesture-spec-modifiers`
`make-gesture-spec`

gesture-spec-modifiers *Function*

Summary Returns the modifiers in a Gesture Spec object.

Package `system`

Signature `gesture-spec-modifiers gspec => mods`

Arguments `gspec` A Gesture Spec object

Values `mods` An integer.

Description The function `gesture-spec-modifiers` returns an integer representing the modifiers in the Gesture Spec object `gspec`.

The value *mods* contains some (or none) of the constants `gesture-spec-accelerator-bit`, `gesture-spec-control-bit`, `gesture-spec-meta-bit`, `gesture-spec-hyper-bit`, `gesture-spec-shift-bit` and `gesture-spec-super-bit`, combined as if by `logior`.

See also	<code>gesture-spec-accelerator-bit</code> <code>gesture-spec-control-bit</code> <code>gesture-spec-data</code> <code>gesture-spec-meta-bit</code> <code>gesture-spec-hyper-bit</code> <code>gesture-spec-shift-bit</code> <code>gesture-spec-super-bit</code> <code>make-gesture-spec</code>
----------	---

gesture-spec-p *Function*

Summary	The predicate for Gesture Spec objects.	
Package	<code>system</code>	
Signature	<code>gesture-spec-p object => result</code>	
Arguments	<i>object</i>	A Lisp object
Values	<i>result</i>	A boolean.
Description	The function <code>gesture-spec-p</code> is the predicate for whether the object <i>object</i> is a Gesture Spec object.	
See also	<code>coerce-to-gesture-spec</code> <code>make-gesture-spec</code>	

gesture-spec-shift-bit *Constant*

Summary	Used in the representation of a keystroke with the <code>shift</code> key.
Package	<code>system</code>
Description	The constant <code>gesture-spec-shift-bit</code> is used to represent the <code>shift</code> modifier key in a Gesture Spec object. Note that you may not construct a Gesture Spec with a <code>both-case-p</code> character represented in the <i>data</i> and with <i>modifiers</i> equal to <code>gesture-spec-shift-bit</code> . See <code>make-gesture-spec</code> for details and examples. See the entry for <code>capi:output-pane</code> in the <i>LispWorks CAPI Reference Manual</i> for more information about the use of Gesture Specs.
See also	<code>coerce-to-gesture-spec</code> <code>gesture-spec-modifiers</code> <code>make-gesture-spec</code>

gesture-spec-super-bit *Constant*

Summary	Used in the representation of a keystroke with the <code>super</code> key.
Package	<code>system</code>
Description	The constant <code>gesture-spec-super-bit</code> is used to represent the <code>super</code> modifier key in a Gesture Spec object. See the entry for <code>capi:output-pane</code> in the <i>LispWorks CAPI Reference Manual</i> for more information about the use of Gesture Specs.
See also	<code>coerce-to-gesture-spec</code> <code>gesture-spec-modifiers</code> <code>make-gesture-spec</code>

gesture-spec-to-character		<i>Function</i>
Summary	Returns the character corresponding to a Gesture Spec object.	
Package	<code>system</code>	
Signature	<code>gesture-spec-to-character gspec => char</code>	
Arguments	<code>gspec</code>	A Gesture Spec object
Values	<code>char</code>	A Lisp character.
Description	The function <code>gesture-spec-to-character</code> returns the Lisp <code>character</code> object corresponding to the Gesture Spec object <code>gspec</code> . Modifier bits in <code>gspec</code> are mapped to Lisp character bits attributes where possible. <code>gesture-spec-accelerator-bit</code> is ignored.	
See also	<code>coerce-to-gesture-spec</code> <code>make-gesture-spec</code>	

get-file-stat		<i>Function</i>
Summary	Provides read access to the C stat structure which describes files.	
	Note: not applicable on Microsoft Windows.	
Package	<code>system</code>	
Signature	<code>get-file-stat filename-or-fd => file-stat (errno)</code>	
Arguments	<code>filename-or-fd</code>	A string denoting a file, or a file descriptor.

Values	<code>file-stat</code>	On success, an object representing the stat values. On failure, <code>nil</code> is returned together with a second value.
	<code>errno</code>	Indicates the errno value returned by the system call. This second value is returned only in the case of failure.
Description	<p><code>file-stat</code> is an object representing the stat values, as would be returned by the system call <code>stat</code> (for a filename) or the system call <code>fstat</code> (for an fd).</p> <p>The values in <code>file-stat</code> are the raw data, and it is the responsibility of the user to interpret them when needed. See the UNIX manual entry for <code>stat</code> for details.</p> <p>The values can be read from <code>file-stat</code> by these readers:</p>	
	<p><code>sys:file-stat-inode</code> The inode of the file.</p> <p><code>sys:file-stat-device</code> The id of the device where the file is.</p> <p><code>sys:file-stat-owner-id</code> The user id of the owner of the file.</p> <p><code>sys:file-stat-group-id</code> The group id of the file's group.</p> <p><code>sys:file-stat-size</code> The size of the file in bytes.</p> <p><code>sys:file-stat-blocks</code> The number of 512-bytes blocks used by the file.</p> <p><code>sys:file-stat-mode</code> The protection value of the file.</p> <p><code>sys:file-stat-last-access</code></p> <p><code>sys:file-stat-last-change</code></p>	

sys:file-stat-last-modify	The times of the last access to the file, last change in the data of the file and the last modification of the file status, in seconds from 1 January 1970.
sys:file-stat-links	The number of hard links to the file.
sys:file-stat-device-type	The device type (sometimes called Rdev).

	<i>Function</i>				
Summary	Gets the path of a special folder on a Microsoft Windows or Mac OS X machine.				
Package	<code>system</code>				
Signature	<code>get-folder-path what &key create => result</code>				
Arguments	<table> <tr> <td><i>what</i></td><td>A keyword.</td></tr> <tr> <td><i>create</i></td><td>A boolean.</td></tr> </table>	<i>what</i>	A keyword.	<i>create</i>	A boolean.
<i>what</i>	A keyword.				
<i>create</i>	A boolean.				
Values	<table> <tr> <td><i>result</i></td><td>A directory pathname naming the path, or <code>nil</code>.</td></tr> </table>	<i>result</i>	A directory pathname naming the path, or <code>nil</code> .		
<i>result</i>	A directory pathname naming the path, or <code>nil</code> .				
Description	<p>The function <code>get-folder-path</code> obtains the current value for various special folders often used by applications. It is useful because these paths may differ between versions of the operating system. <code>get-folder-path</code> is implemented only on Microsoft Windows and Mac OS X.</p> <p><i>what</i> indicates the purpose of the special folder. For instance, <code>:common-appdata</code> means the folder containing application data for all users.</p>				

The following values are recognized on Microsoft Windows and Mac OS X:

```
:appdata, :documents, :my-documents, :common-appdata,  
:common-documents and :local-appdata.  
  
:documents is an alias for :my-documents.
```

The following values are recognized on Mac OS X only:

```
:my-library, :my-appsupport, :my-preferences, :my-  
caches, :my-logs, :common-library, :common-appsupport,  
:common-preferences, :common-caches, :common-logs,  
:system-library.
```

On Mac OS X, :appdata is an alias for :my-appsupport, :common-appdata is an alias for :common-appsupport, and :local-appdata is an alias for :common-appsupport.

If the folder does not exist and *create* is true, the folder is created. The default value of *create* is *nil*.

If the folder does exist, *result* is *nil*.

Compatibility note In LispWorks 5.0 and previous versions, `get-folder-path` returns a string.

Example This form constructs a pathname to a file `foo.lisp` in the user's documents directory:

```
(make-pathname  
  :name "foo"  
  :type "lisp"  
  :defaults  
  (sys:get-folder-path :my-documents))
```

See also `get-user-profile-directory`

get-user-profile-directory	<i>Function</i>
Summary	Gets the root of the user's profile on a Windows NT-based system.
Package	<code>system</code>
Signature	<code>get-user-profile-directory => result</code>
Values	<code>result</code> A directory pathname naming the path, or <code>nil</code> .
Description	<p>The function <code>get-user-profile-directory</code> obtains the path to the current user's profile folder on a Windows NT-based system (including Windows 2000, Windows XP and Windows Vista). <code>get-user-profile-directory</code> is implemented only on Microsoft Windows.</p> <p><code>result</code> names the root of the profile directory.</p> <p>Note that the default path for each user's profile may differ between versions of the operating system.</p>
Compatibility note	In LispWorks 5.0 and previous versions, <code>get-user-profile-directory</code> returns a string.
Example	<p>On Windows XP:</p> <pre>(sys:get-user-profile-directory) => #P"C:/Documents and Settings/fred/"</pre> <p>On Windows 98 SE:</p> <pre>(sys:get-user-profile-directory) => nil</pre>
See also	<code>get-folder-path</code>

guess-external-format	<i>Function</i>								
Summary	Tries to work out the external format								
Package	<code>system</code>								
Signature	<code>guess-external-format pathname ef-spec buffer length => ef-spec</code>								
Arguments	<table> <tr> <td><i>pathname</i></td><td>Pathname identifying location of <i>buffer</i>.</td></tr> <tr> <td><i>ef-spec</i></td><td>An external format spec.</td></tr> <tr> <td><i>buffer</i></td><td>A buffer whose contents are examined.</td></tr> <tr> <td><i>length</i></td><td>Length (an integer) up to which <i>buffer</i> should be examined.</td></tr> </table>	<i>pathname</i>	Pathname identifying location of <i>buffer</i> .	<i>ef-spec</i>	An external format spec.	<i>buffer</i>	A buffer whose contents are examined.	<i>length</i>	Length (an integer) up to which <i>buffer</i> should be examined.
<i>pathname</i>	Pathname identifying location of <i>buffer</i> .								
<i>ef-spec</i>	An external format spec.								
<i>buffer</i>	A buffer whose contents are examined.								
<i>length</i>	Length (an integer) up to which <i>buffer</i> should be examined.								
Values	<i>ef-spec</i> An external format spec.								
Description	<p>If <i>ef-spec</i> is complete, then it is returned. Otherwise <code>guess-external-format</code> calls, in turn, functions on the list <code>*file-encoding-detection-algorithm*</code>. If a complete external format spec is returned it is used, otherwise the return value is passed to the next function. If the name of the external format spec returned by the last function on this list is <code>:default</code>, an error of type <code>file-encoding-resolution-error</code> is signalled. The caller offers a restart for trying again with respecified <code>external-format</code> and/or <code>element-type</code> arguments. Otherwise <code>guess-external-format</code> proceeds to guess the <i>eol-style</i>.</p> <p>To guess the <i>eol-style</i>, functions on the list <code>*file-eol-style-detection-algorithm*</code> are called in turn. If a complete external format spec is returned it is used, otherwise the return value is passed to the next function. If the external format spec returned by the last function on this list does not contain <code>:eol-style</code>, an error of type <code>file-encoding-resolution-error</code> is signalled.</p>								

See also ***file-encoding-detection-algorithm***
file-eol-style-detection-algorithm
file-encoding-resolution-error

in-static-area *Macro*

Summary Allocates the objects produced by the specified forms to the static area.

Package **system**

Signature **in-static-area &rest body => result**

Arguments **body** The forms for which you want the garbage collector to allocate space in the static area.

Values **result** The result of executing *body*.

Description Allocates the objects produced by the specified forms to the static area. Objects in the static area are not moved, though they are garbage collected when there is no longer a pointer to the object.

Note: the macro **in-static-area** is deprecated. Use **make-array** with **:allocation :static** where possible instead.

Example (**system:in-static-area (make-string 10)**)

See also **enlarge-static**
make-array
staticcp

int32 *Type*

Summary A type used to generate optimal 32-bit arithmetic code.

Package	<code>system</code>
Signature	<code>int32</code>
Description	<p>The type <code>int32</code> is used to generate optimal 32-bit arithmetic code.</p> <p>Objects of type <code>int32</code> are generated and can be manipulated using the functions in the INT32 API but the compiler can optimize such source code by eliminating the intermediate <code>int32</code> objects to produce efficient raw 32-bit code.</p> <p>See the section "Fast 32-bit arithmetic" in the <i>LispWorks User Guide</i> for more information.</p>
See also	<code>int32*</code> <code>int32+</code> <code>int32-</code> <code>+int32-0+</code> <code>+int32-1+</code> <code>int32-1+</code> <code>int32-1-</code> <code>int32/</code> <code>int32/=</code> <code>int32<</code> <code>int32<=</code> <code>int32=</code> <code>int32></code> <code>int32>=</code> <code>int32-aref</code> <code>int32-logand</code> <code>int32-logandc1</code> <code>int32-logandc2</code> <code>int32-logeqv</code> <code>int32-logior</code> <code>int32-lognand</code> <code>int32-lognor</code> <code>int32-lognot</code>

```

int32-logorc1
int32-logorc2
int32-logxor
int32-minusp
int32-plusp
int32-to-integer
int32-zerop
integer-to-int32
make-simple-int32-vector
simple-int32-vector

```

int32*	<i>Function</i>				
Summary	The multiply operator for <code>int32</code> objects.				
Package	<code>system</code>				
Signature	<code>int32* x y => int32</code>				
Arguments	<table> <tr> <td><code>x</code></td><td>An <code>int32</code> object or an integer of type <code>(signed-byte 32)</code>.</td></tr> <tr> <td><code>y</code></td><td>An <code>int32</code> object or an integer of type <code>(signed-byte 32)</code>.</td></tr> </table>	<code>x</code>	An <code>int32</code> object or an integer of type <code>(signed-byte 32)</code> .	<code>y</code>	An <code>int32</code> object or an integer of type <code>(signed-byte 32)</code> .
<code>x</code>	An <code>int32</code> object or an integer of type <code>(signed-byte 32)</code> .				
<code>y</code>	An <code>int32</code> object or an integer of type <code>(signed-byte 32)</code> .				
Values	<code>int32</code> An <code>int32</code> object.				
Description	<p>The function <code>int32*</code> is the multiply operator for <code>int32</code> objects.</p> <p>See the section "Fast 32-bit arithmetic" in the <i>LispWorks User Guide</i> for more information about the INT32 API.</p>				
See also	<code>int32</code>				

		<i>Function</i>
Summary	The add operator for <code>int32</code> objects.	
Package	<code>system</code>	
Signature	<code>int32+ x y => int32</code>	
Arguments	<code>x</code>	An <code>int32</code> object or an integer of type <code>(signed-byte 32)</code> .
	<code>y</code>	An <code>int32</code> object or an integer of type <code>(signed-byte 32)</code> .
Values	<code>int32</code>	An <code>int32</code> object.
Description	The function <code>int32+</code> is the add operator for <code>int32</code> objects. See the section "Fast 32-bit arithmetic" in the <i>LispWorks User Guide</i> for more information about the INT32 API.	
See also	<code>int32</code>	

		<i>Function</i>
Summary	The subtract operator for <code>int32</code> objects.	
Package	<code>system</code>	
Signature	<code>int32- x y => int32</code>	
Arguments	<code>x</code>	An <code>int32</code> object or an integer of type <code>(signed-byte 32)</code> .
	<code>y</code>	An <code>int32</code> object or an integer of type <code>(signed-byte 32)</code> .
Values	<code>int32</code>	An <code>int32</code> object.

Description	The function <code>int32-</code> is the subtract operator for <code>int32</code> objects.
	See the section "Fast 32-bit arithmetic" in the <i>LispWorks User Guide</i> for more information about the INT32 API.
See also	<code>int32</code>

+int32-0+ *Symbol Macro*

Summary	Shorthand for <code>(sys:integer-to-int32 0)</code> .
Package	<code>system</code>
Description	The symbol macro <code>+int32-0+</code> expands to <code>(sys:integer-to-int32 0)</code> .
See also	<code>integer-to-int32</code>

+int32-1+ *Symbol Macro*

Summary	Shorthand for <code>(sys:integer-to-int32 1)</code> .
Package	<code>system</code>
Description	The symbol macro <code>+int32-1+</code> expands to <code>(sys:integer-to-int32 1)</code> .
See also	<code>integer-to-int32</code>

int32-1+ *Function*

Summary	The operator for <code>int32</code> objects corresponding to the function <code>1+</code> .
---------	---

Package	<code>system</code>	
Signature	<code>int32-1+ x => int32</code>	
Arguments	<code>x</code>	An <code>int32</code> object or an integer of type <code>(signed-byte 32)</code> .
Values	<code>int32</code>	An <code>int32</code> object.
Description		The function <code>int32-1+</code> is the operator for <code>int32</code> objects that corresponds to the function <code>1+</code> . See the section "Fast 32-bit arithmetic" in the <i>LispWorks User Guide</i> for more information about the INT32 API.
See also		<code>int32</code>

		<i>Function</i>
Summary		The operator for <code>int32</code> objects corresponding to the function <code>1-</code> .
Package	<code>system</code>	
Signature	<code>int32-1- x => int32</code>	
Arguments	<code>x</code>	An <code>int32</code> object or an integer of type <code>(signed-byte 32)</code> .
Values	<code>int32</code>	An <code>int32</code> object.
Description		The function <code>int32-1-</code> is the operator for <code>int32</code> objects that corresponds to the function <code>1-</code> . See the section "Fast 32-bit arithmetic" in the <i>LispWorks User Guide</i> for more information about the INT32 API.

See also [int32](#)

int32/ *Function*

Summary	The divide operator for <code>int32</code> objects.	
Package	<code>system</code>	
Signature	<code>int32/ x y => int32</code>	
Arguments	<code>x</code>	An <code>int32</code> object or an integer of type (<code>signed-byte 32</code>).
	<code>y</code>	An <code>int32</code> object that does not correspond to 0, or a non-zero integer of type (<code>signed-byte 32</code>).
Values	<code>int32</code>	An <code>int32</code> object.
Description	<p>The function <code>int32/</code> is the divide operator for <code>int32</code> objects. See the section "Fast 32-bit arithmetic" in the <i>LispWorks User Guide</i> for more information about the INT32 API.</p>	
See also	int32	

int32/= *Function*

Summary	The <code>/=</code> comparison for <code>int32</code> objects.	
Package	<code>system</code>	
Signature	<code>int32/= x y => result</code>	
Arguments	<code>x</code>	An <code>int32</code> object or an integer of type (<code>signed-byte 32</code>).

	<i>y</i>	An <code>int32</code> object or an integer of type <code>(signed-byte 32)</code> .
Values	<i>result</i>	A boolean.
Description	The function <code>int32/=</code> is the not equal comparison for <code>int32</code> objects. See the section "Fast 32-bit arithmetic" in the <i>LispWorks User Guide</i> for more information about the INT32 API.	
See also	<code>int32</code>	

int32<		<i>Function</i>
Summary		The < comparison for <code>int32</code> objects.
Package		<code>system</code>
Signature		<code>int32< x y => result</code>
Arguments	<i>x</i>	An <code>int32</code> object or an integer of type <code>(signed-byte 32)</code> .
	<i>y</i>	An <code>int32</code> object or an integer of type <code>(signed-byte 32)</code> .
Values	<i>result</i>	A boolean.
Description	The function <code>int32<</code> is the less than comparison for <code>int32</code> objects. See the section "Fast 32-bit arithmetic" in the <i>LispWorks User Guide</i> for more information about the INT32 API.	
See also	<code>int32</code>	

		<i>Function</i>
int32<<		
Summary		A shift left operator for <code>int32</code> objects.
Package		<code>system</code>
Signature		<code>int32<< x y => result</code>
Arguments	<code>x</code>	An <code>int32</code> object or an integer of type <code>(signed-byte 32)</code> .
	<code>y</code>	An <code>int32</code> object or an integer of type <code>(signed-byte 32)</code> .
Values	<code>result</code>	An <code>int32</code> object.
Description		The function <code>int32<<</code> is a shift left operator for <code>int32</code> objects. See the section "Fast 32-bit arithmetic" in the <i>LispWorks User Guide</i> for more information about the INT32 API.
See also		<code>int32</code>

		<i>Function</i>
int32<=		
Summary		The <code><=</code> comparison for <code>int32</code> objects.
Package		<code>system</code>
Signature		<code>int32<= x y => result</code>
Arguments	<code>x</code>	An <code>int32</code> object or an integer of type <code>(signed-byte 32)</code> .
	<code>y</code>	An <code>int32</code> object or an integer of type <code>(signed-byte 32)</code> .
Values	<code>result</code>	A boolean.

Description	The function <code>int32<=</code> is the less than or equal comparison for <code>int32</code> objects.
	See the section "Fast 32-bit arithmetic" in the <i>LispWorks User Guide</i> for more information about the INT32 API.
See also	<code>int32</code>

int32= *Function*

Summary	The <code>=</code> comparison for <code>int32</code> objects.	
Package	<code>system</code>	
Signature	<code>int32= x y => result</code>	
Arguments	<code>x</code>	An <code>int32</code> object or an integer of type <code>(signed-byte 32)</code> .
	<code>y</code>	An <code>int32</code> object or an integer of type <code>(signed-byte 32)</code> .
Values	<code>result</code>	A boolean.
Description	The function <code>int32=</code> is the equal comparison for <code>int32</code> objects.	
	See the section "Fast 32-bit arithmetic" in the <i>LispWorks User Guide</i> for more information about the INT32 API.	
See also	<code>int32</code>	

int32> *Function*

Summary	The <code>></code> comparison for <code>int32</code> objects.
---------	--

Package	<code>system</code>	
Signature	<code>int32> x y => result</code>	
Arguments	<code>x</code>	An <code>int32</code> object or an integer of type <code>(signed-byte 32)</code> .
	<code>y</code>	An <code>int32</code> object or an integer of type <code>(signed-byte 32)</code> .
Values	<code>result</code>	A boolean.
Description	The function <code>int32></code> is the greater than comparison for <code>int32</code> objects.	
	See the section "Fast 32-bit arithmetic" in the <i>LispWorks User Guide</i> for more information about the INT32 API.	
See also	<code>int32</code>	

int32>=		<i>Function</i>
Summary	The <code>>=</code> comparison for <code>int32</code> objects.	
Package	<code>system</code>	
Signature	<code>int32>= x y => result</code>	
Arguments	<code>x</code>	An <code>int32</code> object or an integer of type <code>(signed-byte 32)</code> .
	<code>y</code>	An <code>int32</code> object or an integer of type <code>(signed-byte 32)</code> .
Values	<code>result</code>	A boolean.
Description	The function <code>int32>=</code> is the greater than or equal comparison for <code>int32</code> objects.	

See the section "Fast 32-bit arithmetic" in the *LispWorks User Guide* for more information about the INT32 API.

See also [int32](#)

int32>> *Function*

Summary A shift right operator for `int32` objects.

Package `system`

Signature `int32>> x y => result`

Arguments `x` An `int32` object or an integer of type
`(signed-byte 32)`.

`y` An `int32` object or an integer of type
`(signed-byte 32)`.

Values `result` An `int32` object.

Description The function `int32>>` is a shift right operator for `int32` objects.

See the section "Fast 32-bit arithmetic" in the *LispWorks User Guide* for more information about the INT32 API.

See also [int32](#)

int32-aref *Function*

Summary The accessor for a `simple-int32-vector`.

Package `system`

Signature `int32-aref vector index => int32`

	(setf int32-aref) x vector index => int32	
Arguments	vector	An simple-int32-vector .
	index	A non-negative fixnum.
	x	An int32 object or an integer of type (signed-byte 32) .
Values	int32	An int32 object.
Description	The function int32-aref is the accessor for a simple-int32-vector . The reader returns an int32 object for the value at index index in vector . The writer sets the value at index index in vector to the int32 object or integer x supplied. See the section "Fast 32-bit arithmetic" in the <i>LispWorks User Guide</i> for more information about the INT32 API.	
See also	int32 simple-int32-vector	

		<i>Function</i>
Summary	The logand operator for int32 objects.	
Package	system	
Signature	int32-logand x y => int32	
Arguments	x	An int32 object or an integer of type (signed-byte 32) .
	y	An int32 object or an integer of type (signed-byte 32) .
Values	int32	An int32 object.

Description	The function <code>int32-logand</code> is the bitwise logical 'and' operator for <code>int32</code> objects.
	See the section "Fast 32-bit arithmetic" in the <i>LispWorks User Guide</i> for more information about the INT32 API.
See also	<code>int32</code>

int32-logandc1 *Function*

Summary	The <code>logandc1</code> operator for <code>int32</code> objects.	
Package	<code>system</code>	
Signature	<code>int32-logandc1 x y => int32</code>	
Arguments	<code>x</code>	An <code>int32</code> object or an integer of type <code>(signed-byte 32)</code> .
	<code>y</code>	An <code>int32</code> object or an integer of type <code>(signed-byte 32)</code> .
Values	<code>int32</code>	An <code>int32</code> object.

Description	The function <code>int32-logandc1</code> is the bitwise logical operator for <code>int32</code> objects which 'ands' the complement of <code>x</code> with <code>y</code> .
	See the section "Fast 32-bit arithmetic" in the <i>LispWorks User Guide</i> for more information about the INT32 API.
See also	<code>int32</code>

int32-logandc2 *Function*

Summary	The <code>logandc2</code> operator for <code>int32</code> objects.
---------	--

Package	<code>system</code>	
Signature	<code>int32-logandc2 x y => int32</code>	
Arguments	<code>x</code>	An <code>int32</code> object or an integer of type <code>(signed-byte 32)</code> .
	<code>y</code>	An <code>int32</code> object or an integer of type <code>(signed-byte 32)</code> .
Values	<code>int32</code>	An <code>int32</code> object.
Description	<p>The function <code>int32-logandc2</code> is the bitwise logical operator for <code>int32</code> objects which 'ands' <code>x</code> with the complement of <code>y</code>.</p> <p>See the section "Fast 32-bit arithmetic" in the <i>LispWorks User Guide</i> for more information about the INT32 API.</p>	
See also	<code>int32</code>	

int32-logbitp		<i>Function</i>
Summary	The <code>logbitp</code> operator for <code>int32</code> objects.	
Package	<code>system</code>	
Signature	<code>int32-logbitp index x => result</code>	
Arguments	<code>index</code>	An <code>int32</code> object or an integer of type <code>(signed-byte 32)</code> .
	<code>x</code>	An <code>int32</code> object or an integer of type <code>(signed-byte 32)</code> .
Values	<code>result</code>	An boolean.

Description	The function <code>int32-logbitp</code> is the test for <code>int32</code> objects which returns <code>t</code> if the bit at index <code>index</code> in <code>x</code> is 1, and <code>nil</code> if it is 0.
	See the section "Fast 32-bit arithmetic" in the <i>LispWorks User Guide</i> for more information about the INT32 API.
See also	<code>int32</code>

int32-logeqv *Function*

Summary	The <code>logeqv</code> operator for <code>int32</code> objects.	
Package	<code>system</code>	
Signature	<code>int32-logeqv x y => int32</code>	
Arguments	<code>x</code>	An <code>int32</code> object or an integer of type <code>(signed-byte 32)</code> .
	<code>y</code>	An <code>int32</code> object or an integer of type <code>(signed-byte 32)</code> .
Values	<code>int32</code>	An <code>int32</code> object.
Description	The function <code>int32-logeqv</code> is the bitwise logical operator for <code>int32</code> objects which returns the complement of the 'exclusive or' of <code>x</code> and <code>y</code> .	
	See the section "Fast 32-bit arithmetic" in the <i>LispWorks User Guide</i> for more information about the INT32 API.	
See also	<code>int32</code>	

int32-logior		<i>Function</i>
Summary		The <code>logior</code> operator for <code>int32</code> objects.
Package		<code>system</code>
Signature		<code>int32-logior x y => int32</code>
Arguments	<code>x</code>	An <code>int32</code> object or an integer of type <code>(signed-byte 32)</code> .
	<code>y</code>	An <code>int32</code> object or an integer of type <code>(signed-byte 32)</code> .
Values	<code>int32</code>	An <code>int32</code> object.
Description		The function <code>int32-logior</code> is the bitwise logical 'inclusive or' operator for <code>int32</code> objects. See the section "Fast 32-bit arithmetic" in the <i>LispWorks User Guide</i> for more information about the INT32 API.
See also		<code>int32</code>

int32-logand		<i>Function</i>
Summary		The <code>logand</code> operator for <code>int32</code> objects.
Package		<code>system</code>
Signature		<code>int32-logand x y => int32</code>
Arguments	<code>x</code>	An <code>int32</code> object or an integer of type <code>(signed-byte 32)</code> .
	<code>y</code>	An <code>int32</code> object or an integer of type <code>(signed-byte 32)</code> .

Values	<i>int32</i>	An <code>int32</code> object.
Description	The function <code>int32-logand</code> is the bitwise logical operator for <code>int32</code> objects which returns the complement of the 'and' of <i>x</i> and <i>y</i> .	See the section "Fast 32-bit arithmetic" in the <i>LispWorks User Guide</i> for more information about the INT32 API.
See also	<code>int32</code>	

int32-lognor *Function*

Summary	The <code>lognor</code> operator for <code>int32</code> objects.	
Package	<code>system</code>	
Signature	<code>int32-lognor x y => int32</code>	
Arguments	<i>x</i>	An <code>int32</code> object or an integer of type (<code>signed-byte 32</code>).
	<i>y</i>	An <code>int32</code> object or an integer of type (<code>signed-byte 32</code>).
Values	<i>int32</i>	An <code>int32</code> object.
Description	The function <code>int32-lognor</code> is the bitwise logical operator for <code>int32</code> objects which returns the complement of the 'inclusive or' of <i>x</i> and <i>y</i> .	See the section "Fast 32-bit arithmetic" in the <i>LispWorks User Guide</i> for more information about the INT32 API.
See also	<code>int32</code>	

int32-lognot	<i>Function</i>
Summary	The <code>lognot</code> operator for an <code>int32</code> object.
Package	<code>system</code>
Signature	<code>int32-lognot x => int32</code>
Arguments	<code>x</code> An <code>int32</code> object or an integer of type <code>(signed-byte 32)</code> .
Values	<code>int32</code> An <code>int32</code> object.
Description	The function <code>int32-lognot</code> is the bitwise logical operator for <code>int32</code> objects which returns the complement of its argument <code>X</code> . See the section "Fast 32-bit arithmetic" in the <i>LispWorks User Guide</i> for more information about the INT32 API.
See also	<code>int32</code>

int32-logorc1	<i>Function</i>
Summary	The <code>logorc1</code> operator for <code>int32</code> objects.
Package	<code>system</code>
Signature	<code>int32-logorc1 x y => int32</code>
Arguments	<code>x</code> An <code>int32</code> object or an integer of type <code>(signed-byte 32)</code> . <code>y</code> An <code>int32</code> object or an integer of type <code>(signed-byte 32)</code> .
Values	<code>int32</code> An <code>int32</code> object.

Description	The function <code>int32-logorc1</code> is the bitwise logical operator for <code>int32</code> objects which 'inclusive ors' the complement of <code>x</code> with <code>y</code> . See the section "Fast 32-bit arithmetic" in the <i>LispWorks User Guide</i> for more information about the INT32 API.
See also	<code>int32</code>

int32-logorc2 *Function*

Summary	The <code>logorc2</code> operator for <code>int32</code> objects.	
Package	<code>system</code>	
Signature	<code>int32-logorc2 x y => int32</code>	
Arguments	<code>x</code>	An <code>int32</code> object or an integer of type <code>(signed-byte 32)</code> .
	<code>y</code>	An <code>int32</code> object or an integer of type <code>(signed-byte 32)</code> .
Values	<code>int32</code>	An <code>int32</code> object.
Description	The function <code>int32-logorc2</code> is the bitwise logical operator for <code>int32</code> objects which 'inclusive ors' <code>x</code> with the complement of <code>y</code> . See the section "Fast 32-bit arithmetic" in the <i>LispWorks User Guide</i> for more information about the INT32 API.	
See also	<code>int32</code>	

int32-logtest	<i>Function</i>				
Summary	The logtest operator for int32 objects.				
Package	system				
Signature	int32-logtest x y => result				
Arguments	<table border="0"> <tr> <td>x</td> <td>An int32 object or an integer of type (signed-byte 32).</td> </tr> <tr> <td>y</td> <td>An int32 object or an integer of type (signed-byte 32).</td> </tr> </table>	x	An int32 object or an integer of type (signed-byte 32) .	y	An int32 object or an integer of type (signed-byte 32) .
x	An int32 object or an integer of type (signed-byte 32) .				
y	An int32 object or an integer of type (signed-byte 32) .				
Values	result An boolean.				
Description	<p>The function int32-logtest is the bitwise test for int32 objects which returns t if any of the bits designated by 1 in x is 1 in y, and returns nil otherwise.</p> <p>See the section "Fast 32-bit arithmetic" in the <i>LispWorks User Guide</i> for more information about the INT32 API.</p>				
See also	int32				

int32-logxor	<i>Function</i>				
Summary	The logxor operator for int32 objects.				
Package	system				
Signature	int32-logxor x y => int32				
Arguments	<table border="0"> <tr> <td>x</td> <td>An int32 object or an integer of type (signed-byte 32).</td> </tr> <tr> <td>y</td> <td>An int32 object or an integer of type (signed-byte 32).</td> </tr> </table>	x	An int32 object or an integer of type (signed-byte 32) .	y	An int32 object or an integer of type (signed-byte 32) .
x	An int32 object or an integer of type (signed-byte 32) .				
y	An int32 object or an integer of type (signed-byte 32) .				

Values	<i>int32</i>	An <i>int32</i> object.
Description	The function <code>int32-logxor</code> is the bitwise logical 'exclusive or' operator for <i>int32</i> objects. See the section "Fast 32-bit arithmetic" in the <i>LispWorks User Guide</i> for more information about the INT32 API.	
See also	<code>int32</code>	

int32-minusp *Function*

Summary	The <code>minusp</code> test for an <i>int32</i> object.	
Package	<code>system</code>	
Signature	<code>int32-minusp x => result</code>	
Arguments	<i>x</i>	An <i>int32</i> object or an integer of type <code>(signed-byte 32)</code> .
Values	<i>result</i>	A boolean.
Description	The function <code>int32-minusp</code> tests whether its argument <i>x</i> is <code>int32<</code> than the value of <code>+int32-0+</code> . See the section "Fast 32-bit arithmetic" in the <i>LispWorks User Guide</i> for more information about the INT32 API.	
See also	<code>int32</code>	

int32-plusp *Function*

Summary	The <code>plusp</code> test for an <i>int32</i> object.	
Package	<code>system</code>	

Signature	<code>int32-plusp x => result</code>	
Arguments	<code>x</code>	An <code>int32</code> object or an integer of type <code>(signed-byte 32)</code> .
Values	<code>result</code>	A boolean.
Description	<p>The function <code>int32-plusp</code> tests whether its argument <code>x</code> is <code>int32></code> than the value of <code>+int32-0+</code>.</p> <p>See the section "Fast 32-bit arithmetic" in the <i>LispWorks User Guide</i> for more information about the INT32 API.</p>	
See also	<code>int32</code>	

int32-to-integer *Function*

Summary	The destructor converting an <code>int32</code> object to an integer.	
Package	<code>system</code>	
Signature	<code>int32-to-integer int32 => integer</code>	
Arguments	<code>int32</code>	An <code>int32</code> object or an integer of type <code>(signed-byte 32)</code> .
Values	<code>integer</code>	An integer of type <code>(signed-byte 32)</code> .
Description	<p>The function <code>int32-to-integer</code> returns an integer <code>integer</code> of type <code>(signed-byte 32)</code> corresponding to the <code>int32</code> object <code>int32</code>. The argument <code>int32</code> can also be an integer of type <code>(signed-byte 32)</code>, in which case it is simply returned.</p> <p>An error is signalled if <code>int32</code> is not of type <code>int32</code> or <code>(signed-byte 32)</code>.</p> <p>See the section "Fast 32-bit arithmetic" in the <i>LispWorks User Guide</i> for more information about the INT32 API.</p>	

See also [int32](#)

int32-zerop *Function*

Summary The `zerop` test for an `int32` object.

Package `system`

Signature `int32-zerop x => result`

Arguments `x` An `int32` object or an integer of type
(`signed-byte 32`).

Values `result` A boolean.

Description The function `int32-zerop` tests whether its argument `x` is `int32=` to the value of `+int32-0+`.

See the section "Fast 32-bit arithmetic" in the *LispWorks User Guide* for more information about the INT32 API.

See also [int32](#)

integer-to-int32 *Function*

Summary The constructor for `int32` objects.

Package `system`

Signature `integer-to-int32 integer => int32`

Arguments `integer` An integer of type (`signed-byte 32`).

Values `int32` An `int32` object.

Description	The function <code>integer-to-int32</code> constructs an <code>int32</code> object from an integer. An error is signalled if <code>integer</code> is not of type <code>(signed-byte 32)</code> . See the section "Fast 32-bit arithmetic" in the <i>LispWorks User Guide</i> for more information about the INT32 API.
See also	<code>int32</code>

line-arguments-list *Variable*

Summary	List of the command line arguments used when LispWorks was invoked.
Package	<code>system</code>
Initial Value	<code>nil</code>
Description	<p>This variable contains a list of strings. These are the arguments with which LispWorks was called, in the same order. The first element is the executable itself.</p> <p>You can implement command line processing in your application by testing elements in *line-arguments-list*. Use a string comparison function such as <code>string=</code> to compare them.</p> <p>For a description of the command line arguments processed by LispWorks, see "The Command Line" in the <i>LispWorks User Guide</i>.</p>
See also	<code>lisp-image-name</code>

	<i>Function</i>				
Summary	Loads a fasl file created by <code>dump-forms-to-file</code> or <code>with-output-to-fasl-file</code> .				
Package	<code>system</code>				
Signature	<code>load-data-file file &rest args => result</code>				
Arguments	<table> <tr> <td><code>file</code></td><td>A pathname designator.</td></tr> <tr> <td><code>args</code></td><td>Arguments passed to <code>load</code>.</td></tr> </table>	<code>file</code>	A pathname designator.	<code>args</code>	Arguments passed to <code>load</code> .
<code>file</code>	A pathname designator.				
<code>args</code>	Arguments passed to <code>load</code> .				
Values	<code>result</code> A generalized boolean.				
Description	<p>The function <code>load-data-file</code> loads a fasl file created by <code>dump-forms-to-file</code> or <code>with-output-to-fasl-file</code>.</p> <p><code>load-data-file</code> has the same semantics as <code>load</code>, but treats fasl files differently:</p> <ul style="list-style-type: none"> • it cannot load a fasl generated by <code>compile-file</code>. • it allows loading of fasls generated by <code>dump-forms-to-file</code> or <code>with-output-to-fasl-file</code>, including those generated by a previous version of LispWorks. <p><code>load-data-file</code> is intended to work with data files generated in a previous version of LispWorks. In particular you can load data files generated by LispWorks 4.3, LispWorks 4.4 and LispWorks 5.0 into LispWorks 5.1.</p> <p>Fasl files generated by <code>dump-forms-to-file</code> or <code>with-output-to-fasl-file</code> must only be loaded using <code>load-data-file</code>.</p> <p>The pathname specified by <code>file</code> must be recognized as a fasl file type, otherwise <code>load-data-file</code> will load it as a text file.</p>				
Compatibility Note	The default fasl file type in LispWorks 5.0 and later differs to LispWorks 4.x on Windows and Linux, as described in <code>compile-file</code> . Therefore you may need to do something like this				

to ensure your LispWorks 4.x data file is recognized as a fasl file when loading it in this version of LispWorks:

```
(let ((sys::*binary-file-types*
      (cons "fsl" sys::*binary-file-types*)))
  (sys:load-data-file "C:/temp/data.fsl"))
```

Compatibility Note	The <code>fixnum</code> type in LispWorks 5.0 and later is larger than in LispWorks 4.x on Windows and Linux. A <code>bignum</code> dumped in a LispWorks 4.x data file will be loaded as a <code>fixnum</code> in LispWorks 5.0 and later if its value is within the <code>fixnum</code> range.
See also	<code>dump-forms-to-file</code> <code>with-output-to-fasl-file</code>

	<i>Function</i>								
Summary	Provides an encoding corresponding to the current code page on Microsoft Windows, and the locale on Unix.								
Package	<code>system</code>								
Signature	<code>locale-file-encoding pathname ef-spec buffer length => new-ef-spec</code>								
Arguments	<table> <tr> <td><code>pathname</code></td><td>Pathname identifying location of <code>buffer</code>.</td></tr> <tr> <td><code>ef-spec</code></td><td>An external format spec.</td></tr> <tr> <td><code>buffer</code></td><td>A buffer whose contents are examined.</td></tr> <tr> <td><code>length</code></td><td>Length (an integer) up to which <code>buffer</code> should be examined.</td></tr> </table>	<code>pathname</code>	Pathname identifying location of <code>buffer</code> .	<code>ef-spec</code>	An external format spec.	<code>buffer</code>	A buffer whose contents are examined.	<code>length</code>	Length (an integer) up to which <code>buffer</code> should be examined.
<code>pathname</code>	Pathname identifying location of <code>buffer</code> .								
<code>ef-spec</code>	An external format spec.								
<code>buffer</code>	A buffer whose contents are examined.								
<code>length</code>	Length (an integer) up to which <code>buffer</code> should be examined.								
Values	<table> <tr> <td><code>new-ef-spec</code></td><td>Default external format spec created by merging <code>ef-spec</code> with the encoding that was found.</td></tr> </table>	<code>new-ef-spec</code>	Default external format spec created by merging <code>ef-spec</code> with the encoding that was found.						
<code>new-ef-spec</code>	Default external format spec created by merging <code>ef-spec</code> with the encoding that was found.								

Description	The function <code>locale-file-encoding</code> consults the ANSI code page on Microsoft Windows. If the code page identifier is in <code>win32:*latin-1-code-pages*</code> , <code>locale-file-encoding</code> merges <code>ef-spec</code> with <code>:latin-1</code> . This external format writes Latin-1 on output, giving an error for any non-Latin-1 characters that are written. If the code page identifier is not in <code>win32:*latin-1-code-pages*</code> then <code>locale-file-encoding</code> merges <code>ef-spec</code> with an encoding corresponding to the current code page that gives an error for characters that cannot be encoded. <code>locale-file-encoding</code> merges <code>ef-spec</code> with <code>:latin-1</code> on Unix.
See also	<ul style="list-style-type: none"> <code>*file-encoding-detection-algorithm*</code> <code>*latin-1-code-pages*</code> <code>*multibyte-code-page-ef*</code> <code>safe-locale-file-encoding</code>

make-gesture-spec		<i>Function</i>
Summary		Create a Gesture Spec object.
Package		<code>system</code>
Signature		<code>make-gesture-spec data modifiers => gspec</code>
Arguments	<code>data</code>	A non-negative integer less than <code>char-code-limit</code> , or a Gesture Spec keyword, or <code>nil</code> .
	<code>modifiers</code>	A non-negative integer less than 64, or <code>nil</code> .
Values	<code>gspec</code>	A Gesture Spec object
Description	The function <code>make-gesture-spec</code> returns a new Gesture Spec object <code>gspec</code> . This can be used to represent a keystroke consisting of the key indicated by <code>data</code> , modified by the modifier keys indicated by <code>modifiers</code> .	

If *data* is an integer, it represents the key (`code-char data`). If *data* is a keyword, it must be one of the known Gesture Spec keywords and represents the key with the same name. If *data* is `nil`, then *gspec* has a wild data component.

If *modifiers* is an integer, it represents modifier keys according to the values `gesture-spec-accelerator-bit`, `gesture-spec-control-bit`, `gesture-spec-hyper-bit`, `gesture-spec-meta-bit`, `gesture-spec-shift-bit`, and `gesture-spec-super-bit`. If *modifiers* is `nil`, then *gspec* has a wild modifiers component.

The gesture `shift+x` could potentially be represented by the unmodified uppercase character *x*, or lowercase *x* with the `shift` modifier. In order to ensure a consistent representation the latter form is not supported by Gesture Specs. That is, a `both-case-p` character may not be combined with the single modifier `shift` in the accelerator argument. A `both-case-p` character is allowed with `shift` if there are other modifiers. See the below for examples.

Wild Gesture Specs can be useful when specifying an input model for a `capi:output-pane`.

Example

```
(sys:make-gesture-spec
  97
  (logior sys:gesture-spec-control-bit
          sys:gesture-spec-meta-bit))
```

A `both-case-p` character may not be combined with the single modifier `shift` in the accelerator argument, so code like this signals an error:

```
(sys:make-gesture-spec
  (char-code #\x)
  sys:gesture-spec-shift-bit)
```

Instead you should use:

```
(sys:make-gesture-spec (char-code #\X) 0)
```

A `both-case-p` character is allowed with `shift` if there are other modifiers:

```
(sys:make-gesture-spec
  (char-code #\x)
  (logior sys:gesture-spec-shift-bit
    sys:gesture-spec-meta-bit))
```

See also

`gesture-spec-accelerator-bit`
`gesture-spec-control-bit`
`gesture-spec-data`
`gesture-spec-hyper-bit`
`gesture-spec-meta-bit`
`gesture-spec-modifiers`
`gesture-spec-p`
`gesture-spec-shift-bit`
`gesture-spec-super-bit`
`print-pretty-gesture-spec`

make-simple-int32-vector

Function

Summary	The constructor for <code>simple-int32-vector</code> objects.	
Package	<code>system</code>	
Signature	<code>make-simple-int32-vector length &key initial-contents initial-element => vector</code>	
Arguments	<code>length</code>	A non-negative fixnum.
	<code>initial-contents</code>	A sequence of integers of type (<code>signed-byte 32</code>), or <code>nil</code> .
	<code>initial-element</code>	An integer of type (<code>signed-byte 32</code>).
Values	<code>vector</code>	A <code>simple-int32-vector</code> .
Description	<p>The function <code>make-simple-int32-vector</code> is the constructor for <code>simple-int32-vector</code> objects.</p> <p>The argument <code>initial-contents</code>, if supplied, should be a sequence of length <code>length</code>. It specifies the contents of <code>vector</code>.</p>	

The argument *initial-element*, if supplied, specifies the contents of *vector*.

An error is signalled if both *initial-contents* and *initial-element* are supplied.

See the section "Fast 32-bit arithmetic" in the *LispWorks User Guide* for more information about the INT32 API.

See also `int32`
`simple-int32-vector`

make-stderr-stream *Function*

Summary Returns an output stream connected to stderr.

Package `system`

Signature `make-stderr-stream => stream`

Arguments None.

Values `stream` An output stream.

Description The function `make-stderr-stream` returns an output stream connected to stderr.

On Microsoft Windows, you should take care to not close this stream or make multiple stderr streams.

make-typed-aref-vector *Function*

Summary Makes a vector that can be accessed efficiently.

Package `system`

Signature `make-typed-aref-vector byte-length => vector`

Arguments	<i>byte-length</i>	A non-negative fixnum.
Values	<i>vector</i>	A vector.
Description	The function <code>make-typed-aref-vector</code> returns a vector which is suitable for efficient access at compiler optimization level safety = 0. Use <code>typed-aref</code> to access <i>vector</i> efficiently.	
See also		<code>typed-aref</code>

		<i>Function</i>
Summary		Performs a Marking GC in 64-bit LispWorks.
Package		<code>system</code>
Signature		<code>marking-gc gen-num &key what-to-copy max-size fragmentation-threshold</code>
Arguments	<i>gen-num</i>	An integer in the inclusive range [0,7].
	<i>what-to-copy</i>	One of the keywords :cons, :symbol, :function, :non-pointer, :other, :weak, :all or :default.
	<i>max-size</i>	A positive number or <code>nil</code> .
	<i>fragmentation-threshold</i>	A number in the inclusive range [0, 10].
Description	The function <code>marking-gc</code> garbage collects (GCs) the generation specified by <i>gen-num</i> , and all younger generations. It uses mark and sweep, rather than copy. Mark and sweep garbage collection uses less virtual memory during its operation, but leaves the memory fragmented, which has a detrimental effect on the performance of the sys-	

tem afterwards. It is therefore not used automatically by the system, except to garbage collect static objects.

`marking-gc` is useful when you want to GC a generation which contains large amount (gigabytes) of data, to make sure there are no spurious pointers from this generation to a younger generation, and you do not expect many objects in the large generation to be collected. In this scenario, a Copying GC would use virtual memory which is almost double the size of the large generation during its operation, and so would possibly cause heavy paging.

Since repeated use of `marking-gc` will cause a lot of fragmentation, the arguments `what-to-copy` and `max-size` can be used to specify that part of the data should be collected by copying. Restricting the copying GC will reduce the amount of fragmentation that occurs.

`what-to-copy` specifies the allocation type to copy. It can be one of the main allocation types or `:weak`, meaning copy only objects in segments of that type. `what-to-copy` can also be `:all`, meaning copy objects in all segments. If `what-to-copy` is `:default` then each call to `marking-gc` chooses one of the main allocation types or `:weak` to copy, and successive calls with `:default` cycle through these allocation types.

`max-size` can be used to limit the amount that is copied, and thus limit the virtual memory that the operation needs. If `max-size` is non-`nil`, it specifies the limit, in gigabytes, of memory that can be used for copying. If there is more than `max-size` gigabytes of data of the type `what-to-copy`, the rest of this data is garbage collected by marking. The default value of `max-size` is `nil`, which means there is no limit on the amount that is copied.

`fragmentation-threshold` should be a number between 0 and 10. It specifies a minimum ratio between the free area in a segment that cannot be easily used for more allocation and the allocated area in this segment. Segments that are below this

threshold are not copied. The default value of *fragmentation-threshold* is 1.

Note: this function is implemented only in 64-bit LispWorks. It is not relevant to the Memory Management API in 32-bit implementations.

See also

`gc-generation`
`set-blocking-gen-num`

memory-growth-margin *Function*

Summary Returns the difference between the top of the Lisp heap and a maximum memory limit in 32-bit LispWorks.

Package `system`

Signature `memory-growth-margin => result`

Values `result` An integer address, or `nil`.

Description If a limit on the maximum memory has been set by `set-maximum-memory`, then `memory-growth-margin` returns the difference between the current top of the Lisp heap and that limit. That is, the amount by which the heap can grow.

Otherwise `memory-growth-margin` returns `nil`. This is the default behaviour.

Note: `memory-growth-margin` is implemented only in 32-bit LispWorks. It is not relevant to the Memory Management API in 64-bit implementations.

See also

`set-maximum-memory`

merge-ef-specs	<i>Function</i>				
Summary	Creates a new external format spec from two other external format specs.				
Package	<code>system</code>				
Signature	<code>merge-ef-specs ef-spec1 ef-spec2 => ef-spec</code>				
Arguments	<table> <tr> <td><i>ef-spec1</i></td><td>An external format spec.</td></tr> <tr> <td><i>ef-spec2</i></td><td>An external format spec.</td></tr> </table>	<i>ef-spec1</i>	An external format spec.	<i>ef-spec2</i>	An external format spec.
<i>ef-spec1</i>	An external format spec.				
<i>ef-spec2</i>	An external format spec.				
Values	<i>ef-spec</i> The resultant external format spec created from information in <i>ef-spec1</i> and <i>ef-spec2</i> .				
Description	<p>The function <code>merge-ef-specs</code> returns an external format spec constructed by adding information not supplied in <i>ef-spec1</i> from <i>ef-spec2</i>.</p> <p>Each external format spec argument is either a symbol or a list.</p> <p>If <i>ef-spec1</i> and <i>ef-spec2</i> have the same value for their name component (whether they are lists or symbols), return <i>ef-spec1</i> combined with any parameters from <i>ef-spec2</i> that are not specified in <i>ef-spec1</i>.</p> <p>Otherwise, if <i>ef-spec1</i> is <code>:default</code> or a list beginning with <code>:default</code>, return <i>ef-spec2</i> with parameters modified to be a union of the parameters from <i>ef-spec1</i> and <i>ef-spec2</i>, with those from <i>ef-spec1</i> taking priority.</p> <p>Otherwise, return <i>ef-spec1</i> with any <code>:eol-style</code> parameter from <i>ef-spec2</i> if <i>ef-spec1</i> does not specify <code>:eol-style</code>.</p>				

object-address	<i>Function</i>
Summary	Returns the address of the given <i>object</i> as an integer.

Package	<code>system</code>	
Signature	<code>object-address object => address</code>	
Arguments	<code>object</code>	The object whose address should be returned.
Values	<code>address</code>	The address of <code>object</code> . An integer.
Description		Returns the address of the given <code>object</code> as an integer. Note that the address is likely to change during garbage collection so this integer should be used for debugging purposes only.
Example		This shows that the address returned by <code>sys:object-address</code> is the same as the one printed by the <code>print-object</code> method for <code>generic-function</code> .
		<pre>CL-USER 1 > (let ((gf #'initialize-instance)) (format t "address = ~X~%gf = ~S" (sys:object-address gf) gf)) address = 1cff778 gf = #<STANDARD-GENERIC-FUNCTION INITIALIZE-INSTANCE 1cff778> NIL CL-USER 2 ></pre>
See also		<code>pointer-from-address</code>

open-pipe		<i>Function</i>
Summary		Runs a command in a subshell.
Package	<code>system</code>	
Signature		<code>open-pipe command &key direction element-type interrupt-off shell-type => stream</code>

Arguments	<i>command</i>	A string, a list of strings, a simple-vector of strings, or <code>nil</code> .
	<i>direction</i>	<code>:input</code> , <code>:output</code> or <code>:io</code> .
	<i>element-type</i>	A type specifier.
	<i>interrupt-off</i>	A boolean. Not implemented on Microsoft Windows.
	<i>shell-type</i>	A shell type.
Values	<i>stream</i>	A pipe stream.
Description		<p>On Unix/Linux/Mac OS X the behaviour of <code>open-pipe</code> is analogous to that of <code>popen</code> in the UNIX library. It creates a pipe to/from a subprocess and returns a stream. The stream can be read from or written to as appropriate.</p> <p>On Microsoft Windows <code>open-pipe</code> calls <code>CreateProcess</code> and <code>CreatePipe</code> and returns a bidirectional stream.</p> <p>If <i>command</i> is a string then it is passed to the shell as the command to run without any arguments. If <i>command</i> is a list, then its first element is the command to run directly and the other elements are passed as arguments on the command line (that is, element 0 has its name in <code>argv[0]</code> in C, and so on). If <i>command</i> is a simple-vector of strings, the element at index 0 is the command to run and the other elements are the complete set of arguments seen by the command (that is, element 1 becomes <code>argv[0]</code> in C, and so on). If <i>command</i> is <code>nil</code>, then the shell is run.</p> <p><i>direction</i> is a keyword for the stream direction. The default value is <code>:input</code>. Bidirectional (I/O) pipes may be created by passing <code>:io</code>. See the example below. This argument is ignored on Microsoft Windows.</p> <p><i>element-type</i> specifies the type of the stream as with <code>open</code>. The default value is <code>base-char</code>. This argument is ignored on Microsoft Windows.</p>

interrupt-off, if t, ensures that **ctrl+c** (SIGINT) to the Lisp-Works image is ignored by the subprocess. This argument is not implemented on Microsoft Windows.

shell-type specifies the type of shell to run. On UNIX/Linux/Mac OS X/FreeBSD the default value is `"/bin/sh"`. On Microsoft Windows the default value is `"cmd"`. Note that on Windows ME/98/95 you will need to pass `"command"`.

stream supports mixed character and binary I/O in the same way as file streams constructed by **open**.

Examples

Example on Unix:

```
CL-USER 1 > (setf *ls* (sys:open-pipe "ls"))
Warning: Setting unbound variable *LS*
#<SYSTEM::PIPE-STREAM "ls">

CL-USER 2 > (loop while
                  (print (read-line *ls* nil nil)))

"hello"
"othello"
NIL
NIL

CL-USER 3 > (close *ls*)
T
```

The following example shows you how to use bidirectional pipes.

```
CL-USER 1 > (with-open-stream
                  (s (sys:open-pipe "/bin/csh"
                                    :direction :io))
                  (write-line "whereis ls" s)
                  (force-output s)
                  (read-line s))
"ls: /sbin/ls /usr/bin/ls /usr/share/man/man1.Z/ls.1"
NIL
```

Example on Microsoft Windows

```

CL-USER 40 > (setf *ls* (sys:open-pipe "dir"))
#<WIN32::TWO-WAY-PIPE-STREAM 205F03F4>

CL-USER 41 > (loop while
                  (print (read-line *ls* nil nil)))

" Volume in drive Z is lispsrc"
" Volume Serial Number is 82E3-1342"
"""
" Directory of Z:\\v42\\delivery-tests"
"""
"20/02/02 11:57a      <DIR>          .
"20/02/02 11:57a      <DIR>          ..
"14/02/02 07:04p      6,815,772 othello.exe"
"14/02/02 07:07p      6,553,628 hello.exe"
"                         4 File(s)    13,369,400 bytes"
"                         3,974,103,040 bytes free"
NIL
NIL

CL-USER 42 > (close *ls*)
T

```

See also [call-system](#)
[call-system-showing-output](#)

open-url

Function

Summary	Displays a HTML page in a web browser.	
Package	system	
Signature	<code>open-url <i>url</i></code>	
Arguments	<i>url</i>	A string.
Description	The function <code>open-url</code> displays the page at the URL <i>url</i> in a web browser.	

Supported browsers are Netscape, Firefox, Mozilla, Opera on all platforms, Microsoft Internet Explorer on Microsoft Windows and Mac OS X, plus Safari on Mac OS X.

`open-url` is defined in the "hqn-web" module.

Compatibility Note	If your code uses the unsupported function <code>hqn-web:browse</code> please change to use <code>open-url</code> in LispWorks 5.0 and later.
Examples	(<code>sys:open-url "www.lispworks.com"</code>)
See also	<code>*browser-location*</code>

pid-exit-status *Function*

Summary	Returns the exit status of a process executed with <code>run-shell-command</code> .	
Package	<code>system</code>	
Signature	<code>pid-exit-status pid &key wait name => exit-status</code>	
Arguments	<code>pid</code>	A process ID.
	<code>wait</code>	A boolean, default value <code>t</code> .
	<code>name</code>	A Lisp object, default value <code>pid</code> .
Values	<code>exit-status</code>	An integer, or <code>nil</code> .
Description	<p>The function <code>pid-exit-status</code> returns the exit status of a process executed by <code>run-shell-command</code> with argument <code>save-exit-status</code> passed a non-<code>nil</code> value.</p> <p>If <code>wait</code> is true then <code>pid-exit-status</code> waits until the process exits, using <code>name</code> in the wait message. If <code>wait</code> is <code>nil</code> and the process has not terminated, then <code>pid-exit-status</code> returns <code>nil</code> immediately.</p>	

Note: `pid-exit-status` is implemented only for Unix/Linux/Mac OS X.

See also `run-shell-command`

pointer-from-address Function

Summary Returns the object into which the given address is pointing.

Package `system`

Signature `pointer-from-address address => object`

Arguments `address` An integer giving the address of the object.

Values `object` The object pointed to by `address`.

Description The function `pointer-from-address` returns the object into which the given integer `address` is pointing. Note that this address may not be pointing into this object after a garbage collection, unless the object is static and is still referenced by another Lisp variable or object.

Example

```
CL-USER 8 > (setq static-string
                      (make-array 3
                                 :element-type 'base-char
                                 :allocation :static))
Warning: Setting unbound variable STATIC-STRING
" )?"
```

```
CL-USER 9 > (sys:object-address static-string)
537166552
```

```
CL-USER 10 > (sys:pointer-from-address *)
" )?"
```

```
CL-USER 11 > (eq * static-string)
T
```

See also

[object-address](#)

		<i>Function</i>
Summary		Prints a Gesture Spec object as a keystroke.
Package		<code>system</code>
Signature		<code>print-pretty-gesture-spec gspec stream &key force-meta-to-alt force-shift-for-upcase => gspec</code>
Arguments	<i>gspec</i>	A Gesture Spec object.
	<i>stream</i>	An output stream.
	<i>force-meta-to-alt</i>	A boolean.
	<i>force-shift-for-upcase</i>	A boolean.
Values	<i>gspec</i>	The Gesture Spec objec that was passed.
Description		<p>The function <code>print-pretty-gesture-spec</code> prints the key-stroke represented by the Gesture Spec object <code>gspec</code> to the stream <code>stream</code>.</p> <p>If <code>force-meta-to-alt</code> is true, then <code>gesture-spec-meta-bit</code> is represented as <code>Alt</code> in the output; otherwise it is represented as <code>Meta</code>. <code>force-meta-to-alt</code> defaults to <code>nil</code>.</p> <p>If <code>force-shift-for-upcase</code> is true and <code>gspec</code> represents uppercase input such as <code>A</code>, then the <code>shift</code> modifier is printed, indicating that <code>shift</code> is pressed to obtain the <code>A</code> character. <code>force-shift-for-upcase</code> defaults to <code>t</code>.</p> <p>If <code>gspec</code> has a wild modifiers or data component (that is, <code>gesture-spec-modifiers</code> and/or <code>gesture-spec-data</code> return <code>nil</code>) then <code><wild></code> appears in the output.</p>

See also [gesture-spec-data](#)
[gesture-spec-meta-bit](#)
[gesture-spec-modifiers](#)
[make-gesture-spec](#)

print-symbols-using-bars Variable

Summary Controls how escaping is done when symbols are printed.

Package system

Initial Value nil

Description The variable `*print-symbols-using-bars*` controls how escaping is done when symbols are printed.

When the value is true, printing symbols that must be escaped (for example, those with names containing the colon character :) is done using the bar character | instead of the backslash character \ in cases when the readable case and ***print-case*** are both :upcase or both :downcase.

```
Example CL-USER 1 > readable-case *readtable*
:UPCASE

CL-USER 2 > (let ((sys:*print-symbols-using-bars* t)
                     (*print-case* :upcase))
              (print (intern "FOO:BAR"))
              (values))

|FOO:BAR|

CL-USER 3 > (let ((sys:*print-symbols-using-bars* t)
                     (*print-case* :downcase))
              (print (intern "FOO:BAR"))
              (values))

foo\:bar
```

product-registry-path	<i>Function</i>
Summary	Gets or sets a registry path for use with your software.
Package	<code>system</code>
Signature	<code>product-registry-path <i>product</i> => <i>path-string</i></code>
Signature	<code>(setf product-registry-path) <i>path</i> <i>product</i> => <i>path</i></code>
Arguments	<p><i>product</i> A Lisp object.</p>
Values	<p><i>path</i> The path as a string or a list of strings.</p> <p><i>path-string</i> The path as a string.</p>
Description	<p>The function <code>product-registry-path</code> returns the registry subpath defined for the product denoted by <i>product</i>, as a string.</p> <p>The function <code>(setf product-registry-path)</code> sets the registry subpath for the product denoted by <i>product</i>.</p> <p>If <i>path</i> is a string it can contain backslash \ or forward slash / as directory separators - these are translated internally to the separator appropriate for the system. Note that any backslash will need escaping (with another backslash) if you input the string value via the Lisp reader.</p> <p>If <i>path</i> is a list of strings, then it is interpreted like the directory component of a pathname.</p> <p>This registry subpath is used when reading and storing user preferences with <code>user-preference</code>.</p> <p>Note that while <i>product</i> can be any Lisp object, values of <i>product</i> are compared by <code>eq</code>, so you should use keywords.</p> <p>Note: to store CAPI window geometries under the registry path for your product, see the entry for <code>capi:top-level-</code></p>

`interface-geometry-key`, in the *LispWorks CAPI Reference Manual*.

Example

```
(setf (sys:product-registry-path :deep-thought)
      (list "Deep Thought" "1.0"))
```

Then, on Unix/Linux/Mac OS X systems:

```
(sys:product-registry-path :deep-thought)
=>
"Deep Thought/1.0"
```

And on Microsoft Windows:

```
(sys:product-registry-path :deep-thought)
=>
"Deep Thought\\1.0"
```

See also

`copy-preferences-from-older-version`
`user-preference`

room-values *Function*

Summary

Returns information about the state of internal storage.

Package

`system`

Signature

`room-values => result`

Values

`result` A plist

```
(:total-size size
            :total-allocated allocated
            :total-free free)
```

Description

`room-values` returns a plist containing information about the state of internal storage. This information is the same as would be printed by (`room`).

Note: In 64-bit LispWorks you can also use `count-gen-num-allocation` and `gen-num-segments-fragmentation-state`.

See also `count-gen-num-allocation`
`room`

		<i>Function</i>
run-shell-command		
Package	<code>system</code>	
Signature		<code>run-shell-command command &key input output error-output separate-streams wait if-input-does-not-exist if-output-exists if-error- output-exists show-window environment element-type save-exit- status => result</code>
Signature		<code>run-shell-command command &key input output error-output separate-streams wait if-input-does-not-exist if-output-exists if-error- output-exists show-window environment element-type save-exit- status => stream, error-stream, process</code>
Arguments	<i>command</i>	A string, a list of strings, a simple-vector of strings, or <code>nil</code> .
	<i>input</i>	<code>nil, :stream</code> or a file designator. Default value <code>nil</code> .
	<i>output</i>	<code>nil, :stream</code> or a file designator. Default value <code>nil</code> .
	<i>error-output</i>	<code>nil, :stream, :output</code> or a file designator. Default value <code>nil</code> .
	<i>separate-streams</i>	A boolean. True value not currently supported.
	<i>wait</i>	A boolean, default value <code>t</code> .
	<i>if-input-does-not-exist</i>	<code>:error, :create</code> or <code>nil</code> . Default value <code>:error</code> .
	<i>if-output-exists</i>	<code>:error, :overwrite, :append, :supersede</code> or <code>nil</code> . Default value <code>:error</code> .
	<i>if-error-output-exists</i>	

		:error, :overwrite, :append, :supersede or nil. Default value :error.
	<i>show-window</i>	A boolean. True value not currently supported.
	<i>environment</i>	An alist of strings naming environment variables and values. Default value nil.
	<i>element-type</i>	Default value base-char.
	<i>save-exit-status</i>	A boolean, default value nil.
Values	<i>result</i>	The exit status of the process running command, or a process ID
	<i>stream</i>	A stream, or nil.
	<i>error-stream</i>	A stream, or nil.
	<i>process</i>	A process ID.
Description		<p>The function <code>run-shell-command</code> allows Unix shell commands to be called from Lisp code with redirection of the std-out, std-in and std-err to Lisp streams. It creates a subprocess which executes the command <i>command</i>.</p> <p>The argument <i>command</i> is interpreted as by <code>call-system</code>. The shell in which the command is run is determined by the environment variable SHELL, or defaults to /bin/csh or /bin/sh if that does not exist</p> <p>If <i>wait</i> is true, then <code>run-shell-command</code> executes <i>command</i> and does not return until the process has exited. In this case none of <i>input</i>, <i>output</i> or <i>error-output</i> may have the value :stream, and the single value <i>result</i> is the exit status of the process that ran <i>command</i>.</p> <p>If <i>wait</i> is nil and none of <i>input</i>, <i>output</i> or <i>error-output</i> have the value :stream then <code>run-shell-command</code> executes <i>command</i> and returns a single value <i>result</i> which is the process ID of the process running <i>command</i>.</p>

If *wait* is `nil` and either of *input* or *output* have the value `:stream` then `run-shell-command` executes *command* and returns three values: *stream* is a Lisp stream which acts as the stdout of the process if *output* is `:stream`, and is the stdin of the process if *input* is `:stream`. *error-stream* is determined by the argument *error-output* as described below. *process* is the process ID of the process.

If *wait* is `nil` and neither of *input* or *output* have the value `:stream` then the first return value, *stream*, is `nil`.

If *wait* is `nil` and *error-output* has the value `:stream` then `run-shell-command` executes *command* and returns three values. *stream* is determined by the arguments *input* and *output* as described above. *error-stream* is a Lisp stream which acts as the stderr of the process. *process* is the process ID of the process.

If *wait* is `nil` and *error-output* is not `:stream` then the second return value, *error-stream*, is `nil`. If *error-output* is `:output`, then stderr goes to the same place as stdout.

If *input* is a pathname or string, then `open` is called with `:if-does-not-exist` *if-input-does-not-exist*. The resulting `file-stream` acts as the stdin of the process.

If *output* is a pathname or string, then `open` is called with `:if-exists` *if-output-exists*. The resulting `file-stream` acts as the stdout of the process.

If *error-output* is a pathname or string, then `open` is called with `:if-exists` *if-error-output-exists*. The resulting `file-stream` acts as the stderr of the process.

This table describes the streams created, for each combination of stream arguments:

Table 16.1 The streams created by `run-shell-command`

Arguments	<i>stream</i>	<i>error-stream</i>
<i>input is :stream</i> <i>output is :stream</i> <i>error-output is :stream</i>	An I/O stream connected to stdin and stdout	An input stream connected to stderr
<i>input is not :stream</i> <i>output is :stream</i> <i>error-output is :stream</i>	An input stream connected to stdout	An input stream connected to stderr
<i>input is :stream</i> <i>output is not :stream</i> <i>error-output is :stream</i>	An output stream connected to stdin	An input stream connected to stderr
<i>input is not :stream</i> <i>output is not :stream</i> <i>error-output is :stream</i>	<code>nil</code>	An input stream connected to stderr
<i>input is :stream</i> <i>output is :stream</i> <i>error-output is :output</i>	An I/O stream connected to stdin, stdout and stderr	<code>nil</code>
<i>input is not :stream</i> <i>output is :stream</i> <i>error-output is :output</i>	An input stream connected to stdout and stderr	<code>nil</code>
<i>input is :stream</i> <i>output is not :stream</i> <i>error-output is :output</i>	An output stream connected to stdin	<code>nil</code>
<i>input is not :stream</i> <i>output is not :stream</i> <i>error-output is :output</i>	<code>nil</code>	<code>nil</code>
<i>input is :stream</i> <i>output is :stream</i> <i>error-output is not :stream or :output</i>	An I/O stream connected to stdin and stdout	<code>nil</code>
<i>input is not :stream</i> <i>output is :stream</i> <i>error-output is not :stream or :output</i>	An input stream connected to stdout	<code>nil</code>

Table 16.1 The streams created by `run-shell-command`

Arguments	<i>stream</i>	<i>error-stream</i>
<i>input</i> is <code>:stream</code> <i>output</i> is not <code>:stream</code> <i>error-output</i> is not <code>:stream</code> or <code>:output</code>	An output stream connected to stdin	<code>nil</code>
<i>input</i> is not <code>:stream</code> <i>output</i> is not <code>:stream</code> <i>error-output</i> is not <code>:stream</code> or <code>:output</code>	<code>nil</code>	<code>nil</code>

If any of *input*, *output* or *error-output* are streams, then they must be `file-streams` or `socket-streams` capable of acting as the stdin, stdout or stderr of the process.

environment should be an alist of strings naming environment variables and their values. The process runs in an environment inherited from the Lisp process, augmented by *environment*.

If *save-exit-status* is true, then the system stores the exit status of the process, so that it can be recovered by calling `pid-exit-status`.

Note: `run-shell-command` is implemented only for Unix/Linux/Mac OS X.

Example

```
(multiple-value-bind (out err pid)
  (sys:run-shell-command "sh -c 'echo foo >&2; echo
bar'"'
    :wait nil
    :output :stream
    :error-output :stream)
  (with-open-stream (out out)
    (with-open-stream (err err)
      (values (read-line out) (read-line err)))))

=>
"bar", "foo"
```

See also **call-system**
call-system-showing-output
open-pipe

safe-locale-file-encoding *Function*

Summary Provides a safe encoding which corresponds to the current code page on Microsoft Windows, and the locale on Unix.

Package **system**

Signature **safe-locale-file-encoding pathname ef-spec buffer length => new-ef-spec**

Description The function **safe-locale-file-encoding** is similar to **locale-file-encoding** except that it always returns a safe external format. That is, the external format does not signal error on writing characters not in the encoding.
On Microsoft Windows, **safe-locale-file-encoding** consults the ANSI code page. If the code page identifier *id* is in **win32:latin-1-code-pages***, it merges *ef-spec* with **:latin-1-safe**. This external format writes Latin-1 on output, using 63 (ASCII '?') to replace any non-Latin-1 characters that are written. If the code page identifier *id* is not in **win32:latin-1-code-pages*** then **safe-locale-file-encoding** merges *ef-spec* with an encoding corresponding to the current code page that uses the code page's replacement code for characters that cannot be encoded.
safe-locale-file-encoding merges *ef-spec* with **:latin-1-safe** on Unix.

See also ***file-encoding-detection-algorithm***
latin-1-code-pages
locale-file-encoding

	<i>Function</i>
Summary	Sets a function or functions to call after an automatic GC in 64-bit LispWorks.
Package	<code>system</code>
Signature	<code>set-automatic-gc-callback <i>blocking-gen-num-func</i> &optional <i>other-func</i> => <i>other-func</i></code>
Arguments	<p><i>blocking-gen-num-func</i> A function designator for a function of two arguments, or <code>nil</code>.</p> <p><i>other-func</i> A function designator for a function of one argument, or <code>nil</code>.</p>
Values	<i>other-func</i> A function designator for a function of one argument, or <code>nil</code> .
Description	<p>The function <code>set-automatic-gc-callback</code> sets a function or functions to call after an automatic garbage collection (GC).</p> <p>If <i>blocking-gen-num-func</i> is a function designator it should take two arguments: the generation number and, if <i>do-gc</i> in the last call to <code>set-blocking-gen-num</code> was a number, the number of copied segments. It is called whenever the blocking generation is GCed automatically. If <i>blocking-gen-num-func</i> is <code>nil</code>, then this callback is switched off.</p> <p>If <i>other-func</i> is a function designator it should take one argument, the generation number that was GCed. It is called whenever an automatic GC occurred and <i>blocking-gen-num-func</i> was not called, either because the blocking generation was not GCed, or because <i>blocking-gen-num-func</i> was passed as <code>nil</code>. If <i>other-func</i> is <code>nil</code> (the default) then this callback is switched off.</p> <p>The calls occur after the GC has finished and there is no restriction on what they can do. If the call ends up allocating</p>

enough to trigger another automatic GC, they enter again recursively.

Note: this function is implemented only in 64-bit LispWorks. It is not relevant to the Memory Management API in 32-bit implementations.

See also **set-blocking-gen-num**

set-blocking-gen-num *Function*

Summary Sets the blocking generation in 64-bit LispWorks.

Package **system**

Signature **set-blocking-gen-num** *gen-num* &**key** *do-gc* *max-size* *gc-threshold*
 \Rightarrow *old-blocking-gen-num*, *do-gc*, *max-size*, *old-gc-threshold*

Arguments *gen-num* An integer between 0 and 7, inclusive.

do-gc One of **t**, **nil** and **:mark**, or a real number between 0 and 10, inclusive.

max-size A positive real number, or **nil**.

gc-threshold An integer greater than 12800, or a real in the inclusive range [0 100], or **nil**.

Values *old-blocking-gen-num*

 An integer between 0 and 7, inclusive.

do-gc One of **t**, **nil** and **:mark**, or a real number between 0 and 10, inclusive.

max-size A positive real number.

old-gc-threshold A number.

Description The function `set-blocking-gen-num` sets *gen-num* as the generation that blocks. That is, no object is automatically promoted out of generation *gen-num* to a higher generation.

If *do-gc* is non-`nil`, then generation *gen-num* is automatically collected when needed, as defined by *gc-threshold* (see `set-gen-num-gc-threshold`).

The actual value of *do-gc* specifies how to GC the blocking generation when required. The possible values of *do-gc* are interpreted as follows:

`t` Use Copying GC.

`:mark` Use Marking GC.

A number in the inclusive range [0, 10]

Use Marking GC with copying of fragmented segments. The value specifies the *fragmentation-threshold* (the same as the argument to `marking-gc`). This is the ratio between the amount of free space that cannot be easily used and the amount of allocated space inside a segment. Only segments with fragmentation higher than the threshold are copied.

The default value of *do-gc* is `t`.

max-size is meaningful only if *do-gc* is a number. It specifies the maximum size in Gigabytes to try to copy. If the fragmented segments contain more data than this value, only some of them are copied in each GC.

If *gc-threshold* is non-`nil`, it is used to set the threshold for automatic GC using `set-gen-num-gc-threshold`.

The initial setup is as if this call has been made:

```
(sys:set-blocking-gen-num 3)
```

That is, the system will GC automatically according to the default *gc-threshold* using Copying GC.

Setting the blocking generation *gen-num* to a lower number is useful into two situations:

1. When you have an operation that allocates a significant amount of data, and almost of it goes when the operation finishes, it is useful to reduce the blocking *gen-num* during the operation. The macro `block-promotion` is a convenient way of doing that.
2. If you have a good idea of how your application behaves, it may be useful to block at a lower generation (2 or 1), and then periodically call `gc-generation` explicitly to promote long living objects to a higher generation. The advantage of doing this is that you can call `gc-generation` in places where you know there are not many short-lived objects alive.

Passing a *do-gc* value other than `t` is useful when the blocking generation can be large enough that copying it all may cause very serious paging. Passing `do-gc :mark` will stop the system from copying the blocking generation, but may cause fragmentation if a significant number of long-lived objects die after a while, and there are not explicit calls to `gc-generation` or `marking-gc`.

`set-blocking-gen-num` returns four values: the old blocking generation number, the old value of *do-gc*, the *max-size*, and the old value of *gc-threshold*. It can be called with `gen-num nil` to query the values without changing any of them.

Note: this function is implemented only in 64-bit LispWorks. It is not relevant to the Memory Management API in 32-bit implementations.

See also

`block-promotion`
`gc-generation`
`marking-gc`
`set-automatic-gc-callback`
`set-gen-num-gc-threshold`

set-default-segment-size *Function*

Summary Sets the default initial size of a segment in 64-bit LispWorks.

Package `system`

Signature `set-default-segment-size gen-num allocation-type size-in-mb
=> segment-size`

Arguments `gen-num` An integer between 0 and 3, inclusive.

`allocation-type` One of :`cons`, :`symbol`, :`function`, :`non-pointer`, :`other`, :`mixed`, :`cons-static`, :`non-pointer-static`, :`mixed-static`, :`weak`, :`other-big`, and :`non-pointer-big`.

`size-in-mb` A number, or `nil`.

Values `segment-size` A number.

Description The function `set-default-segment-size` sets the default initial size of a segment for a specific generation and allocation type.

The default initial size is also used as the default size for enlargement of the segment.

`allocation-type` can be any of the allocation types. However, if `allocation-type` is :`other-big` or :`non-pointer-big`, this function has no effect.

If `size-in-mb` is a number, it specifies the size in megabytes. If `size-in-mb` is `nil` then `set-default-segment-size` returns the default initial segment size without altering it.

The returned value, `segment-size`, is the previous default initial segment size.

During automatic garbage collections (GCs) the system collects an ephemeral generation when any of its segments for the main allocation types is full. Thus the size of the segments defines the frequency of GCs in these generations.

Note: this function is implemented only in 64-bit LispWorks.
 It is not relevant to the Memory Management API in 32-bit implementations, where `enlarge-generation` is available.

See also `avoid-gc`
`enlarge-generation`
`set-maximum-segment-size`

set-delay-promotion *Function*

Summary	Delays promotion for a specified generation in 64-bit LispWorks.	
Package	<code>system</code>	
Signature	<code>set-delay-promotion gen-num on => on</code>	
Arguments	<code>gen-num</code>	An integer between 0 and 7, inclusive.
	<code>on</code>	A generalized boolean.
Values	<code>on</code>	A generalized boolean.
Description	<p>The function <code>set-delay-promotion</code> delays promotion for generation <code>gen-num</code>, which means that objects are promoted to the next generation in the second garbage collection (GC) that they survive in generation <code>gen-num</code>. By default, objects are promoted in the first GC.</p> <p>It is not obvious under what circumstances delayed promotion is more useful than the default behavior. If you find this function useful, please let us know at Lisp Support.</p>	
	<p>Note: this function is implemented only in 64-bit LispWorks. It is not relevant to the Memory Management API in 32-bit implementations.</p>	
See also	<code>set-blocking-gen-num</code>	

	<i>Function</i>								
Summary	Sets the modification and access times of a file.								
Package	<code>system</code>								
Signature	<code>set-file-dates file &key creation modification access</code>								
Arguments	<table border="0"> <tr> <td><i>file</i></td><td>A pathname designator.</td></tr> <tr> <td><i>creation</i></td><td>A non-negative integer, or <code>nil</code>.</td></tr> <tr> <td><i>modification</i></td><td>A non-negative integer, or <code>nil</code>.</td></tr> <tr> <td><i>access</i></td><td>A non-negative integer, or <code>nil</code>.</td></tr> </table>	<i>file</i>	A pathname designator.	<i>creation</i>	A non-negative integer, or <code>nil</code> .	<i>modification</i>	A non-negative integer, or <code>nil</code> .	<i>access</i>	A non-negative integer, or <code>nil</code> .
<i>file</i>	A pathname designator.								
<i>creation</i>	A non-negative integer, or <code>nil</code> .								
<i>modification</i>	A non-negative integer, or <code>nil</code> .								
<i>access</i>	A non-negative integer, or <code>nil</code> .								
Description	<p>The function <code>set-file-dates</code> sets the modification and access times of the file <i>file</i> for each of modification and access that is non-<code>nil</code>.</p> <p>On Microsoft Windows, if <i>creation</i> is non-<code>nil</code>, the creation time of the file is also set. <i>creation</i> is ignored on other platforms.</p> <p>Each keyword argument is interpreted as a universal time representing the time to set, unless it is <code>nil</code> in which case the corresponding time for <i>file</i> is not changed. Each keyword argument has default value <code>nil</code>.</p> <p>An error of type <code>file-error</code> is signalled on failure.</p>								
See also	<code>open</code>								

	<i>Function</i>
Summary	Sets the additional allocation threshold that triggers a GC in the blocking generation in 64-bit LispWorks.
Package	<code>system</code>

Signature	<code>set-gen-num-gc-threshold</code> <i>gen-num threshold => old-threshold</i>	
Arguments	<i>gen-num</i>	An integer between 0 and 7, inclusive.
	<i>threshold</i>	An integer greater than 12800, or a real in the inclusive range [0 100], or <code>nil</code> .
Values	<i>old-threshold</i>	A number.
Description	<p>The function <code>set-gen-num-gc-threshold</code> sets the threshold for additional allocation that triggers a garbage collection (GC) in generation <i>gen-num</i> when this is the blocking generation (as set by <code>set-blocking-gen-num</code>). A GC is triggered when the allocation in generation <i>gen-num</i> grows more than <i>threshold</i> over the allocation after the last GC of this generation (or a GC of a higher generation).</p> <p>To set the threshold, <i>threshold</i> can be an integer greater than 12800, which is interpreted as the absolute value. Alternatively <i>threshold</i> can be a real number in the inclusive range [0 100], which is multiplied by the allocation since the previous GC to get the actual threshold to set.</p> <p>The default threshold for all generations is 1. That is, for all generations <i>gen-num</i>, when generation <i>gen-num</i> is the blocking generation and allocation in it has doubled since the previous GC, generation <i>gen-num</i> is collected automatically.</p> <p><code>set-gen-num-gc-threshold</code> can be called when the generation <i>gen-num</i> is not the blocking generation, and will set the value for that <i>gen-num</i>. Such a call will not take effect until the generation <i>gen-num</i> becomes the blocking generation, as set by a call to <code>set-blocking-gen-num</code> (with <code>:do-gc non-nil</code>).</p> <p>Increasing the threshold reduces the number of GC calls, but may increase the virtual memory usage.</p>	

`set-gen-num-gc-threshold` returns the old threshold for the generation *gen-num*. It can be called with `threshold nil` to return the threshold value without changing it.

Note: this function is implemented only in 64-bit LispWorks. It is not relevant to the Memory Management API in 32-bit implementations.

See also

`set-blocking-gen-num`

set-maximum-memory

Function

Summary Sets or removes a limit for the top of the Lisp heap in 32-bit LispWorks.

Package `system`

Signature `set-maximum-memory address`

Arguments `address` An integer address, or `nil`.

Description `set-maximum-memory` sets or removes a limit for the maximum address that the Lisp heap can grow to. If `address` is an integer, this becomes the maximum address. If `address` is `nil`, any limit set by `set-maximum-memory` is removed.

In 32-bit implementations on platforms other than Linux and Macintosh, by default the maximum memory is not set. LispWorks (32-bit) for Linux and LispWorks (32-bit) for Macintosh both set the maximum memory on startup. In all cases the system is constrained by the size of the physical memory.

When the maximum memory is reached (either that set by `set-maximum-memory` or the physical memory limit) the system will become unstable. Therefore this situation should be avoided. The benefit of having the maximum memory set is that a useful error is signaled if the limit is reached.

An application which is likely to grow to the maximum memory should test the amount of available memory using `memory-growth-margin` or `room-values` at suitable times, and take action to reclaim memory. Do not rely on handling the error signaled when the maximum memory is reached, since the system is already unstable at this point.

Note: `set-maximum-memory` is implemented only in 32-bit LispWorks. It is not relevant to the Memory Management API in 64-bit implementations.

See also	<code>check-fragmentation</code> <code>mark-and-sweep</code> <code>memory-growth-margin</code> <code>room-values</code>
----------	--

	<i>Function</i>						
set-maximum-segment-size							
Summary	Defines the maximum segment size for a generation and allocation type in 64-bit LispWorks.						
Package	<code>system</code>						
Signature	<code>set-maximum-segment-size gen-num allocation-type size-in-mb</code>						
Arguments	<table> <tr> <td><i>gen-num</i></td><td>An integer between 0 and 7, inclusive.</td></tr> <tr> <td><i>allocation-type</i></td><td>One of :cons, :symbol, :function, :non-pointer, :other, :mixed, :cons-static, :non-pointer-static, :mixed-static, :weak, :other-big, and :non-pointer-big.</td></tr> <tr> <td><i>size-in-mb</i></td><td>An integer between 1 and 256 inclusive, or <code>nil</code>.</td></tr> </table>	<i>gen-num</i>	An integer between 0 and 7, inclusive.	<i>allocation-type</i>	One of :cons, :symbol, :function, :non-pointer, :other, :mixed, :cons-static, :non-pointer-static, :mixed-static, :weak, :other-big, and :non-pointer-big.	<i>size-in-mb</i>	An integer between 1 and 256 inclusive, or <code>nil</code> .
<i>gen-num</i>	An integer between 0 and 7, inclusive.						
<i>allocation-type</i>	One of :cons, :symbol, :function, :non-pointer, :other, :mixed, :cons-static, :non-pointer-static, :mixed-static, :weak, :other-big, and :non-pointer-big.						
<i>size-in-mb</i>	An integer between 1 and 256 inclusive, or <code>nil</code> .						
Values	<code>max-segment-size</code> A number.						

Description	<p>The function <code>set-maximum-segment-size</code> sets the maximum segment size for a generation and allocation type in 64-bit LispWorks.</p> <p><i>allocation-type</i> can be any of the allocation types. However, if <i>allocation-type</i> is <code>:other-big</code> or <code>:non-pointer-big</code>, this function has no effect.</p> <p><i>size-in-mb</i> is the size in megabytes.</p> <p>For the non-ephemeral generations (that is, the blocking generation and above), if the system needs more memory of some allocation type in some generation, its normal operation is to enlarge one of the existing segments in this generation of this allocation type. If it does not find a segment that it can enlarge, it allocates a new segment of the same allocation type in the same generation. Therefore the maximum segment size affects the number of segments that will be used.</p> <p>There is an overhead to using more segments, so normally having the largest segment size which the implementation allows (256MB) is the best. Reducing the size may be useful when using <code>marking-gc</code> with <i>what-to-copy</i> <code>non-nil</code> or <code>set-blocking-gen-num</code> with <i>do-gc</i> a number to prevent fragmentation in the blocking generation. In this situation, reducing the size of each segment makes it easier for the system to find segments to copy, even if the <i>max-size</i> parameter is set to a low number to avoid using too much virtual memory.</p> <p>The returned value, <i>max-segment-size</i>, is the previous maximum segment size.</p> <p>If <i>size-in-mb</i> is a number, it specifies the size in megabytes. If <i>size-in-mb</i> is <code>nil</code> then <code>set-maximum-segment-size</code> returns the maximum segment size without altering it.</p> <p>Note: this function is implemented only in 64-bit LispWorks. It is not relevant to the Memory Management API in 32-bit implementations.</p>
-------------	--

See also [marking-gc](#)
[set-blocking-gen-num](#)
[set-default-segment-size](#)

set-memory-check		Function
Summary	Sets a memory check in 64-bit LispWorks.	
Package	<code>system</code>	
Signature	<code>set-memory-check size function</code>	
Arguments	<code>size</code>	An integer.
	<code>function</code>	A function designator.
Description	<p>The function <code>set-memory-check</code> sets a memory check. <code>size</code> must be an integer. It specifies the total size in bytes of the mapped areas of Lisp at which the check is triggered. <code>function</code> is a function of no arguments.</p> <p>After each automatic garbage collection (GC) the system checks whether the mapped area (excluding stacks) is larger than <code>size</code>. If it is larger, <code>function</code> is called with no arguments.</p> <p>Inside the dynamic scope of the call, the check is disabled. There are no restrictions or special considerations on what the function <code>function</code> does.</p> <p>The current mapped area can be found by the <code>:total-size</code> value returned by <code>room-values</code>.</p> <p>Note: this function is implemented only in 64-bit LispWorks. It is not relevant to the Memory Management API in 32-bit implementations.</p>	

set-memory-exhausted-callback *Function*

Summary Sets a callback that is called when memory is exhausted in 64-bit LispWorks.

Package `system`

Signature `set-memory-exhausted-callback function &optional where => callbacks`

Arguments `function` A function designator, the keyword `:reset`, or `nil`.

`where` `:first`, `:last` or `nil`.

Values `callbacks` A list of function designators.

Description The function `set-memory-exhausted-callback` adds a callback that is called when memory is exhausted. That is, when the system fails to map memory.

Note: `set-memory-check` is a more robust way to protect against memory exhaustion problems.

If `function` is a function designator then it should be a function with signature

`function gen-num size type-name static`

`function` is expected to report what the system was trying to allocate when it failed to map memory. Its arguments are:

`gen-num` The number of the generation in which it was trying to allocate.

`size` The size in bytes which it was trying to allocate.

`type-name` A string naming the allocation type it was trying to allocate.

`static` A boolean, true if it was trying to allocate a static object, and false otherwise.

function can also have the special value `:reset`, which resets the callback list to `nil`.

function can also be `nil`, which means do nothing but simply return the current list of callbacks.

where defines the position in the list that the callback *function* is placed. Its allowed values are:

`:first` *function* is placed first in the callbacks list.

`:last` *function* is placed last in the callbacks list.

`nil` *function* is removed from the callbacks list.

`set-memory-exhausted-callback` always first removes *function* from the callbacks list, and then adds it according to *where*. The default value of *where* is `:first`. Functions in the list are compared with `equalp`.

`set-memory-exhausted-callback` returns the callback list.

When a callback is called, Lisp already failed to map memory. This means that you must not rely on the callback to do real work. It should therefore attempt only a minimal amount of work such as clean-ups and generating debug information. It should not try to do real work.

After all the callbacks are called, the system signals an error of type `storage-exhausted`. The condition can be accessed using the accessors described for `storage-exhausted`.

Note: this function is implemented only in 64-bit LispWorks. It is not relevant to the Memory Management API in 32-bit implementations.

See also

`set-memory-check`
`storage-exhausted`

set-signal-handler

Function

Summary

Installs or removes a handler for a Unix signal.

Note: applicable only on UNIX/Linux/Mac OS X.

Package `system`

Signature `set-signal-handler signum handler`

Arguments `signum` A Unix signal number.
 `handler` A function or `nil`.

Description `set-signal-handler` with a function `handler` configures Lisp-Works such that `handler` is called when the Unix signal `signum` occurs.

If `handler` is `nil`, any handler for `signum` is removed.

`handler` should be defined to take an `&rest` argument, and ignore it. There are no restrictions on `handler` other than those applying to any asynchronous function call, and that it may be called in any thread. In particular there is no need to handle the signal immediately.

The configuration established by `set-signal-handler` is not persistent over image saving (or application delivery), so it should be called each time the image (or application) is started.

Note: the currently defined signal handlers are shown in the output of the bug report template which can generated via the `:bug-form` listener command. For example, there is a `SIGINT` handler which calls `break`. You should consult Lisp Support before overwriting existing signal handlers.

Note: LispWorks initially has no `SIGHUP` handler. `SIGHUP` will kill a LispWorks process which does not have a `SIGHUP` handler installed. When the Common LispWorks GUI starts up, a `SIGHUP` handler (which attempts to release locks in the environment) is installed. However if you need a `SIGHUP` handler in a server application, for example, you should install one using `set-signal-handler`.

Example

```
(defun my-hup-handler (&rest x)
  (declare (ignorable x))
  (cerror "Continue"
    "Got a HUP signal"))

(sys:set-signal-handler 1 'my-hup-handler)
```

Note that the Common LispWorks GUI overwrites a `SIGHUP` handler, so you would need to reinstall it after GUI startup.

set-spare-keeping-policy*Function*

Summary Controls the behavior of the system when a segment is emptied in 64-bit LispWorks.

Package `system`

Signature `set-spare-keeping-policy gen-num policy => old-policy`

Arguments `gen-num` An integer in the inclusive range [0,7].
 `policy` A generalized boolean.

Values `old-policy` A generalized boolean.

Description The function `set-spare-keeping-policy` controls the behavior of the system when a segment is emptied in 64-bit LispWorks.

If `policy` is non-`#nil`, then when a segment in generation `gen-num` is emptied by copying all the objects out from it, it may be kept as a spare segment to be used in the future. This increases the use of virtual memory, but reduces the number of calls to `mmap` and `munmap`. It may be useful in applications that allocate at a very high rate.

If timing an application reveals a lot (more than 5%) of time in the "System Time", and especially if this shows up in the GC times produced by `extended-time`, it may be useful to set

the policy to `non-nil` in generation 1, 2 and maybe in generation 3.

The default policy is `nil` for all generations, meaning that empty segments are discarded.

The returned value *old-policy* is the previous policy for the generation *gen-num*.

Note: this function is implemented only in 64-bit LispWorks. It is not relevant to the Memory Management API in 32-bit implementations.

See also [extended-time](#)

sg-default-size *Variable*

Summary Default initial size of a stack group.

Package [system](#)

Initial Value In LispWorks (64-bit) for Solaris:

`20000`

In all other implementations:

`16000`

Description The value of the variable `*sg-default-size*` is the initial size of a stack group, in 32 bit words (in 32-bit implementations) or in 64 bit words (in 64-bit implementations).

`*sg-default-size*` can be bound around a call to a process creation function. Note that setting the global value of this variable affects the size of all system processes too, so this is not recommended.

Example To create a process with a stack of 32000 words:

```
(let ((sys:*sg-default-size* 32000))
  (mp:process-run-function "Larger stack" '()
    #'(lambda ()
        (print (hcl:current-stack-length)))))
```

See also `current-stack-length`
`*stack-overflow-behaviour*`

simple-augmented-string *Type*

Summary	The simple augmented string type.	
Package	<code>system</code>	
Signature	<code>simple-augmented-string length</code>	
Arguments	<code>length</code>	The length of the string (or <code>*</code> , meaning any).
Description	This is the simple version of <code>augmented-string</code> , that is, the string itself is simple. Equivalent to: <code>(simple-vector character length)</code>	
See also	<code>augmented-string</code>	

simple-augmented-string-p *Function*

Summary	Tests if an object is a simple augmented string.	
Package	<code>system</code>	
Signature	<code>simple-augmented-string-p object => bool</code>	
Arguments	<code>object</code>	The object to be tested.

Values	<code>bool</code>	<code>t</code> if <i>object</i> is a simple augmented string; <code>nil</code> otherwise.
Description		This is the predicate for simple augmented strings.
See also		<code>simple-augmented-string</code>

simple-int32-vector *Type*

Summary	A type for simple vectors of <code>int32</code> objects.
Package	<code>system</code>
Signature	<code>simple-int32-vector</code>
Description	The type <code>simple-int32-vector</code> provides simple vectors of <code>int32</code> objects and can be used to generate optimal 32-bit arithmetic code. Create a <code>simple-int32-vector</code> by calling <code>make-simple-int32-vector</code> . See the section "Fast 32-bit arithmetic" in the <i>LispWorks User Guide</i> for more information.
See also	<code>int32</code> <code>int32-aref</code> <code>make-simple-int32-vector</code>

stack-overflow-behaviour *Variable*

Summary	Controls behavior when stack overflow occurs.
Package	<code>system</code>
Initial Value	<code>:error</code>

Description	<p>The variable <code>*stack-overflow-behaviour*</code> controls behavior when stack overflow occurs.</p> <p>When <code>*stack-overflow-behaviour*</code> is set to <code>:error</code>, LispWorks signals an error.</p> <p>When it is set to <code>:warn</code>, LispWorks increases the stack size automatically to accommodate the overflow, but prints a warning message to signal that this has happened.</p> <p>When it is set to <code>nil</code>, LispWorks increases stack size silently.</p>
Compatibility Note	In LispWorks 4.4 and previous on Windows and Linux platforms, automatic stack extension is not implemented. This has been fixed in LispWorks 5.0 and later.
See also	<code>*sg-default-size*</code>

	<i>Function</i>
staticp	
Summary	Specifies whether a given object has been allocated in static memory.
Package	<code>system</code>
Signature	<code>staticp obj => bool</code>
Arguments	<i>obj</i> An object.
Values	<i>bool</i> <code>t</code> if the object is allocated in static memory; <code>nil</code> otherwise.
Description	<p>This predicate can be used on an object to find out whether it is allocated in static memory.</p> <p>Foreign instantiations made by Lisp — for example in a Foreign Language Interface program — are made in static memory. The Lisp representations of these alien objects are <i>not</i>,</p>

however. Therefore `staticcp` applied to an alien returns `nil` even though the alien instance itself is really allocated in static memory. To establish this, you can check the pointer to the alien instance within its Lisp representation (a structure).

storage-exhausted

Class

Summary	A condition class for failures to map memory.	
Superclasses	<code>storage-condition</code>	
Initargs	<code>:gen-num</code>	The number of the generation in which the system was trying to allocate.
	<code>:size</code>	The size in bytes which the system was trying to allocate.
	<code>:type</code>	A string naming the allocation type the system was trying to allocate.
	<code>:static</code>	A boolean, true if the system was trying to allocate a static object, and false otherwise.
Accessors	<code>storage-exhausted-gen-num</code> <code>storage-exhausted-size</code> <code>storage-exhausted-static</code> <code>storage-exhausted-type</code>	
Description	The class <code>storage-condition</code> is a condition class used for reporting failures to map memory. Allocation types are as described in <code>set-maximum-segment-size</code> .	
See also	<code>set-memory-exhausted-callback</code>	

	<i>Function</i>				
sweep-gen-num-objects					
Summary	Applies a function to all the live objects in a generation in 64-bit LispWorks.				
Package	<code>system</code>				
Signature	<code>sweep-gen-num-objects gen-num function</code>				
Arguments	<table border="0"> <tr> <td><i>gen-num</i></td><td>An integer in the inclusive range [0,7].</td></tr> <tr> <td><i>function</i></td><td>A designator for a function of one argument, the object.</td></tr> </table>	<i>gen-num</i>	An integer in the inclusive range [0,7].	<i>function</i>	A designator for a function of one argument, the object.
<i>gen-num</i>	An integer in the inclusive range [0,7].				
<i>function</i>	A designator for a function of one argument, the object.				
Values	<code>sweep-gen-num-objects</code> returns <code>nil</code> .				
Description	<p>The function <code>sweep-gen-num-objects</code> applies <i>function</i> to all the live objects in the generation <i>gen-num</i>.</p> <p><i>function</i> should take one argument, the object. It can allocate, but if it allocates heavily the sweeping becomes unreliable. Small amounts of allocation will normally happen only in generation 0, and so will not affect sweeping of other generations.</p> <p>Note: <code>sweep-gen-num-objects</code> is not implemented in 32-bit LispWorks, where you can use <code>sweep-all-objects</code> instead.</p>				
See also	<code>sweep-all-objects</code>				

	<i>Function</i>
typed-aref	
Summary	Accesses a typed aref vector efficiently.
Package	<code>system</code>
Signature	<code>typed-aref type vector byte-index => value</code> <code>(setf typed-aref) value type vector byte-index => value</code>

Arguments	<i>type</i>	A type specifier.
	<i>vector</i>	A vector created by <code>make-typed-aref-vector</code> .
	<i>byte-index</i>	A non-negative fixnum.
Values	<i>value</i>	An object of type <i>type</i> .
Description		<p>The function <code>typed-aref</code> allows efficient access to a typed aref vector.</p> <p><i>type</i> must evaluate to one of: <code>double-float</code>, <code>float</code>, <code>single-float</code>, <code>(unsigned-byte 32)</code>, <code>(signed-byte 32)</code>, <code>(unsigned-byte 16)</code>, <code>(signed-byte 16)</code>, <code>(unsigned-byte 8)</code> or <code>(signed-byte 8)</code>.</p> <p><i>vector</i> must be an object returned by <code>make-typed-aref-vector</code>.</p> <p><i>byte-index</i> specifies the index in bytes from the start of the data in the vector. It must be a non-negative fixnum which is less than the <i>byte-length</i> argument passed to <code>make-typed-aref-vector</code>.</p> <p><code>typed-aref</code> and <code>(setf typed-aref)</code> will be inlined to code which is as efficient as possible when compiled with <code>(optimize (safety 0))</code> and a constant type. As usual, you need to add <code>(optimize (float 0))</code> to remove boxing for the float types.</p> <p>Note: Efficient access to foreign arrays is also available. See <code>fli:foreign-typed-aref</code> in the <i>LispWorks Foreign Language Interface User Guide and Reference Manual</i></p>
Example		<pre>(defun double-float-typed-aref-incf (x y z) (declare (optimize (float 0) (safety 0))) (incf (sys:typed-aref 'double-float x y) (the double-float z)) x)</pre>
See also		<code>make-typed-aref-vector</code>

	<i>Function</i>
wait-for-input-streams	
Summary	Waits for input on a list of socket streams, returning those that are ready.
Package	<code>system</code>
Signature	<code>wait-for-input-streams streams &key wait-function wait-reason timeout => result</code>
Arguments	<p><i>streams</i> A list, each member of which is a <code>socket-stream</code>.</p> <p><i>wait-function</i> A function of no arguments.</p> <p><i>wait-reason</i> A string.</p> <p><i>timeout</i> A real number or <code>nil</code>.</p>
Values	<i>result</i> A list of <code>socket-streams</code> or <code>nil</code> .
Description	<p>The function <code>wait-for-input-streams</code> waits for any of the streams in the argument <code>streams</code> to be ready for input. "Ready for input" typically means that some input is available from the stream, but can also mean that the peer closed the connection or there is an attempt to connect to the socket. Note that this function first checks the buffer for buffered streams.</p> <p>When any of the streams is ready for input, <code>wait-for-input-streams</code> returns a list of all the streams that are ready, in the same order that they appear in <code>streams</code>.</p> <p>If <code>timeout</code> is non-<code>nil</code> it must be a real number, specifying a timeout in seconds. If <code>timeout</code> seconds pass and none of the streams is ready, <code>wait-for-input-streams</code> returns <code>nil</code>.</p> <p>If <code>timeout</code> is 0, <code>wait-for-input-streams</code> returns all of the streams that are ready immediately, without waiting at all. That is, it behaves like <code>listen</code> on many streams.</p> <p>If <code>wait-function</code> is supplied, it is called periodically with no arguments, and if it returns non-<code>nil</code> then <code>wait-for-input-</code></p>

`streams` returns `nil`. Note that, like the *wait-function* of `process-wait`, *wait-function* is called often and on other threads, so need to be an inexpensive call and independent of dynamic context.

If *wait-reason* is supplied it is used as the *wait-reason* for the Lisp process that calls `wait-for-input-streams` while it is waiting.

Note: `wait-for-input-streams` may return the list *streams* that was passed to it as is, if all the streams are ready.

See also [wait-for-input-streams-returning-first](#)

wait-for-input-streams-returning-first *Function*

Summary Waits for input on a list of socket streams, returning the first stream that is ready.

Package `system`

Signature `wait-for-input-streams-returning-first streams &key wait-function wait-reason timeout => result`

Arguments `streams` A list, each member of which is a `socket-stream`.

`wait-function` A function of no arguments.

`wait-reason` A string.

`timeout` A real number or `nil`.

Values `result` A `socket-stream` or `nil`.

Description The function `wait-for-input-streams-returning-first` behaves just like `wait-for-input-streams` except that it returns the first stream in the list *streams* that is ready for input.

See also [**wait-for-input-streams**](#)

17

The WIN32 Package (including DDE)

This chapter describes the symbols available in the `WIN32` package, including reference entries for accessing the Windows registry, and Dynamic Data Exchange (DDE).

The functions are listed in four sections: miscellaneous `WIN32` symbols, the registry API, the DDE client interface, and the DDE server interface. You should use the DDE sections in conjunction with the relevant chapter in the *LispWorks User Guide*.

Note: this chapter applies only to LispWorks for Windows, and not the UNIX, Linux, FreeBSD or Mac OS X platforms.

Note: the `WIN32` package is not a supported implementation of the Win32 API. Define your own interfaces to Windows functions as you need - see the *LispWorks Foreign Language Interface User Guide and Reference Manual* for details.

17.1 Miscellaneous WIN32 symbols

Function
<code>dismiss-splash-screen</code>

Summary Makes a startup screen disappear.

Package	<code>win32</code>
Signature	<code>dismiss-splash-screen &optional forcep</code>
Arguments	<code>forcep</code> A generalized boolean.
Description	The function <code>dismiss-splash-screen</code> makes a startup screen (as specified via the <code>:startup-bitmap-file</code> delivery keyword) disappear. If <code>forcep</code> is <code>nil</code> then the startup screen is displayed for a minimum of 5 seconds before disappearing. If <code>forcep</code> is true then the startup screen disappears when <code>dismiss-splash-screen</code> is called. The default value of <code>forcep</code> is <code>nil</code> . If <code>dismiss-splash-screen</code> is not called, the startup screen appears for 30 seconds. Note: the user can dismiss the startup screen by clicking on it. For more information about specifying a startup screen in your application, see the entry for <code>:startup-bitmap-file</code> in the <i>LispWorks Delivery User Guide</i> .

latin-1-code-pages	<i>Variable</i>
Summary	Windows Code Pages for which Latin-1 encoded files are used.
Package	<code>win32</code>
Initial Value	<code>(1252 28591)</code>
Description	The value of <code>*latin-1-code-pages*</code> is a list of integers, which must be Windows code page identifiers. When the current Code Page is on this list, the default file encoding detection algorithm will cause <code>(:latin-1 :encoding-error-action 63)</code> to be used for file

I/O. Files will be written as Latin-1 with '?' replacing any non-Latin-1 character. This is faster than converting to the code page.

If `safe-locale-file-encoding` is used for file encoding detection, then the `:latin-1-safe` external format will be used.

Note: the LispWorks editor binds `*latin-1-code-pages*` to `nil` when reading and writing files, in order to ensure that code page characters outside of Latin-1 are handled regardless of the configuration of `open`.

See also `*file-encoding-detection-algorithm*`

multibyte-code-page-ef *Variable*

Summary Holds the external format corresponding to the current Windows multi-byte code page.

Package `win32`

Description This variable holds the external format corresponding to the current Windows multi-byte code page. It is automatically initialized to the right value, when the image is started. If you change the code page (using `_setmbcp`), you need to set this variable, too.

See also `locale-file-encoding`

set-application-themed *Function*

Summary Controls whether LispWorks should be themed.

Package `win32`

Signature	<code>set-application-themed on/off</code>	
Arguments	<i>on/off</i>	A generalized boolean.
Description	<p>The function <code>set-application-themed</code> controls whether a LispWorks application should be themed.</p> <p>On Windows XP, LispWorks is "themed", that is it uses the current theme of the desktop. You can switch this off by calling</p>	
	<pre>(win32:set-application-themed nil)</pre>	
<p>On non-XP systems, or when the application does not have Common Controls 6, this call has no effect.</p> <p><code>set-application-themed</code> affects only windows that are created after it was called. Normally, it should be called before any window is created, so all LispWorks windows will appear with the same theme. However, <code>set-application-themed</code> can be called multiple times in the same run.</p>		

17.2 Windows registry API

		<i>Function</i>
	close-registry-key	
Summary	Closes a handle to an open registry key.	
Signature	<code>close-registry-key handle &key errorp => successp, error-code</code>	
Arguments	<i>handle</i>	A handle to an open registry key.
Values	<i>successp</i>	A boolean.
	<i>error-code</i>	An integer error code or <code>nil</code> .

Description	The function <code>close-registry-key</code> closes <i>handle</i> , which should be an open registry key handle. The return value on success is <code>t</code> . If an error occurs and <i>errorp</i> is true then an error is signalled. Otherwise, the return values are <code>nil</code> and the Windows <i>error-code</i> . The default value of <i>errorp</i> is <code>t</code> .
See also	<code>create-registry-key</code> <code>open-registry-key</code>

collect-registry-subkeys *Function*

Summary	Returns names of the subkeys of a registry key.	
Signature	<code>collect-registry-subkeys subkey &key root max-name-size max-names errorp value-function => subsubkeys</code>	
Arguments	<code>subkey</code>	A string specifying the name of the key.
	<code>root</code>	A keyword or handle.
	<code>max-name-size</code>	An integer.
	<code>max-names</code>	An integer.
	<code>errorp</code>	A boolean.
	<code>value-function</code>	A function designator or <code>nil</code> .
Values	<code>subsubkeys</code>	A list.
Description	The function <code>collect-registry-subkeys</code> returns a list of names which are subsubkeys of <i>subkey</i> under the key <i>root</i> . <i>subkey</i> and <i>root</i> are interpreted as described for <code>create-registry-key</code> . The default value of <i>root</i> is <code>:user</code> .	

max-name-size specifies the maximum length of the returned name. If the name is longer than this, an error is signalled. The default value of *max-name-size* is 256.

max-names specifies the maximum number of names returned. Names after this number are ignored. The default value of *max-names* is `most-positive-fixnum`.

If *value-function* is `non-nil`, it should be a function with signature

`value-function handle subsubkey-name => name, collectp`

value-function is funcalled for each subsubkey with the handle of *subkey* and the name of the subsubkey. If *collectp* is `non-nil` then *name* is collected into the list *subsubkeys* to return from `collect-registry-subkeys`. Otherwise it is ignored.

If *value-function* is `nil`, then the returned *subsubkeys* is a list of strings naming all (subject to *max-names*) of the subsubkeys. The default value of *value-function* is `nil`.

If an error occurs opening *subkey* and *errorp* is true then an error is signalled. Otherwise, *subsubkeys* is returned as `nil` if *subkey* could not be opened. The default value of *errorp* is `t`.

See also `collect-registry-values`
 `create-registry-key`

collect-registry-values *Function*

Summary Returns the values of a registry key.

Signature `collect-registry-values subkey &key root max-name-size max-buffer-size expected-type errorp value-function => values-alist`

Arguments *subkey* A string specifying the name of the key.

root A keyword or handle.

	<i>max-name-size</i>	An integer.
	<i>max-buffer-size</i>	An integer.
	<i>expected-type</i>	A keyword or <code>t</code> .
	<i>errorp</i>	A boolean.
	<i>value-function</i>	A function or symbol.
Values	<i>values-alist</i>	An alist.
Description	<p>The function <code>collect-registry-values</code> returns an alist of all of the values of <i>subkey</i> under the key <i>root</i>. <i>subkey</i> and <i>root</i> are interpreted as described for <code>create-registry-key</code>. The default value of <i>root</i> is <code>:user</code>.</p> <p><i>max-name-size</i> specifies the maximum length of the returned name. If the name is longer than this, an error is signalled. The default value of <i>max-name-size</i> is 256.</p> <p><i>max-buffer-size</i> specifies the maximum length in bytes of the data. If the data is longer than this, an error is signalled. The default value of <i>max-buffer-size</i> is 1024.</p> <p>If <i>value-function</i> is <code>nil</code>, the returned <i>values-alist</i> is an association list containing pairs (<code>name . data</code>) consisting of the names and data of the values of <i>subkey</i>. <i>expected-type</i> controls how certain types are converted to Lisp objects as described for <code>enum-registry-value</code>. The default value of <i>expected-type</i> is <code>t</code>.</p> <p>If <i>value-function</i> is non-<code>nil</code>, it should be a function with signature</p> <pre><code>value-function handle subsubkey-name-and-value => name-and-value, collectp</code></pre> <p><i>value-function</i> is funcalled for each subsubkey with the handle of <i>subkey</i> and a cons of the name and value of the subsubkey. If <i>collectp</i> is non-<code>nil</code> then <i>name-and-value</i> is collected into</p>	

the alist *values-alist* to return from `collect-registry-values`. Otherwise *name-and-value* is ignored.

If an error occurs and *errorp* is true, then an error is signalled. Otherwise, *values-alist* is returned as `nil` if *subkey* could not be opened at all or contains `nil` for the data of any particular pair that cannot be read. The default value of *errorp* is `t`.

See also `collect-registry-subkeys`
`create-registry-key`
`enum-registry-value`

create-registry-key *Function*

Summary	Creates a new registry key.	
Signature	<code>create-registry-key subkey &key class root access errorp => handle, disposition, error-code</code>	
Arguments	<i>subkey</i>	A string specifying the name of the key.
	<i>class</i>	A string.
	<i>root</i>	A keyword or handle.
	<i>access</i>	A keyword or an integer.
	<i>errorp</i>	A generalized boolean.
Values	<i>handle</i>	The handle of the new key.
	<i>disposition</i>	A keyword, either <code>:created-new-key</code> or <code>:opened-existing-key</code> .
	<i>error-code</i>	An integer error code or <code>nil</code> .
Description	The function <code>create-registry-key</code> creates a new registry key named <i>subkey</i> under the parent key <i>root</i> . If the key already exists, it is opened and returned.	

subkey is a string specifying a path from a root. Each component of the path is separated by a backslash. Use "" to denote the null path (that is, the root).

class can be used to specify the class of the key if it is created.

root should be a handle to an open registry key (for example a key returned by `create-registry-key` or `open-registry-key` or one of the keywords `:classes`, `:user`, `:local-machine` or `:users` which represent the standard top level roots in the registry. The default value of *root* is `:user`.

If *access* is `:read`, then the key is created with `KEY_READ` permissions. If *access* is `:write`, then the key is created with `KEY_WRITE` permissions. If *access* is an integer, then the value *access* specifies the desired Win32 access rights. The default value of *access* is `:read`.

The return values on success are the handle of the new key and a keyword `:created-new-key` or `:opened-existing-key` indicating whether a new key was created or opened.

If an error occurs and *errorp* is true then an error is signalled. Otherwise, the return values are `nil`, `nil` and the Windows *error-code*. The default value of *errorp* is `t`.

See also `delete-registry-key`
`open-registry-key`

delete-registry-key *Function*

Summary Deletes a registry key.

Signature `delete-registry-key subkey &key root errorp => successp, error-code`

Arguments *subkey* A string specifying the name of the key.
root A keyword or handle.

	<i>errorp</i>	A generalized boolean.
Values	<i>successp</i>	A boolean.
	<i>error-code</i>	An integer error code or <code>nil</code> .
Description		<p>The function <code>delete-registry-key</code> deletes the registry key named <i>subkey</i> under the parent key <i>root</i>.</p> <p><i>subkey</i> and <i>root</i> are interpreted as described for <code>create-registry-key</code>. The default value of <i>root</i> is <code>:user</code>.</p> <p>The value <code>t</code> is returned if the key is deleted successfully.</p> <p>If an error occurs and <i>errorp</i> is true then an error is signalled. Otherwise, the return values are <code>nil</code> and the Windows <i>error-code</i>. The default value of <i>errorp</i> is <code>t</code>.</p>
See also		<code>create-registry-key</code>

enum-registry-value		<i>Function</i>
Summary		Enumerates the values of a registry key.
Signature		<code>enum-registry-value subkey index &key root max-name-size max-buffer-size expected-type errorp => name, data-type, data, error-code</code>
Arguments	<i>subkey</i>	A string specifying the name of the key.
	<i>index</i>	An integer.
	<i>root</i>	A keyword or handle.
	<i>max-name-size</i>	An integer.
	<i>max-buffer-size</i>	An integer.
	<i>expected-type</i>	A keyword or <code>t</code> .
	<i>errorp</i>	A boolean.

Values	<i>name</i>	A string.
	<i>data-type</i>	A keyword.
	<i>data</i>	A lisp object.
	<i>error-code</i>	An integer error code or <code>nil</code> .
Description	<p>The function <code>enum-registry-value</code> allows the values of subkey under the key <i>root</i> to be enumerated.</p> <p><i>subkey</i> and <i>root</i> are interpreted as described for <code>create-registry-key</code>. The default value of <i>root</i> is <code>:user</code>.</p> <p><i>index</i> specifies which value to return, with 0 being the first item.</p> <p><i>max-name-size</i> specifies the maximum length of the returned name. If the name is longer than this, an error is signalled. The default value of <i>max-name-size</i> is 256.</p> <p><i>max-buffer-size</i> specifies the maximum length in bytes of the value. The value is longer than this, an error is signalled. The default value of <i>max-buffer-size</i> is 1024.</p> <p>If the value exists (that is, <i>index</i> is not too large), then the return values are the name, data type and data associated with the value in the registry. The argument <i>expected-type</i> controls how certain data types are converted to Lisp objects as follows:</p>	

Table 17.1 Conversion of registry values to Lisp objects

<i>data-type</i>	<i>expected-type</i>	Description of converted data
<code>:string</code>	<code>:lisp-object</code>	String made with <code>read-from-string</code>
<code>:string</code>	Not supplied	String, exactly as in the registry

Table 17.1 Conversion of registry values to Lisp objects

<i>data-type</i>	<i>expected-type</i>	Description of converted data
<code>:environment-string</code>	<code>:string</code>	String, exactly as in the registry
<code>:environment-string</code>	Not supplied	String, environment variables expanded
<code>:integer</code>	Not supplied	Integer
<code>:little-endian-integer</code>	Not supplied	Integer
<code>:binary</code>	Not supplied	A newly allocated foreign object
<code>:binary</code>	<code>:lisp-object</code>	Vector, element type (<code>unsigned-byte 8</code>)

The default value of *expected-type* is `t`.

If an error occurs and *errorp* is true, then an error is signalled. Otherwise, the return values are `nil`, `nil`, `nil` and the Windows *error-code*. The default value of *errorp* is `t`.

See also [create-registry-key](#)

open-registry-key *Function*

Summary Opens a registry key.

Signature `open-registry-key subkey &key root access errorp => handle, error-code`

Arguments `subkey` A string specifying the name of the key.

`root` A keyword or handle.

	<i>access</i>	An integer or keyword.
	<i>errorp</i>	A generalized boolean.
Values	<i>handle</i>	The handle of the key.
	<i>error-code</i>	An integer error code or <code>nil</code> .
Description		<p>The function <code>open-registry-key</code> opens a registry key named <i>subkey</i> under the parent key <i>root</i>.</p> <p><i>subkey</i> and <i>root</i> are interpreted as described for <code>create-registry-key</code>. If <i>subkey</i> is an empty string, then the <i>root</i> key is returned. The default value of <i>root</i> is <code>:user</code>.</p> <p>If <i>access</i> is <code>:read</code>, then it opens the key with <code>KEY_READ</code> permissions. If <i>access</i> is <code>:write</code>, then it opens the key with <code>KEY_WRITE</code> permissions. If <i>access</i> is an integer, then the value <i>access</i> specifies the desired Win32 access rights. If <i>access</i> is omitted and <i>root</i> is <code>:user</code>, then <code>open-registry-key</code> uses <code>KEY_ALL_ACCESS</code>. Otherwise it uses <code>KEY_READ</code>.</p> <p>The return value on success is the <i>handle</i> of the opened key.</p> <p>If an error occurs and <i>errorp</i> is true, then an error is signalled. Otherwise, the return values are <code>nil</code> and the Windows <i>error-code</i>. The default value of <i>errorp</i> is <code>t</code>.</p>
See also		<code>create-registry-key</code>

query-registry-key-info *Function*

Summary	Returns information about an open registry key handle.	
Signature	<code>query-registry-key-info key => info, error-code</code>	
Arguments	<i>key</i>	A handle.
Values	<i>info</i>	A property list.

	error-code	An integer error code or <code>nil</code> .
Description	The function <code>query-registry-key-info</code> returns a plist of information about the open registry key handle <i>key</i> . The elements of the plist <i>info</i> are:	
	:class	A string naming the class of the key, if any.
	:subkeys-count	An integer giving the number of subkeys.
	:subkey-max-len	An integer giving the length of the longest subkey name.
	:class-name-max-len	An integer giving the length of the longest class name.
	:values-count	An integer giving the number of values.
	:value-max-len	An integer giving the length of the longest value name.
	:max-data-len	An integer giving the length of the longest value data.
	:security-len	An integer giving the length of the security descriptor.

		<i>Function</i>
Summary	Returns a value stored in the registry.	
Signature	<code>query-registry-value <i>subkey</i> <i>name</i> &key <i>root</i> <i>expected-type</i> errorp => <i>data</i>, <i>successp</i>, <i>error-code</i></code>	
Arguments	<i>subkey</i>	A string specifying the name of the key.
	<i>name</i>	A string specifying the name of the value.
	<i>root</i>	A keyword or handle.

	<i>expected-type</i>	A keyword or t.
	<i>errorp</i>	A boolean.
Values	<i>data</i>	A Lisp object.
	<i>successp</i>	A boolean.
	<i>error-code</i>	An integer error code or nil.
Description		<p>The function <code>query-registry-value</code> returns the value associated with <i>name</i> in <i>subkey</i> under the key <i>root</i>. <i>subkey</i> and <i>root</i> are interpreted as described for <code>create-registry-key</code>. If <i>subkey</i> is an empty string, then the <i>root</i> key is returned. The default value of <i>root</i> is :user.</p> <p>If the value exists, then the return values are the data and true. <i>expected-type</i> controls how certain types are converted to the Lisp object <i>data</i> as described for <code>enum-registry-value</code>. The default value of <i>expected-type</i> is t.</p> <p>If an error occurs and <i>errorp</i> is true then an error is signalled. Otherwise, the return values are nil, nil and the Windows <i>error-code</i>. The default value of <i>errorp</i> is t.</p>
See also		create-registry-key enum-registry-value

		<i>Function</i>
Summary		The predicate for whether a registry key can be opened.
Signature		<code>registry-key-exists-p subkey &key root access => existsp</code>
Arguments	<i>subkey</i>	A string specifying the name of the key.
	<i>root</i>	A keyword or handle.
	<i>access</i>	An integer or keyword.

Values	<code>existsp</code>	A boolean.
Description		<p>The function <code>registry-key-exists-p</code> checks whether the registry key named <i>subkey</i> can be opened under the parent key <i>root</i> with the supplied <i>access</i> permissions.</p> <p><i>subkey</i> and <i>root</i> are interpreted as described for <code>create-registry-key</code>. The default value of <i>root</i> is <code>:user</code>.</p> <p>If <i>access</i> is <code>:read</code>, then it opens the key with <code>KEY_READ</code> permissions. If <i>access</i> is <code>:write</code>, then it opens the key with <code>KEY_WRITE</code> permissions. If <i>access</i> is an integer, then the value <i>access</i> specifies the desired Win32 access rights. If <i>access</i> is omitted and <i>root</i> is <code>:user</code>, then <code>registry-key-exists-p</code> uses <code>KEY_ALL_ACCESS</code>. Otherwise it uses <code>KEY_READ</code>.</p> <p><code>registry-key-exists-p</code> closes the key before returning, but the return value is <code>t</code> if the key could actually be opened and <code>nil</code> otherwise.</p>
See also		<code>create-registry-key</code>

registry-value *Accessor*

Summary	Gets or sets a value in the registry.	
Signature	<code>registry-value subkey name &key root expected-type errorp => data, successp, error-code</code> <code>(setf registry-value) value subkey name &key root expected-type errorp => value</code>	
Arguments	<code>subkey</code>	A string specifying the name of the key.
	<code>name</code>	A string specifying the name of the value.
	<code>root</code>	A keyword or handle.
	<code>expected-type</code>	A keyword or <code>t</code> .
	<code>errorp</code>	A boolean.

Values	<i>data</i>	A Lisp object.
	<i>successp</i>	A boolean.
	<i>error-code</i>	An integer error code or <code>nil</code> .
Description	<p>The function <code>registry-value</code> returns the value associated with <i>name</i> in <i>subkey</i> under the key <i>root</i>.</p> <p><i>subkey</i> and <i>root</i> are interpreted as described for <code>create-registry-key</code>. The default value of <i>root</i> is <code>:user</code>.</p> <p>If the value exists, then the return values are the data and true. <i>expected-type</i> controls how certain types are converted to Lisp objects as described for <code>enum-registry-value</code>. The default value of <i>expected-type</i> is <code>t</code>.</p> <p>If an error occurs and <i>errorp</i> is true then an error is signalled. Otherwise, the return values are <code>nil</code>, <code>nil</code> and the Windows <i>error-code</i>. The default value of <i>errorp</i> is <code>t</code>.</p> <p>The function (<code>setf registry-value</code>) sets the value associated with <i>name</i> in <i>subkey</i> under the key <i>root</i>, creating the subkey if necessary. The default value of <i>root</i> is <code>:user</code>.</p>	
See also	<code>set-registry-value</code>	

set-registry-value *Function*

Summary	Stores a value in the registry.	
Signature	<code>set-registry-value data subkey name &key root expected-type errorp => error-code</code>	
Arguments	<i>data</i>	A Lisp object.
	<i>subkey</i>	A string specifying the name of the key.
	<i>name</i>	A string specifying the name of the value.
	<i>root</i>	A keyword or handle.

	<i>expected-type</i>	A keyword or t.
	<i>errorp</i>	A boolean.
Values	<i>error-code</i>	An integer error code or nil.
Description		The function <code>set-registry-value</code> sets the value associated with <i>name</i> in <i>subkey</i> under the key <i>root</i> . <i>subkey</i> and <i>root</i> are interpreted as described for <code>create-registry-key</code> . The default value of <i>root</i> is :user. The stored value is derived from <i>data</i> , converted according to <i>expected-type</i> as follows:

Table 17.2 Conversion of Lisp objects to registry values

Lisp data	<i>expected-type</i>	Registry type
A string	:string	REG_SZ exactly as in <i>data</i>
Lisp value	:lisp-object	REG_SZ made with <code>prin1-to-string</code> of <i>data</i>
An integer	:integer	REG_DWORD containing <i>data</i>
A foreign pointer	:binary	REG_BINARY containing bytes of one element at the pointer
An array	:binary	REG_BINARY containing bytes from the array

The default value of *expected-type* is t.

If an error occurs and *errorp* is true then an error is signalled.
The default value of *errorp* is t.

See also [create-registry-key](#)
[registry-value](#)

with-registry-key Macro

Summary Runs code with an open registry key handle.

Signature `with-registry-key (handle subkey &key root access errorp)
 &body body => values`

Arguments `handle` A variable name.
`subkey` A string specifying the name of the key.
`root` A keyword or handle.
`access` An integer or keyword.
`errorp` A boolean.

Values `values` The values returned by `body`.

Description The macro `with-registry-key` evaluates `body` with the variable `handle` bound to the registry key handle opened as if by calling

```
(open-registry-key subkey :root root
                  :access access
                  :errorp errorp)
```

`subkey` and `root` are interpreted as described for `create-registry-key`.

If `errorp` is `nil` and `subkey` cannot be opened then `body` is not evaluated.

See also [create-registry-key](#)

17.3 DDE client interface reference entries

dde-advise-start	<i>Function</i>																
Summary	Sets up an advise loop on a specified data item for a conversation.																
Package	<code>win32</code>																
Signature	<code>dde-advise-start <i>conversation item</i> &key <i>key function format datap type errorp => result</i></code>																
Arguments	<table border="0"> <tr> <td><i>conversation</i></td><td>A conversation object.</td></tr> <tr> <td><i>item</i></td><td>A string or symbol.</td></tr> <tr> <td><i>key</i></td><td>An object.</td></tr> <tr> <td><i>function</i></td><td>A function name.</td></tr> <tr> <td><i>format</i></td><td>A clipboard format specifier.</td></tr> <tr> <td><i>datap</i></td><td>A boolean.</td></tr> <tr> <td><i>type</i></td><td>A keyword.</td></tr> <tr> <td><i>errorp</i></td><td>A boolean.</td></tr> </table>	<i>conversation</i>	A conversation object.	<i>item</i>	A string or symbol.	<i>key</i>	An object.	<i>function</i>	A function name.	<i>format</i>	A clipboard format specifier.	<i>datap</i>	A boolean.	<i>type</i>	A keyword.	<i>errorp</i>	A boolean.
<i>conversation</i>	A conversation object.																
<i>item</i>	A string or symbol.																
<i>key</i>	An object.																
<i>function</i>	A function name.																
<i>format</i>	A clipboard format specifier.																
<i>datap</i>	A boolean.																
<i>type</i>	A keyword.																
<i>errorp</i>	A boolean.																
Values	<i>result</i> A boolean.																
Description	<p>The <code>dde-advise-start</code> function sets up an advise loop for the data item specified by <i>item</i> on the specified <i>conversation</i>. The argument <i>format</i> should be one of the following:</p> <ul style="list-style-type: none"> • A DDE format specifier, consisting of either a standard clipboard format or a registered clipboard format. • A string containing either the name of a standard clipboard format (without the <code>CF_</code> prefix), or the name of a registered clipboard format. 																

- A symbol, in which case its print name is taken to specify the clipboard format.
- The keyword :text – the default value of *format*. The keyword :text is treated specially. If supported by the server it uses the CF_UNICODETEXT clipboard format, otherwise it used the CF_TEXT format.

The argument *type* specifies how the response data should be converted to a Lisp object. For text formats, the default value indicates that a Lisp string should be created. The value :string-list may be specified to indicate that the return value should be taken as a tab-separated list of strings; in this case the Lisp return value is a list of strings. The default conversation class only supports text formats, unless *type* is specified as :foreign, which can be used with any clipboard format. It returns a `clipboard-item` structure, containing a foreign pointer to the data, the data length, and the format identifier.

If *datap* is `t` (the default value), a hot link is established, where the new data is supplied whenever it changes. If *datap* is `nil`, a warm link is established, where the data is not passed, and must be explicitly requested using `dde-request`.

The argument *key* is used to identify this link. If specified as `nil` (the default value), it defaults to the conversation. Multiple links are permitted on a conversation with the same *item* and *format* values, as long as their *key* values differ.

If the link is established, the return value *result* is `t`. If the link could not be established, the behavior depends on the value of *errorp*. If *errorp* is `t` (the default value), LispWorks signals an error. If it is `nil`, the function returns `nil` to indicate failure.

If the link is established, the function *function* is called whenever the data changes. If *function* is `nil` (the default value), then the generic function `dde-client-advise-data` will be called.

The function specified by *function* should have a lambda list similar to the following:

```
key item data &key conversation &allow-other-keys
```

The arguments *key* and *item* identify the link. The argument *data* contains the new data for hot links; for warm links it is *nil*.

See also

```
dde-advise-start*  
dde-advise-stop  
dde-client-advise-data
```

dde-advise-start* *Function*

Summary Sets up an advise loop for a specified data item for an automatically managed conversation.

Package `win32`

Signature `dde-advise-start* service topic item &key key function format datap type errorp connect-error-p new-conversation-p => result`

Arguments *service* A string or symbol.

topic A string or symbol.

item A string or symbol.

key An object.

function A function name.

format A clipboard format specifier.

datap A boolean.

type A keyword.

errorp A boolean.

connect-error-p A boolean.

new-conversation-p

A boolean.

Values *result* A boolean.

Description The **dde-advise-start*** function is similar to the **dde-advise-start**, and sets up an advise loop for the data item specified by *item* on a conversation recognizing the *service/topic* pair.

See **dde-advise-start** for information on the *format*, *type*, and *datap* arguments.

The argument *key* is used to identify this link. If specified as **nil** (the default value), it defaults to the conversation. Multiple links are permitted on a conversation with the same *item* and *format* values, as long as their *key* values differ.

If the link is established, the return value *result* is **t**. If the link could not be established, the behavior depends on the value of *errorp*. If *errorp* is **t** (the default value), LispWorks signals an error. If it is **nil**, the function returns **nil** to indicate failure.

If the link is established, the function *function* will be called whenever the data changes. If *function* is **nil** (the default value), the generic function **dde-client-advise-data** will be called.

The function specified by *function* should have a lambda list similar to the following:

key item data &key conversation &allow-other-keys

The arguments *key* and *item* identify the link. The argument *data* contains the new data for hot links; for warm links it is **nil**.

See also **dde-advise-start**
dde-advise-stop

```
dde-advise-stop*
dde-client-advise-data
```

dde-advise-stop	<i>Function</i>														
Summary	Removes a link from a conversation specified by a given item and key.														
Package	<code>win32</code>														
Signature	<code>dde-advise-stop <i>conversation item &key key format errorp disconnectp no-advise-ok => result</i></code>														
Arguments	<table border="0"> <tr> <td><i>conversation</i></td><td>A conversation object.</td></tr> <tr> <td><i>item</i></td><td>A string or symbol.</td></tr> <tr> <td><i>key</i></td><td>An object.</td></tr> <tr> <td><i>format</i></td><td>A clipboard format specifier.</td></tr> <tr> <td><i>errorp</i></td><td>A boolean.</td></tr> <tr> <td><i>disconnectp</i></td><td>A boolean.</td></tr> <tr> <td><i>no-advise-ok</i></td><td>A boolean.</td></tr> </table>	<i>conversation</i>	A conversation object.	<i>item</i>	A string or symbol.	<i>key</i>	An object.	<i>format</i>	A clipboard format specifier.	<i>errorp</i>	A boolean.	<i>disconnectp</i>	A boolean.	<i>no-advise-ok</i>	A boolean.
<i>conversation</i>	A conversation object.														
<i>item</i>	A string or symbol.														
<i>key</i>	An object.														
<i>format</i>	A clipboard format specifier.														
<i>errorp</i>	A boolean.														
<i>disconnectp</i>	A boolean.														
<i>no-advise-ok</i>	A boolean.														
Values	<i>result</i> A boolean.														
Description	<p>The function <code>dde-advise-stop</code> removes a particular link from <i>conversation</i> specified by <i>item</i>, <i>format</i> and <i>key</i>. If <i>key</i> is the last key for the <i>item/format</i> pair, the advise loop for the pair is terminated.</p> <p>If <i>disconnectp</i> is <code>t</code>, and the last advise loop for the conversation is terminated, the conversation is disconnected.</p> <p>Attempting to remove a link that does not exist raises an error, unless <i>no-advise-ok</i> is <code>t</code>.</p>														

If this function succeeds, it returns `t`. If it fails, the behavior depends on the value of `errorp`. If `errorp` is `t` (the default value), LispWorks signals an error. If `errorp` is `nil`, the function returns `nil` to indicate failure.

See also [dde-advise-start](#)
[dde-advise-start*](#)
[dde-advise-stop*](#)
[dde-client-advise-data](#)

		<i>Function</i>
Summary	Removes a link from an automatically managed conversation specified by a given item and key.	
Package	<code>win32</code>	
Signature	<code>dde-advise-stop* service topic item &key key format errorp disconnectp => result</code>	
Arguments	<code>service</code> A string or symbol. <code>topic</code> A string or symbol. <code>item</code> A string or symbol. <code>key</code> An object. <code>format</code> A clipboard format specifier. <code>errorp</code> A boolean. <code>disconnectp</code> A boolean.	
Values	<code>result</code> A boolean.	
Description	The function <code>dde-advise-stop*</code> is similar to the function <code>dde-advise-stop</code> , and removes a particular link from a con-	

versation specified by the *service/topic* pair indicated by *item*, *format* and *key*. If *key* is the last key for the *item/format* pair, the advise loop for the pair is terminated.

If *disconnectp* is `t` (the default value), and the last advise loop for the conversation is terminated, the conversation is disconnected.

If this function succeeds, it returns `t`. If it fails, the behavior depends on the value of *errorp*. If *errorp* is `t` (the default value), LispWorks signals an error. If *errorp* is `nil`, the function returns `nil` to indicate failure.

See also

`dde-advise-start`

`dde-advise-start*`

`dde-advise-stop`

`dde-client-advise-data`

Generic Function

Summary Called when data changes in an advise loop.

Package `win32`

Signature `dde-client-advise-data key item data &key &allow-other-keys =>`

Arguments `key` An object.

`item` A string or symbol.

`data` A string.

Values None.

Description The generic function `dde-client-advise-data` is the default function called when an advise loop informs a client that the data monitored by the loop has changed. By default it does nothing, but it may be specialized on the object used as the

key in `dde-advise-start` or `dde-advise-start*`, or on a client conversation class if the default key is used.

See also `dde-advise-start`
`dde-advise-stop`

dde-connect *Function*

Summary	Attempts to create a conversation with a specified DDE server.		
Package	<code>win32</code>		
Signature	<code>dde-connect service topic &key class errorp => object</code>		
Arguments	<code>service</code>	A symbol or string.	
	<code>topic</code>	A symbol or string.	
	<code>class</code>	The class of the conversation object to create.	
	<code>errorp</code>	A boolean.	
Values	<code>object</code>	A conversation object.	
Description	<p>The function <code>dde-connect</code> attempts to create a conversation with a DDE server. If <code>server</code> names a client service registered with <code>define-dde-client</code>, the registered service name is used as the DDE service name. If <code>server</code> is any other symbol, the print name of the symbol is used as the DDE service name. If <code>server</code> is a string, that string is used as the DDE service name.</p> <p>The <code>topic</code> argument specifies the DDE topic name to be used in the conversation. If it is a symbol, the symbol's print name is used. If it is a string, the string is used.</p> <p>The <code>class</code> argument specifies the class of the conversation object to create. It must be a subclass of <code>dde-client-</code></p>		

`conversation`, or `nil`. If it is `nil` (the default value), then a conversation of class `dde-client-conversation` is created, unless `server` names a client service registered with `define-dde-client`, in which case the registered class (if any) is used.

On executing successfully, this function returns a conversation object. If unsuccessful, the behavior depends on the value of `errorp`. If `errorp` is `t` (the default value), then an error is raised. If `errorp` is false, the function returns `nil`.

Note that conversation objects may only be used within the thread (lightweight process) in which they were created.

See also `dde-disconnect`

dde-disconnect *Function*

Summary Disconnects a conversation object.

Package `win32`

Signature `dde-disconnect conversation => result`

Arguments `conversation` A conversation object.

Values `result` A boolean.

Description The function `dde-disconnect` disconnects the conversation object. The conversation may no longer be used. If the conversation disconnects successfully, `t` is returned.

See also `dde-connect`

dde-execute *Function*

Summary An alternative syntax for `dde-execute-command`.

Package	<code>win32</code>						
Signature	<code>dde-execute conversation command &rest {args}* => result</code>						
Arguments	<table> <tr> <td><i>conversation</i></td><td>A conversation object.</td></tr> <tr> <td><i>command</i></td><td>A string or symbol.</td></tr> <tr> <td><i>args</i></td><td>An argument.</td></tr> </table>	<i>conversation</i>	A conversation object.	<i>command</i>	A string or symbol.	<i>args</i>	An argument.
<i>conversation</i>	A conversation object.						
<i>command</i>	A string or symbol.						
<i>args</i>	An argument.						
Values	<i>result</i> A boolean.						
Description	The function <code>dde-execute</code> provides an alternative syntax for <code>dde-execute-command</code> . Unlike <code>dde-execute-command</code> , <code>dde-execute</code> takes the arguments for <code>command</code> as a sequence of <code>args</code> following <code>&rest</code> , and does not have an argument for specifying how to handle an error.						
See also	<code>dde-execute*</code> <code>dde-execute-command*</code> <code>dde-execute-string</code>						

dde-execute*		<i>Function</i>								
Summary	An alternative syntax for <code>dde-execute-command*</code> .									
Package	<code>win32</code>									
Signature	<code>dde-execute* service topic command &rest {args}* => result</code>									
Arguments	<table> <tr> <td><i>service</i></td><td>A string or symbol.</td></tr> <tr> <td><i>topic</i></td><td>A string symbol.</td></tr> <tr> <td><i>command</i></td><td>A string or symbol.</td></tr> <tr> <td><i>args</i></td><td>An argument.</td></tr> </table>		<i>service</i>	A string or symbol.	<i>topic</i>	A string symbol.	<i>command</i>	A string or symbol.	<i>args</i>	An argument.
<i>service</i>	A string or symbol.									
<i>topic</i>	A string symbol.									
<i>command</i>	A string or symbol.									
<i>args</i>	An argument.									

Values	<i>result</i>	A boolean.
Description	The function <code>dde-execute*</code> provides an alternative syntax for <code>dde-execute-command*</code> . Unlike <code>dde-execute-command*</code> , <code>dde-execute*</code> takes the arguments for <i>command</i> as a sequence of <i>args</i> following <code>&rest</code> , and does not have any arguments for specifying how to handle errors.	
See also	<code>dde-execute</code> <code>dde-execute-command</code> <code>dde-execute-string</code>	

dde-execute-command *Function*

Summary	Sends a command string to a specified conversation.	
Package	<code>win32</code>	
Signature	<code>dde-execute-command <i>conversation</i> <i>command</i> <i>arg-list</i> &key <i>errorp</i> => <i>result</i></code>	
Arguments	<i>conversation</i>	A conversation object.
	<i>command</i>	A string or symbol.
	<i>arg-list</i>	A list of strings, integers, and floats.
	<i>errorp</i>	A boolean.
Values	<i>result</i>	A boolean.
Description	The function <code>dde-execute-command</code> sends a command string to the conversation specified by <i>conversation</i> . The command string consists of <i>command</i> and <i>arg-list</i> , which are combined using the appropriate argument-marshalling conventions. By default, the syntax is [<i>command</i> (<i>arg1</i> , <i>arg2</i> ,...)]	

On success, this function returns a result of `t`. On failure, the behavior depends on the value of the `errorp` argument. If `errorp` is `t` (the default value), LispWorks signals an error. If it is `nil`, the function returns `nil` to indicate failure.

See also

`dde-execute`
`dde-execute-string`

dde-execute-command* *Function*

Summary Sends a command string to a specified service on a given topic.

Package `win32`

Signature `dde-execute-command* service topic command arg-list &key
errorp connect-error-p new-conversation-p => result`

Arguments `service` A string or symbol.

`topic` A string or symbol.

`command` A string or symbol.

`arg-list` A list of strings, integers, and floats.

`errorp` A boolean.

`connect-error-p` A boolean.

`new-conversation-p`

A boolean.

Values `result` A boolean.

Description The function `dde-execute-command*` is similar to `dde-execute-command`, and sends a command string to the server specified by `service` on a topic given by `topic`. The command string consists of `command` and `arg-list`, which are combined

using the appropriate argument-marshalling conventions. By default, the syntax is

```
[command(arg1,arg2,...)]
```

If *server* names a client service registered with **define-dde-client**, the registered service name is used as the DDE service name. If *server* is any other symbol, the print name of the symbol is used as the DDE service name. If *server* is a string, that string is used as the DDE service name.

The *topic* argument specifies the DDE topic name to be used in the conversation. If it is a symbol, the symbol's print name is used. If it is a string, the string is used.

If necessary, the function **dde-execute-command*** creates a conversation for the duration of the transaction, but if a suitable conversation already exists, the transaction is executed over that conversation. Hence, if several transactions will be made with the same *service* and *topic*, placing them inside a **with-dde-conversation** prevents a new conversation being established for each transaction.

If *new-conversation-p* is set to **t** a new conversation is always established for the transaction. This new conversation is always automatically disconnected when the transaction is completed.

If *connect-error-p* is **t** (the default value) and a conversation cannot be established, then LispWorks signals an error. If it is **nil**, **dde-execute-command*** returns **nil** if a conversation cannot be established. This allows the caller to distinguish between the cases when the server is not running, and when the server is running but the transaction fails.

Upon success, this function returns a result of **t**. On failure, the behavior depends on the value of the *errorp* argument. If *errorp* is **t** (the default value), LispWorks signals an error. If it is **nil**, the function returns **nil** to indicate failure.

See also

dde-execute

dde-execute-string
dde-execute-command

	<i>Function</i>						
dde-execute-string							
Summary	Issues an execute transaction consisting of a specified string.						
Package	<code>win32</code>						
Signature	<code>dde-execute-string conversation command &key errorp => result</code>						
Arguments	<table><tr><td><i>conversation</i></td><td>A conversation object.</td></tr><tr><td><i>command</i></td><td>A string or symbol.</td></tr><tr><td><i>errorp</i></td><td>A boolean.</td></tr></table>	<i>conversation</i>	A conversation object.	<i>command</i>	A string or symbol.	<i>errorp</i>	A boolean.
<i>conversation</i>	A conversation object.						
<i>command</i>	A string or symbol.						
<i>errorp</i>	A boolean.						
Values	<i>result</i> A boolean.						
Description	<p>The function <code>dde-execute-string</code> issues an execute transaction consisting of the string <i>command</i>. No processing of the string is performed.</p> <p>On success, this function returns <code>t</code>. On failure, the behavior depends on the value of the <i>errorp</i> argument. If <i>errorp</i> is <code>t</code> (the default value), LispWorks signals an error. If it is <code>nil</code>, the function returns <code>nil</code> to indicate failure.</p>						
See also	<code>dde-execute</code> <code>dde-execute-command</code> <code>dde-execute-string*</code>						

	<i>Function</i>												
Summary	Issues an execute transaction consisting of a specified string on an automatically managed conversation.												
Package	<code>win32</code>												
Signature	<code>dde-execute-string* service topic command &key errorp connect-error-p new-conversation-p => result</code>												
Arguments	<table> <tr> <td><i>service</i></td><td>A symbol or string.</td></tr> <tr> <td><i>topic</i></td><td>A symbol or string.</td></tr> <tr> <td><i>command</i></td><td>A string or symbol.</td></tr> <tr> <td><i>errorp</i></td><td>A boolean.</td></tr> <tr> <td><i>connect-error-p</i></td><td>A boolean.</td></tr> <tr> <td><i>new-conversation-p</i></td><td>A boolean.</td></tr> </table>	<i>service</i>	A symbol or string.	<i>topic</i>	A symbol or string.	<i>command</i>	A string or symbol.	<i>errorp</i>	A boolean.	<i>connect-error-p</i>	A boolean.	<i>new-conversation-p</i>	A boolean.
<i>service</i>	A symbol or string.												
<i>topic</i>	A symbol or string.												
<i>command</i>	A string or symbol.												
<i>errorp</i>	A boolean.												
<i>connect-error-p</i>	A boolean.												
<i>new-conversation-p</i>	A boolean.												
Values	<i>result</i> A boolean.												
Description	<p>The function <code>dde-execute-string*</code> is similar to <code>dde-execute-string</code>, in that it issues an execute transaction consisting of the string <i>command</i>. However, the conversation across which <i>command</i> is issued is managed automatically. No processing of the string is performed.</p> <p>If <i>server</i> names a client service registered with <code>define-dde-client</code>, the registered service name is used as the DDE service name. If <i>server</i> is any other symbol, the print name of the symbol is used as the DDE service name. If <i>server</i> is a string, that string is used as the DDE service name.</p> <p>The <i>topic</i> argument specifies the DDE topic name to be used in the conversation. If it is a symbol, the symbol's print name is used. If it is a string, the string is used.</p>												

If necessary, the function `dde-execute-string*` will create a conversation for the duration of the transaction, but if a suitable conversation already exists, the transaction will be executed over that conversation. Hence, if several transactions will be made with the same *service* and *topic*, placing them inside a `with-dde-conversation` prevents a new conversation being established for each transaction.

If *new-conversation-p* is set to `t` a new conversation is always established for the transaction. This new conversation is always automatically disconnected when the transaction is completed.

If *connect-error-p* is `t` (the default value), then LispWorks signals an error if a conversation cannot be established. If it is `nil`, `dde-execute-string*` returns `nil` if a conversation cannot be established. This allows the caller to distinguish between the cases when the server is not running, and when the server is running but the transaction fails.

Upon success, the function returns `t`. On failure, the behavior depends on the value of the *errorp* argument. If *errorp* is `t` (the default value), LispWorks signals an error. If it is `nil`, the function returns `nil` to indicate failure.

See also

`dde-execute`
`dde-execute-command`
`dde-execute-string`

dde-item

Accessor

Summary

An accessor which can perform a request transaction or a poke transaction.

Package

`win32`

Signature

`dde-item conversation item &key format type errorp => result`

Arguments	<i>conversation</i>	A conversation object.
	<i>item</i>	A string or symbol.
	<i>format</i>	A clipboard format specifier.
	<i>type</i>	A keyword.
	<i>errorp</i>	A boolean.
Values	<i>result</i>	A boolean.
Description	<p>The accessor <code>dde-item</code> performs a request transaction when read. It performs a poke transaction when set.</p> <p>To illustrate, the following <code>dde-request</code> command</p>	
	<pre>(dde-request <i>conversation item :format format :type type :errorp errorp</i>)</pre>	
	<p>can also be issued using <code>dde-item</code> as follows:</p>	
	<pre>(dde-item <i>conversation item :FORMAT format :TYPE type :ERRORP errorp</i>)</pre>	
	<p>Similarly, the following <code>dde-poke</code> command</p>	
	<pre>(dde-poke <i>conversation item data :format format :type type :errorp errorp</i>)</pre>	
	<p>can be issued using <code>dde-item</code> as follows:</p>	
	<pre>(setf (dde-item <i>conversation item :format format :type type :errorp errorp</i>) <i>data</i>)</pre>	
	<p>except that the <i>format</i> always returns <i>data</i>.</p>	
	<p>Upon success, this function returns a <i>result</i> of <code>t</code>. On failure, the behavior depends on the value of the <i>errorp</i> argument. If <i>errorp</i> is <code>t</code> (the default value), LispWorks signals an error. If it is <code>nil</code>, the function returns <code>nil</code> to indicate failure.</p>	
See also	<p><code>dde-item*</code></p> <p><code>dde-poke</code></p> <p><code>dde-request</code></p>	

dde-item*	Accessor																
Summary	An accessor which can perform a request transaction or a poke transaction on an automatically managed conversation.																
Package	<code>win32</code>																
Signature	<code>dde-item* service topic item &key format type errorp connect-error-p new-conversation-p => result</code>																
Arguments	<table border="0"> <tr> <td><i>service</i></td><td>A string or symbol.</td></tr> <tr> <td><i>topic</i></td><td>A string or symbol.</td></tr> <tr> <td><i>item</i></td><td>A string or symbol.</td></tr> <tr> <td><i>format</i></td><td>A clipboard format specifier.</td></tr> <tr> <td><i>type</i></td><td>A keyword.</td></tr> <tr> <td><i>errorp</i></td><td>A boolean.</td></tr> <tr> <td><i>connect-error-p</i></td><td>A boolean.</td></tr> <tr> <td><i>new-conversation-p</i></td><td>A boolean.</td></tr> </table>	<i>service</i>	A string or symbol.	<i>topic</i>	A string or symbol.	<i>item</i>	A string or symbol.	<i>format</i>	A clipboard format specifier.	<i>type</i>	A keyword.	<i>errorp</i>	A boolean.	<i>connect-error-p</i>	A boolean.	<i>new-conversation-p</i>	A boolean.
<i>service</i>	A string or symbol.																
<i>topic</i>	A string or symbol.																
<i>item</i>	A string or symbol.																
<i>format</i>	A clipboard format specifier.																
<i>type</i>	A keyword.																
<i>errorp</i>	A boolean.																
<i>connect-error-p</i>	A boolean.																
<i>new-conversation-p</i>	A boolean.																
Values	<i>result</i> A boolean.																
Description	<p>The accessor <code>dde-item*</code> is similar to <code>dde-item</code>, and performs a request transaction when read. It performs a poke transaction when set.</p> <p>To illustrate, the following <code>dde-request*</code> command</p> <pre>(dde-request* service topic item :format format :type type :errorp errorp connect-error-p new-conversation-p)</pre> <p>can also be issued using <code>dde-item*</code> as follows:</p> <pre>(dde-item* service topic item :FORMAT format :TYPE type :ERRORP errorp connect-error-p new-conversation-p)</pre> <p>Similarly, the following <code>dde-poke*</code> command</p>																

```
(dde-poke* conversation item data :format format :type type  
:errorp errorp connect-error-p new-conversation-p)
```

can be issued using `dde-item*` as follows:

```
(setf (dde-item* conversation item :format format :type type  
:errorp errorp connect-error-p new-conversation-p) data)
```

except that the *format* always returns *data*.

If necessary, the accessor `dde-item*` creates a conversation for the duration of the transaction, but if a suitable conversation already exists, the transaction is executed over that conversation. If you need to make several transactions with the same *service* and *topic*, placing them inside a `with-dde-conversation` prevents a new conversation being established for each transaction.

If *new-conversation-p* is set to `t` a new conversation is always established for the transaction. This new conversation is always automatically disconnected when the transaction is completed.

If *connect-error-p* is `t` (the default value), then LispWorks signals an error if a conversation cannot be established. If it is `nil`, `dde-item*` returns `nil` if a conversation cannot be established. This allows the caller to distinguish between the cases when the server is not running, and when the server is running but the transaction fails.

On success, the function returns `t`. On failure, the behavior depends on the value of the *errorp* argument. If *errorp* is `t` (the default value), LispWorks signals an error. If it is `nil`, the function returns `nil` to indicate failure.

See also

`dde-item`

`dde-poke`

`dde-request`

dde-poke	<i>Function</i>												
Summary	Issues a poke transaction on a conversation, to set the value of a specified item.												
Package	<code>win32</code>												
Signature	<code>dde-poke conversation item data &key format type errorp => result</code>												
Arguments	<table> <tr> <td><i>conversation</i></td><td>A conversation object.</td></tr> <tr> <td><i>item</i></td><td>A string or symbol.</td></tr> <tr> <td><i>data</i></td><td>A string.</td></tr> <tr> <td><i>format</i></td><td>A clipboard format specifier.</td></tr> <tr> <td><i>type</i></td><td>A keyword.</td></tr> <tr> <td><i>errorp</i></td><td>A boolean.</td></tr> </table>	<i>conversation</i>	A conversation object.	<i>item</i>	A string or symbol.	<i>data</i>	A string.	<i>format</i>	A clipboard format specifier.	<i>type</i>	A keyword.	<i>errorp</i>	A boolean.
<i>conversation</i>	A conversation object.												
<i>item</i>	A string or symbol.												
<i>data</i>	A string.												
<i>format</i>	A clipboard format specifier.												
<i>type</i>	A keyword.												
<i>errorp</i>	A boolean.												
Values	<table> <tr> <td><i>result</i></td><td>A boolean.</td></tr> </table>	<i>result</i>	A boolean.										
<i>result</i>	A boolean.												
Description	<p>The function <code>dde-poke</code> issues a poke transaction on <i>conversation</i> to set the value of the item specified by <i>item</i> to the value specified by <i>data</i>. The argument <i>item</i> should be a string, or a symbol. If it is a symbol its print name is used.</p> <p>The argument <i>format</i> should be one of the following:</p> <ul style="list-style-type: none"> • A DDE format specifier, consisting of either a standard clipboard format or a registered clipboard format. • A string containing either the name of a standard clipboard format (without the <code>CF_</code> prefix), or the name of a registered clipboard format. • A symbol, in which case its print name is taken to specify the clipboard format. • The keyword <code>:text</code>. This is the default value. 												

The keyword `:text` is treated specially. If supported by the server it uses the `CF_UNICODETEXT` clipboard format, otherwise it used the `CF_TEXT` format.

For text transactions, the default value of *type* indicates that *data* is a Lisp string to be used. If *type* is `:string-list`, then *data* is taken to be a list of strings, and is sent as a tab-separated string.

Alternatively, *data* can be a `clipboard-item` structure, containing a foreign pointer to the data to send and the length of the data. In this case the *type* argument is ignored.

On success, this function returns `t`. On failure, the behavior depends on the value of the *errorp* argument. If *errorp* is `t` (the default value), LispWorks signals an error. If it is `nil`, the function returns `nil` to indicate failure.

See also [dde-item](#)
[dde-request](#)

	<i>Function</i>										
dde-poke*											
Summary	Issues a poke transaction on an automatically managed conversation, to set the value of a specified item.										
Package	<code>win32</code>										
Signature	<code>dde-poke* service topic item data &key format type errorp connect-error-p new-conversation-p => result</code>										
Arguments	<table border="0"> <tr> <td><i>service</i></td><td>A symbol or string.</td></tr> <tr> <td><i>topic</i></td><td>A symbol or string.</td></tr> <tr> <td><i>item</i></td><td>A string or symbol.</td></tr> <tr> <td><i>data</i></td><td>A string.</td></tr> <tr> <td><i>format</i></td><td>A clipboard format specifier.</td></tr> </table>	<i>service</i>	A symbol or string.	<i>topic</i>	A symbol or string.	<i>item</i>	A string or symbol.	<i>data</i>	A string.	<i>format</i>	A clipboard format specifier.
<i>service</i>	A symbol or string.										
<i>topic</i>	A symbol or string.										
<i>item</i>	A string or symbol.										
<i>data</i>	A string.										
<i>format</i>	A clipboard format specifier.										

	<i>type</i>	A keyword.
	<i>errorp</i>	A boolean.
	<i>connect-error-p</i>	A boolean.
	<i>new-conversation-p</i>	
		A boolean.
Values	<i>result</i>	A boolean.
Description		<p>The function <code>dde-poke*</code> is the same as <code>dde-poke</code>, except that conversations are managed automatically. The function issues a poke transaction to set the value of the item specified by <i>item</i> to the value specified by <i>data</i>. The argument <i>item</i> should be a string, or a symbol. If it is a symbol its print name is used.</p> <p>If <i>server</i> names a client service registered with <code>define-dde-client</code>, the registered service name is used as the DDE service name. If <i>server</i> is any other symbol, the print name of the symbol is used as the DDE service name. If <i>server</i> is a string, that string is used as the DDE service name.</p> <p>The <i>topic</i> argument specifies the DDE topic name to be used in the conversation. If it is a symbol, the symbol's print name is used. If it is a string, the string is used.</p> <p>For information on the <i>format</i>, <i>type</i>, and <i>errorp</i> arguments, see <code>dde-poke</code>.</p> <p>If necessary, the function <code>dde-poke*</code> creates a conversation for the duration of the transaction, but if a suitable conversation already exists, the transaction is executed over that conversation. Hence, if several transactions are made with the same <i>service</i> and <i>topic</i>, placing them inside a <code>with-dde-conversation</code> prevents a new conversation being established for each transaction.</p> <p>If <i>new-conversation-p</i> is set to <code>t</code> a new conversation is always established for the transaction. This new conversation is</p>

always automatically disconnected when the transaction is completed.

If *connect-error-p* is `t` (the default value), LispWorks signals an error if a conversation cannot be established. If it is `nil`, `dde-poke*` returns `nil` if a conversation cannot be established. This allows the caller to distinguish between the cases when the server is not running, and when the server is running but the transaction fails.

See also

`dde-item`

`dde-request`

		<i>Function</i>
dde-request		
Summary	Issues a request transaction on a conversation for a specified item.	
Package	<code>win32</code>	
Signature	<code>dde-request conversation item &key format type errorp => result successp</code>	
Arguments	<code>conversation</code> A conversation object. <code>item</code> A string or symbol. <code>format</code> A clipboard format specifier. <code>type</code> A keyword. <code>errorp</code> A boolean.	
Values	<code>result</code> The return value of the transaction. <code>successp</code> A boolean.	
Description	The function <code>dde-request</code> issues a request transaction on <i>conversation</i> for the specified <i>item</i> . The argument <i>item</i> should	

be a string, or a symbol. If it is a symbol its print name is used.

The argument *format* should be one of the following:

- A DDE format specifier, consisting of either a standard clipboard format or a registered clipboard format.
- A string containing either the name of a standard clipboard format (without the `CF_` prefix), or the name of a registered clipboard format.
- A symbol, in which case its print name is taken to specify the clipboard format.
- The keyword `:text`. This is the default value.

The keyword `:text` is treated specially. If supported by the server it uses the `CF_UNICODETEXT` clipboard format, otherwise it used the `CF_TEXT` format.

The default conversation class only supports text formats, unless *type* is specified as `:foreign`. The argument *type* specifies how the response data should be converted to a Lisp object. For text formats, the default value indicates that a Lisp string should be created. The value `:string-list` may be specified for *type* to indicate that the return value should be taken as a tab-separated list of strings; in this case the Lisp return value is a list of strings. The value `:foreign` can be used with any clipboard format. It returns a `clipboard-item` structure, containing a foreign pointer to the data, the data length, and the format identifier.

This function returns two values, *result* and *successp*. If successful, *result* is the return value of the transaction (which may be `nil` in the case of `:string-list`), and *successp* is true to indicate success.

On failure, the result of the function depends on the *errorp* argument. If *errorp* is `t` (the default), the function signals an error. If *errorp* is `nil`, the function returns (`values nil nil`).

See also

[**dde-item**](#)
[**dde-poke**](#)
[**dde-request***](#)

dde-request* *Function*

Summary	Issues a request transaction on an automatically managed conversation for a specified item.	
Package	<code>win32</code>	
Signature	<code>dde-request* service topic item &key format type errorp connect-error-p new-conversation-p => result successp</code>	
Arguments	<code>service</code>	A symbol or string.
	<code>topic</code>	A symbol or string.
	<code>item</code>	A string or symbol.
	<code>format</code>	A clipboard format specifier.
	<code>type</code>	A keyword.
	<code>errorp</code>	A boolean.
	<code>connect-error-p</code>	A boolean.
	 <code>new-conversation-p</code>	
	A boolean.	
Values	<code>result</code>	The return value of the transaction.
Description	The function <code>dde-request*</code> is similar to <code>dde-request</code> , except that conversations are managed automatically. The function issues a request transaction for the specified <code>item</code> . The argument <code>item</code> should be a string, or a symbol. If it is a symbol its print name is used.	

If *server* names a client service registered with `define-dde-client`, the registered service name is used as the DDE service name. If *server* is any other symbol, the print name of the symbol is used as the DDE service name. If *server* is a string, that string is used as the DDE service name.

The *topic* argument specifies the DDE topic name to be used in the conversation. If it is a symbol, the symbol's print name is used. If it is a string, the string is used.

For information on the *format*, *type*, and *errorp* arguments see `dde-request`.

If necessary, the function `dde-request*` will create a conversation for the duration of the transaction, but if a suitable conversation already exists, the transaction will be executed over that conversation. Hence, if several transactions will be made with the same *service* and *topic*, placing them inside a `with-dde-conversation` prevents a new conversation being established for each transaction.

If *new-conversation-p* is set to `t` a new conversation is always established for the transaction. This new conversation is always automatically disconnected when the transaction is completed.

If *connect-error-p* is `t` (the default value), then LispWorks signals an error if a conversation cannot be established. If it is `nil`, `dde-request*` returns `nil` if a conversation cannot be established. This allows the caller to distinguish between the cases when the server is not running, and when the server is running but the transaction fails.

See also

`dde-item`

`dde-poke`

`dde-request`

	<i>Macro</i>						
Summary	Registers a client service.						
Package	<code>win32</code>						
Signature	<code>define-dde-client name &key service class => name</code>						
Arguments	<table><tr><td><code>name</code></td><td>A symbol.</td></tr><tr><td><code>service</code></td><td>A string.</td></tr><tr><td><code>class</code></td><td>A subclass of <code>dde-client-conversation</code>.</td></tr></table>	<code>name</code>	A symbol.	<code>service</code>	A string.	<code>class</code>	A subclass of <code>dde-client-conversation</code> .
<code>name</code>	A symbol.						
<code>service</code>	A string.						
<code>class</code>	A subclass of <code>dde-client-conversation</code> .						
Values	<table><tr><td><code>name</code></td><td>A symbol.</td></tr></table>	<code>name</code>	A symbol.				
<code>name</code>	A symbol.						
Description	<p>The macro <code>define-dde-client</code> defines a mapping from the symbol <code>name</code> to the DDE service name with which to establish a conversation, and the conversation class to use for this conversation. The argument <code>service</code> is a string which names the DDE service. It defaults to the print-name of <code>name</code>. The argument <code>class</code> is a subclass of <code>dde-client-conversation</code> which is used for all conversations with this service. It defaults to <code>dde-client-conversation</code>. Specifying a subclass allows various aspects of the behavior of the conversation to be specialized.</p> <p>Note that it is generally not necessary to register client services unless a specialized conversation type is required. However, it is sometimes convenient to register a client service in order to allow the service name to be changed in the future.</p> <p>If the macro executes successfully, the <code>name</code> of the DDE service is returned.</p>						
See also	<p><code>dde-connect</code></p> <p><code>dde-disconnect</code></p> <p><code>with-dde-conversation</code></p>						

	<i>Macro</i>
Summary	Dynamically binds a conversation to a server across a given body of code.
Package	<code>win32</code>
Signature	<code>with-dde-conversation (conv service topic &key errorp new-conversation-p) &body body => result</code>
Arguments	<p><code>conv</code> A conversation object.</p> <p><code>service</code> A symbol or string.</p> <p><code>topic</code> A symbol or string.</p> <p><code>errorp</code> A boolean.</p> <p><code>new-conversation-p</code> A boolean.</p> <p><code>body</code> A list of Lisp forms.</p>
Values	<code>result</code> A boolean.
Description	<p>The macro <code>with-dde-conversation</code> dynamically binds a conversation with a server across the scope of a body of code specified by <code>body</code>. The argument <code>conv</code> is bound to a conversation with the server specified by <code>service</code>, and the topic specified by <code>topic</code>.</p> <p>If <code>server</code> names a client service registered with <code>define-dde-client</code>, the registered service name is used as the DDE service name. If <code>server</code> is any other symbol, the print name of the symbol is used as the DDE service name. If <code>server</code> is a string, that string is used as the DDE service name.</p> <p>The <code>topic</code> argument specifies the DDE topic name to be used in the conversation. If it is a symbol, the symbol's print name is used. If it is a string, the string is used.</p>

An existing conversation may be used, if available, unless *new-conversation-p* is true, in which case a new conversation is always created.

If a new conversation is created, it is disconnected after *body* has executed as an implicit program.

If a conversation cannot be established, the result returned by the function depends on the value of *errorp*. If *errorp* is `t` (the default value), then LispWorks signals an error. If *errorp* is `nil`, the body is not executed, and `nil` is returned.

See also `define-dde-client`

17.4 DDE server interface reference entries

		<i>Generic Function</i>
	dde-server-poke	
Summary		Called when a poke transaction is received.
Package		<code>win32</code>
Signature		<code>dde-server-poke server topic item data &key format &allow-other-keys => successp</code>
Arguments	<code>server</code>	A server object.
	<code>topic</code>	A topic object.
	<code>item</code>	A string.
	<code>data</code>	A string.
	<code>format</code>	A keyword.
Values	<code>successp</code>	A boolean.
Description		The generic function <code>dde-server-poke</code> is called in response to a poke transaction. A method specializing on the classes of

server and *topic* should poke the data given by *data* into the item specified by *item*.

The keyword *format* indicates the format in which the item is being requested. By default, only text transfers are supported (and the *format* argument will have the value `:text`).

The set of supported formats may be extended in future releases, so applications should always check the value of the *format* parameter and reject transactions which use formats not supported by the application.

If the poke transaction is successful, `non-nil` should be returned, and `nil` should be returned for failure.

See also

`dde-poke`
`dde-request`
`dde-server-request`

dde-server-request

Generic Function

Summary Called when a request transaction is received.

Package `win32`

Signature `dde-server-request server topic item &key format &allow-other-keys => data`

Arguments `server` A server object.

`topic` A topic object.

`item` A string.

`format` A keyword.

Values `data` The returned data.

Description	<p>The generic function <code>dde-server-request</code> is called in response to a request transaction. A method specializing on the classes of <code>server</code> and <code>topic</code> should return the data in <code>item</code>. The expected format of the data is given by <code>format</code>, which defaults to <code>:text</code>. The set of supported formats may be extended in future releases, so applications should always check the value of the <code>format</code> parameter and reject transactions which use formats not supported by the application.</p> <p>If the request fails, <code>nil</code> should be returned.</p>
-------------	--

See also	dde-poke dde-request dde-server-poke
----------	--

	<i>Generic Function</i>				
Summary	Called whenever a client attempts to connect to a server with a given topic.				
Package	<code>win32</code>				
Signature	<code>dde-server-topic server topic-name => topic</code>				
Arguments	<table> <tr> <td><code>server</code></td> <td>A server.</td> </tr> <tr> <td><code>topic-name</code></td> <td>A string.</td> </tr> </table>	<code>server</code>	A server.	<code>topic-name</code>	A string.
<code>server</code>	A server.				
<code>topic-name</code>	A string.				
Values	<code>topic</code> A topic.				
Description	The generic function <code>dde-server-topic</code> is called whenever a client attempts to make a connection to the server. The argument <code>topic-name</code> is a string identifying a topic. If the server recognizes the topic, a method specializing on the server should return an instance of one of the server's topic classes.				

If the server does not recognize the topic, the method should return `nil`.

See also `dde-server-topics`
`dde-topic-items`

dde-server-topics *Generic Function*

Summary Returns a list of the available general topics on a given server.

Package `win32`

Signature `dde-server-topics server => topic-list`

Arguments `server` A server object.

Values `topic-list` A list of strings.

Description The generic function `dde-server-topics` returns a list of the available general topics on a given server. A suitable method specializing on the server class should be defined. Dispatching topics (see `define-dde-dispatch-topic`) should not be returned, as they are handled automatically by LispWorks. If you do not provide a `dde-server-topics` method, the default method returns `:unknown`, which prevents the DDE server from responding to the topics request.

Generally only one canonical name should be returned for each topic, even though the server may recognize several alternative forms of name for a topic. For example, if an application implements a topic for each open file, the topics `foo`, `foo.doc` and `c:\foo.doc` may all be acceptable strings for referring to the same topic; however `dde-server-topics` should return each topic once only.

The application must also provide a method on the `dde-server-topic` generic function.

See also [dde-server-topic](#)
[dde-topic-items](#)

dde-topic-items *Generic Function*

Summary Returns the valid items in a topic.

Package win32

Signature **dde-topic-items server topic => item-strings**

Values *item-strings* A list of strings.

Description The generic function `dde-topic-items` returns a list of strings corresponding to the valid items in the topic. A method specializing on a server and topic should be defined.

If it is not practical to return a list of the items (for example, if the list is potentially infinite), the generic function returns `:unknown`.

See also [dde-server-topic](#)
[dde-server-topics](#)

define-dde-dispatch-topic *Macro*

Summary Defines a dispatch topic.

Package win32

Signature	<code>define-dde-dispatch-topic name &key server topic-name => name</code>	
Arguments	name	A symbol.
	server	A server class.
	topic-name	A string.
Values	name	A symbol.
Description	<p>The macro <code>define-dde-dispatch-topic</code> defines a dispatching topic. A dispatching topic is a topic which has a fixed name and always exists. Dispatching topics provide dispatching capabilities, whereby appropriate application-supplied code is executed for each supported transaction. Note that the server implementation also provides some dispatching capabilities.</p> <p>The name of the dispatching topic object is specified by <i>name</i>.</p> <p>The topic is identified by the string <i>topic-name</i>.</p> <p>The class of the server to attach the topic to is given by <i>server</i>.</p> <p>The macro <code>define-dde-dispatch-topic</code> returns the name of the dispatching topic, <i>name</i>.</p> <p>Use <code>define-dde-server-function</code> with the <code>:topic</code> option to define items for a dispatch topic.</p>	
Example:	<pre>(define-dde-dispatch-topic topic1 :server demo-server) (define-dde-server-function (item1 :topic topic1) :request () ..handle topic1.item1 request..)</pre>	
See also	dde-server-topic dde-server-topics define-dde-server-function	

define-dde-server	<i>Macro</i>
Summary	Defines a class for a Lisp DDE server.
Package	<code>win32</code>
Signature	$\text{define-dde-server } \textit{class-name} \textit{service-name} \Rightarrow \textit{class-name}$ $\text{define-dde-server } \textit{class-name} \textit{superclasses} \textit{slot-specs} \textit{options} \Rightarrow \textit{class-name}$
Arguments	<p><code>class-name</code> A class name.</p> <p><code>service-name</code> A string.</p> <p><code>superclasses</code> A list of superclasses.</p> <p><code>slot-specs</code> The specifications for the class' slots.</p> <p><code>options</code> A keyword option.</p>
Values	<code>class-name</code> A class name.
Description	<p>The macro <code>define-dde-server</code> defines a class for a Lisp DDE server. The class inherits from <code>dde-server</code>.</p> <p>The long form of the macro is similar to <code>defclass</code>, but with one extra option, <code>:service</code>, which is used to specify the service name string to which this server will respond.</p> <p>The short form is provided to handle the common simple case; <code>class-name</code> is the name of the Lisp class to be defined, and <code>service-name</code> is the service name string to which this server will respond.</p>
Example	<p>The first example uses the short version of <code>define-dde-server</code> to define a class, called <code>lisp-server</code>, which has the service name “LISP”.</p> <pre>(define-dde-server lisp-server "LISP")</pre>

The second example shows how to use the long form of the macro to define the same class, and illustrates the use of the *superclasses* and *options* arguments.

```
(define-dde-server lisp-server (dde-server)
  ())
  (:service "LISP"))
```

See also **dde-server-topic**
dde-server-topics
dde-topic-items

define-dde-server-function

Macro

Summary Defines a server function that is called when a specific transaction occurs.

Package **win32**

Signature **define-dde-server-function** *name-and-options transaction*
(binding) form* => name*

name-and-options ::= name | (name [[option]])

transaction ::= :request | :poke | :execute

option ::= :server server |
:topic-class topic-class |
:topic topic |
:item item |
:format format |
:command command |
:result-type result-type |
:advisepr advisepr

binding ::= var-binding | execute-arg-binding

var-binding ::= (var :server) |
(var :topic) |
(var :data [data-type]) |
(var :format)

execute-arg-binding ::= **var** | (**var** **type-spec**)

Arguments	<i>name</i>	A symbol.
	<i>transaction</i>	A keyword.
	<i>server</i>	A server object.
	<i>topic-class</i>	A topic class.
	<i>topic</i>	A symbol naming a dispatch topic.
	<i>item</i>	A string.
	<i>format</i>	A keyword.
	<i>command</i>	A string.
	<i>result-type</i>	A data type.
	<i>advisep</i>	A boolean.
	<i>var</i>	A variable.
	<i>data-type</i>	A data type.
	<i>type-spec</i>	A data type.
	<i>form</i>	A Lisp form.
Values	<i>name</i>	A symbol.
Description	<p>The macro define-dde-server-function is used to define a server function, called <i>name</i>, which is called when a specific transaction occurs. The defined function may either be attached to a server class (using the dispatching capabilities built into the server implementation) or to a named dispatch topic.</p> <ul style="list-style-type: none"> • To attach the definition to a server, :server should be used to specify the server class. :topic-class may be used to specify the topic-class for which this definition should be used. It can be a symbol which names a topic-class, or t (meaning All topics, this is the default for execute transactions), or :system (The System topic), or :non-system (any 	

topic except the System topic). In the case of execute transactions only, `:topic-class` defaults to `t`; in all other cases, it must be specified. Typically, execute transactions ignore the topic of the conversation. Alternatively, you may choose to only support execute transactions in the system topic.

- A server function may instead be attached to a particular instance of `dde-dispatch-topic`, previously defined by `define-dde-dispatch-topic`. This is the main use of dispatching topics. In this case `:topic` should be provided with a symbol that names a dispatching topic. The function is installed on that topic, and only applies to that topic.

In the case of a request or poke transaction, `item` is a string defining the item name for which this definition should be invoked. It defaults to the capitalized print-name of `name`, with hyphens removed.

For request transactions, the `:format` option is used to specify the format understood. It defaults to `:text`. It can be specified as `:all`, in which case the `:format` binding may be used to determine the actual format requested (see below).

In the case of an execute transaction, `command` is a string specifying the name of the command for which this definition should be invoked. It defaults to the capitalized print-name of `name`, with hyphens removed.

The `execute-arg-bindings` are only used with execute transactions. They specify the arguments expected. `type-spec` should be one of `t`, `string`, `number`, `integer` or `float`. If not specified, `t` is assumed.

The `var-bindings` may appear anywhere in the binding list, and in any order. Binding variables to `:server` and `:topic` is useful with all transaction types. A `:server` binding causes the variable to be bound to the server object, whereas a `:topic` binding causes the variable to be bound to the topic

object. This allows the server and/or the topic to be referred to in the body of the function.

A `:format` binding can only be used with request and poke transactions, where an *option* of `:format :all` has been specified. It causes the variable specified by `var` to be bound to the format of data requested or supplied. The body of the defined function should fail the transaction if it does not support the requested format.

A `:data` binding can only be used with poke transactions. It binds a variable to the data to be poked. For text transfers, the data variable is normally bound to a string. However, if `datatype` is specified as `:string-list`, the data in the transaction is interpreted as a tab-separated list of strings, and the data variable is bound to a list of strings.

For execute and poke transactions, the body of the defined function is expected to return `t` for success and `nil` for failure.

For request transactions, the body of the defined function is normally expected to return a result value, or `nil` for failure.

The `result-type` option may only be specified for request transactions. If it is specified as `:string-list`, then for text requests the body is expected to return a list of strings, which are used to create a tab-separated list to be returned to the client.

Sometimes, it may be necessary to support returning `nil` to mean the empty list, rather than failure. In this case, the `result-type` can be specified as `(:string-list t)`. The body is then expected to return two values: a list of strings, and a flag indicating success.

In the case of execute transactions, the command name and arguments are unmarshalled by the default argument unmarshalling. This is compatible with the default argument unmarshalling described under `dde-execute-command`. The execute string is expected to be of the following syntax:

```
[command1(arg1,arg2,...)][command2(arg1,arg2,...)]....]
```

Note that multiple commands may be packed into a single execute transaction. However, `dde-execute-command` does not currently generate such strings.

See also `dde-execute-command`

`define-dde-client`

`define-dde-dispatch-topic`

`define-dde-server`

start-dde-server *Function*

Summary Creates and starts an instance of a DDE server.

Package `win32`

Signature `start-dde-server name => server`

Arguments `name` A DDE server class

Values `server` A server object

Description The function `start-dde-server` creates an instance of a server of the class specified by `name` which then starts accepting transactions. If successful the function returns the server, otherwise `nil` is returned.

You need to call `start-dde-server` in a thread that will process Windows messages. This can either be done by using `capi:execute-with-interface` to run it in the thread of an application's main window (if there is one) or by running it in a dedicated thread as in the example. DDE callbacks will happen in this thread.

Example

```
(mp:process-run-function
  "DDE Server"
  ()
  #'(lambda ()
      (win32:start-dde-server 'lispworks-dde-server)
      (loop
        (mp:wait-processing-events
         nil
         :wait-reason "DDE Request Loop"))))
```

See also

define-dde-server

18

Dynamic library C functions

This chapter describes the C functions available in a LispWorks dynamic library, that is a library created by passing *dll-exports* or *dll-added-files* to `save-image` or `deliver`.

For an overview of this functionality with examples of use, see the section “LispWorks as a dynamic library” in the *LispWorks User Guide*.

Note: this chapter applies only to 32-bit LispWorks on Microsoft Windows, Intel Macintosh, Linux and FreeBSD, and 64-bit LispWorks on Windows, Intel Macintosh and Linux.

InitLispWorks	C function
Summary	Provides control over the initialization of a LispWorks dynamic library.
Signature	On Windows: <code>int __stdcall InitLispWorks (int MilliTimeOut, void *BaseAddress, size_t ReserveSize)</code> On Linux/Macintosh/FreeBSD:

```
int InitLispWorks (int MilliTimeOut, void *BaseAddress,
size_t ReserveSize)
```

Description The C function `InitLispWorks` allows you to relocate a LispWorks dynamic library if this is necessary, and offers control of the initialization process.

A LispWorks dynamic library is automatically initialized by any call to its exported symbols, so in most cases there is no need to call `InitLispWorks`. It is however necessary when you need to relocate LispWorks or when you need finer control over the initialization process.

For more information about relocating a LispWorks dynamic library, see "Startup Relocation" in the *LispWorks User Guide*

MilliTimeOut specifies the time in milliseconds to wait for LispWorks to finish initializing before returning. `InitLispWorks` checks whether the library was initialized and if not initiates initialization. It then waits at most *MilliTimeOut* milliseconds before returning.

BaseAddress specifies the base address for relocation. Can be 0.

ReserveSize specifies the reserve size for relocation. Can be 0.

BaseAddress and *ReserveSize* are interpreted as described in "Startup Relocation" in the *LispWorks User Guide*.

Non-negative return values indicate success:

1 LispWorks was already initialized or in the process of initializing, and finished initializing by the time `InitLispWorks` returned.

0 `InitLispWorks` initialized LispWorks and the initialization finished successfully.

Values in the inclusive range [-1, -99] indicate a timeout:

-1 `InitLispWorks` started initialization and timed out before LispWorks finished mapping itself from the file.

- 2 LispWorks already started initialization, and `InitLispWorks` timed out before LispWorks finished mapping itself from the file.
- 3 `InitLispWorks` started initialization and timed out after LispWorks mapped itself from the file, but before the initialization was complete.
- 4 LispWorks already started initialization, and `InitLispWorks` timed out before after LispWorks mapped itself from the file, but before the initialization was complete.

After `InitLispWorks` times out, the state of LispWorks can be queried by `LispWorksState`.

Lower values indicate failure, as follows:

- 1000 Failure to start a thread to do the initialization.
- 1401 The file seems to be corrupted.
- 1402 Failure to map into memory.
- 1403 Failure to read the LispWorks header from the file.
- 1406 Bad base address.

Additionally, a value *value* in the inclusive range [-1400, -1001] on Linux/Macintosh/FreeBSD platforms indicates an error in a system call. Calculate the errno number by -1001 - *value*.

Note: If LispWorks is already initialized or in the process of being initialized, `InitLispWorks` does not initiate the process of initialization. Therefore the arguments to `InitLispWorks` have no effect if LispWorks was already initialized when it is called. On Microsoft Windows, the default behavior is to initialize a LispWorks dynamic library automatically during loading, so this needs to be disabled to use `InitLispWorks`

effectively. Disable automatic initialization of a library as described for `deliver` and `save-image`.

Note: Once `QuitLispWorks` has returned 0, LispWorks can be initialized again. It is possible to quit and restart LispWorks several times, at the same address or at a different address.

Note: On Linux/Macintosh/FreeBSD you can create wrappers to the C functions described in this chapter from your application by writing them in C and adding them to the dynamic library using *dll-added-files* in `deliver` and `save-image`. Such wrappers can be used to add calls to `InitLispWorks` before actually calling into Lisp.

`InitLispWorks` is defined in each LispWorks dynamic library. For information about creating a LispWorks dynamic library, see `deliver` and `save-image`. For an overview of LispWorks as a dynamic library, see the section “LispWorks as a dynamic library” in the *LispWorks User Guide*.

See also

`deliver`
`LispWorksState`
`save-image`
`QuitLispWorks`

LispWorksDlsym

C function

Summary Returns the address of a foreign callable.

Signature On Windows:

```
void __stdcall *LispWorksDlsym (const char * name)
```

On Linux/Macintosh/FreeBSD:

```
void *LispWorksDlsym (const char * name)
```

Description	<p>The C function <code>LispWorksD1sym</code> returns the address of a foreign callable <i>name</i> which is defined in Lisp using <code>fli:define-foreign-callable</code>.</p> <p><code>LispWorkD1sym</code> first checks if the LispWorks dynamic library finished initializing, and if not uses <code>InitLispWorks</code> to initialize it (with <i>MilliTimeOut</i> 200). If this fails <code>LispWorkD1sym</code> returns NULL. When the LispWorks dynamic library is initialized, <code>LispWorksD1sym</code> returns the address of <i>name</i>, or NULL if it is not defined.</p> <p><code>LispWorksD1sym</code> is defined in each LispWorks dynamic library. For information about creating a LispWorks dynamic library, see <code>deliver</code> and <code>save-image</code>. For an overview of LispWorks as a dynamic library, see the section “LispWorks as a dynamic library” in the <i>LispWorks User Guide</i>.</p>
See also	<code>InitLispWorks</code>

LispWorksState *C function*

Summary	Returns the state of a LispWorks dynamic library.
Signature	<p>On Windows:</p> <pre>int __stdcall LispWorksState (int MilliTimeOut)</pre> <p>On Linux/Macintosh/FreeBSD:</p> <pre>int LispWorksState (int MilliTimeOut)</pre>
Description	<p>The C function <code>LispWorksState</code> returns the state of a LispWorks dynamic library.</p> <p><i>MilliTimeOut</i> specifies the time to wait in milliseconds if LispWorks is in the process of initialization.</p> <p>If LispWorks has not been initialized, or has been quit by <code>QuitLispWorks</code>, <code>LispWorksState</code> returns -100. Otherwise, it returns the same values as <code>InitLispWorks</code>. In particular, if</p>

LispWorks is already properly initialized it returns 1, and if LispWorks is still in the process of initialization it returns -2 or -4. Otherwise it returns a more negative number indicating an error.

`LispWorksState` is defined in each LispWorks dynamic library. For information about creating a LispWorks dynamic library, see `deliver` and `save-image`. For an overview of LispWorks as a dynamic library, see the section “LispWorks as a dynamic library in the *LispWorks User Guide*.

See also [InitLispWorks](#)
[QuitLispWorks](#)

SimpleInitLispWorks *C function*

Summary Initializes a LispWorks dynamic library.

Signature On Windows:

```
int __stdcall SimpleInitLispWorks (void)
```

On Linux/Macintosh/FreeBSD:

```
int SimpleInitLispWorks (void)
```

Description The C function `SimpleInitLispWorks` calls `InitLispWorks(0,0,0)` and returns the value of that call. `SimpleInitLispWorks` is defined in each LispWorks dynamic library. For information about creating a LispWorks dynamic library, see `deliver` and `save-image`. For an overview of LispWorks as a dynamic library, see the section “LispWorks as a dynamic library in the *LispWorks User Guide*.

See also [InitLispWorks](#)

QuitLispWorks

C function

Summary Allows a LispWorks dynamic library to be unloaded.

Signature On Windows:

```
int __stdcall QuitLispWorks(int Force, int MilliTimeOut)
```

On Linux/Macintosh/FreeBSD:

```
int QuitLispWorks(int Force, int MilliTimeOut)
```

Description The C function `quitLispWorks` allows a LispWorks dynamic library to be unloaded. You should make a LispWorks dynamic library 'quit' by calling `quitLispWorks` before unloading the library. This call causes LispWorks to cleanup everything it uses, in particular the memory and threads.

In general, `quitLispWorks` should be called only when the LispWorks dynamic library is idle. That is, when there is no callback into the library that has not returned, and there are no processes that has started by a callback. All callbacks should return, and any processes should be killed before calling `quitLispWorks`.

Force should be 0 or 1. It specifies whether to force quitting even if LispWorks is still executing something.

MilliTimeOut specifies how long to wait for LispWorks to complete the cleanup.

If LispWorks is idle, `quitLispWorks` signals it to quit, and waits *MilliTimeOut* milliseconds for it to finish the cleanup. If LispWorks finished cleanup, `quitLispWorks` return 0 (SUCCESS). If the cleanup is not finished it returns -2 (TIMEOUT).

If LispWorks is not idle, that is there are still some active callbacks or there are processes that have started by a callback (even if they are inside `process-wait`), `quitLispWorks` checks the value of *Force*. If *Force* is 0, `quitLispWorks` returns -1 (NOT_IDLE). If *Force* is 1, `quitLispWorks` signals it to quit and behaves as if LispWorks is idle, described above.

`QuitLispWorks` can be called repeatedly to check if LispWorks finished the cleanup.

When `QuitLispWorks` returns NOT_IDLE, it has done nothing, and the LispWorks dynamic library can be used for further callbacks. Once `QuitLispWorks` returns any other value, callbacks into the dynamic library will result in undefined behavior.

Once `QuitLispWorks` returns SUCCESS, it is safe to unload the dynamic library. Unloading it before `QuitLispWorks` returns SUCCESS gives undefined results.

Once `QuitLispWorks` returns SUCCESS, LispWorks can be initialized again. Calling any exported function (supplied to `save-image` or `deliver` in `dll-exports`) or any of `InitLispWorks`, `SimpleInitLispWorks` and `LispWorksDlSym` will cause LispWorks to initialize again.

Note: On Linux, Macintosh and FreeBSD it is possible to add calls to `QuitLispWorks` at the right places via *dll-added-files*.

Note: A possible reason for failure to finish the cleanup is that a LispWorks process is stuck inside a foreign call. Dynamic library applications that need to be unloaded should be careful to ensure that they do not get stuck in a foreign function call.

`QuitLispWorks` is defined in each LispWorks dynamic library. For information about creating a LispWorks dynamic library, see `deliver` and `save-image`. For an overview of LispWorks as a dynamic library, see the section “LispWorks as a dynamic library” in the *LispWorks User Guide*.

See also

`deliver`
`dll-quit`
`save-image`

Index

Numerics

`16-bit-string` type 310
`8-bit-string` type 309

A

accessors
 `dde-item` 785
 `dde-item*` 787
 `lob-stream-lob-locator` 524
 `socket-stream-socket` 45
 `sql-database-data-error` 582
 `sql-error-database-message` 582
 `sql-error-secondary-error-id` 582
 `storage-exhausted-gen-num` 745
 `storage-exhausted-size` 745
 `storage-exhausted-static` 745
 `storage-exhausted-type` 745
 `stream-read-timeout` 46
 `stream-write-timeout` 46
 `*active-finders*` variable 151
`add-special-free-action` function 193
`add-sql-stream` function 479
`add-symbol-profiler` function 194
allocation of stacks 650
`allocation-in-gen-num` macro 195
`appendf` macro 310
`apply-with-allocation-in-gen-num`
 function 637
`apropos` function 63
`apropos-list` function 64

argument list 354
arguments
 lisp function 354
`at-location` macro 152
`attach-ssl` function 21
`attribute-type` function 480
`augmented-string` type 638
`augmented-string-p` function 639
`avoid-gc` function 196

B

`base-char` type 311
`base-character` type 310
`base-character-p` function 311
`base-char-code-limit` constant 312
`base-char-p` function 311
`base-string` type 65
`base-string-p` function 312
`:base-table` class option 502
`*binary-file-type*` variable 72
`*binary-file-types*` variable 69, 72
`binds-who` function 197
`block-promotion` macro 197
`break-new-instances-on-access`
 function 1
`break-on-access` function 2
`break-on-unresolved-functions`
 function 303
`browser` 313
`*browser-location*` variable 313
browsing documentation 313
`buffered-stream` class 603
`building-universal-intermediate-p`
 function 199

C

C functions

- `InitLispWorks` 811
- `LispWorksDlsym` 814
- `LispWorksState` 815
- `QuitLispWorks` 817
- `SimpleInitLispWorks` 816
- `cache-table-queries` function 482
- `*cache-table-queries-default*` variable 483
- `call-next-advice` function 313
- `calls-who` function 200
- `call-system` function 639
- `call-system-showing-output` function 641
- `canonicalize-dspec` function 153
- `cd` macro 200
- `cdr-assoc` function 644
- `change-directory` function 201
- `change-process-priority` function 403
- `char-external-code` function 185
- `check-fragmentation` function 202
- `*check-network-server*` variable 645
- class options
 - parsing of 8
 - `:base-table` 502
 - `:extra-initargs` 3, 5, 78
 - `:optimize-slot-access` 14, 78
- classes
 - `buffered-stream` 603
 - `funcallable-standard-object` 6
 - `fundamental-binary-input-stream` 605
 - `fundamental-binary-output-stream` 606
 - `fundamental-binary-stream` 606
 - `fundamental-character-input-stream` 607
 - `fundamental-character-output-stream` 607
 - `fundamental-character-stream` 608
 - `fundamental-input-stream` 609
 - `fundamental-output-stream` 610
 - `fundamental-stream` 610
 - `lob-stream` 523
 - `socket-error` 45
 - `socket-stream` 45
 - `ssl-closed` 53
 - `ssl-condition` 53
 - `ssl-error` 54
 - `ssl-failure` 54
 - `ssl-x509-lookup` 55
- `standard-db-object` 591
- `storage-exhausted` 745
- `class-extra-initargs` generic function 3
- `clean-down` function 203
- `clean-generation-0` function 204
- `close` generic function 65
- `close-registry-key` function 754
- `close-serial-port` function 471
- Cocoa application 412
- Cocoa event loop 412
- `coerce` function 66
- `coerce-to-gesture-spec` function 645
- `*coff-loading-verbose*` 308
- `collect-generation-2` function 205
- `collect-highest-generation` function 205
- `collect-registry-subkeys` function 755
- `collect-registry-values` function 756
- command line arguments 698
- command line processing 698
- commands
 - listener 651
 - top level 651
- `commit` function 483
- `compile` function 67
- `compile-file` function 68
- `compile-file-if-needed` function 207
- compiler explanations 74
- compiler help 74
- `*compiler-break-on-error*` variable 206
- `compile-system` function 315
- `compute-class-potential-initargs` generic function 4
- `compute-discriminating-function` generic function 6
- `concatenate` function 72
- `concatenate-system` function 317
- conditions
 - `external-format-error` 188
 - `file-encoding-resolution-error` 658
 - `sql-connection-error` 581
 - `sql-database-data-error` 582
 - `sql-database-error` 582
 - `sql-fatal-error` 585
 - `sql-temporary-error` 590
 - `sql-timeout-error` 590
 - `sql-user-error` 591
- `connect` function 484
- `connected-databases` function 491
- `*connect-if-exists*` variable 491
- console 255

console application 255
 constants
 base-char-code-limit 312
 copy-preferences-from-older-version
 function 647
 copy-to-weak-simple-vector
 function 208
 count-gen-num-allocation function 649
 create-index function 492
 create-registry-key function 758
 create-simple-process function 404
 create-table function 493
 create-universal-binary function 209
 create-view function 494
 create-view-from-class function 496
 :ctx-configure-callback initarg 45
 current-pathname function 318
 current-process special variable 406
 current-stack-length function 210

D

database-name function 496
 dde-advise-start function 770
 dde-advise-start* function 772
 dde-advise-stop function 774
 dde-advise-stop* function 775
 dde-client-advise-data generic
 function 776
 dde-connect macro 777
 dde-disconnect macro 778
 dde-execute function 778
 dde-execute* function 779
 dde-execute-command function 780
 dde-execute-command* function 781
 dde-execute-string function 783
 dde-execute-string* function 784
 dde-item accessor 785
 dde-item* accessor 787
 dde-poke function 789
 dde-poke* function 790
 dde-request function 792
 dde-request* function 794
 dde-server-poke generic function 798
 dde-server-request generic function 799
 dde-server-topic generic function 800
 dde-server-topics generic function 801
 dde-topic-items generic function 802
 debug-other-process function 407
 debug-print-length variable 137
 debug-print-level variable 138
 declaim macro 73
 declaration

alias 74
 :explain 74
 invisible-frame 74
 lambda-list 74
 values 74
 declare :explain 74
 declare special form 74
 decode-external-string function 186
 def macro 154
 defadvice macro 320
 default directory 233
 default file directory 233
 default-action-list-sort-time
 variable 323
 default-character-element-type
 parameter 324
 default-database variable 497
 default-database-type variable 497
 default-eol-style function 650
 default-libraries variable 307
 default-package-use-list
 variable 211
 default-process-priority
 variable 407
 default-profiler-collapse
 variable 211
 default-profiler-cutoff
 variable 212
 default-profiler-limit variable 212
 default-profiler-sort variable 213
 default-simple-process-priority
 variable 407
 default-stack-group-list-length
 variable 650
 default-update-objects-max-len
 variable 498
 defclass macro 77
 define-action macro 324
 define-action-list macro 326
 define-dde-client macro 796
 define-dde-dispatch-topic macro 802
 define-dde-server macro 804
 define-dde-server-function macro 805
 define-dspec-alias macro 155
 define-dspec-class macro 156
 define-form-parser macro 159
 define-top-loop-command macro 651
 defpackage macro 81
 defparser macro 467
 defsystem macro 328
 defsystem-verbose variable 333
 deftransform macro 133

def-view-class macro 498
delete-advice macro 213
delete-directory function 333
delete-instance-records generic
 function 505, 518
delete-records function 506
delete-registry-key function 759
delete-sql-stream function 507
deliver function 334
deliverable
 filename 364
 pathname 364
describe function 82
describe-length variable 335
describe-level variable 335
describe-print-length variable 337
describe-print-level variable 337
destroy-ssl function 23
destroy-ssl-ctx function 23
detach-ssl function 24
detect-eol-style function 653
detect-japanese-encoding-in-file
 function 654
detect-unicode-bom function 655
:direction initarg 45, 524, 603
directory function 83
directory-link-transparency special
 variable 83
directory-link-transparency
 variable 656
disable-sql-reader-syntax
 function 508
disable-trace variable 215
disassemble function 87
discard-source-info function 169
disconnect function 508
dismiss-splash-screen function 751
DLL
 filename 364
 pathname 364
dll-quit function 338
DNS 26
documentation generic function 88
domain 26
do-nothing function 341
do-query macro 509
do-rand-seed function 25
DOS command
 call-system 640
 call-system-showing-output 641
 open-pipe 710
dotted-list-length function 340
dotted-list-p function 341
double-float type 89
drop-index function 511
drop-table function 511
drop-view function 512
drop-view-from-class function 512
dspec-class function 163
dspec-classes variable 163
dspec-defined-p function 164
dspec-definition-locations
 function 164
dspec-equal function 165
dspec-name function 166
dspec-primary-name function 166
dspec-progenitor function 167
dspec-subclass-p function 168
dspec-undefiner function 168
dump-form function 215
dump-forms-to-file function 216

E

:ef-spec initarg 658
:element-type initarg 45, 603
enable-sql-reader-syntax function 513
encode-lisp-string function 187
enlarge-generation function 217
enlarge-static function 218
ensure-process-cleanup function 408
ensure-ssl function 25
enter-debugger-directly
 variable 342
enum-registry-value function 760
environment-variable function 342
errno-value function 344
error output 704
example-compile-file function 345
example-file function 344
example-load-binary-file function 346
executable
 filename 364
 pathname 364
execute-actions macro 346
execute-command function 514
expand-generation-1 function 219
extend-current-stack function 220
extended-char type 348
extended-character type 348
extended-character-p function 349
extended-char-p function 349
extended-spaces variable 398, 657
extended-time function 221
external-format-error condition 188

```

external-format-foreign-type
  function 188
*external-formats* variable 350
external-format-type function 189
:extra-initargs class option 3, 5, 78

F
false function 351
*features* variable 89
file-directory-p function 351
*file-encoding-detection-algorithm*
  variable 657
file-encoding-resolution-error
  condition 658
*file-eol-style-detection-algo-
  rithm* variable 659
filename of deliverable 364
filename of DLL 364
filename of dynamic library 364
filename of executable 364
filename of lisp image 364
*filename-pattern-encoding-matches*
  variable 659
file-stat-blocks function 669
file-stat-device function 669
file-stat-device-type function 670
file-stat-group-id function 669
file-stat-inode function 669
file-stat-last-access function 669
file-stat-last-change function 669
file-stat-last-modify function 670
file-stat-links function 670
file-stat-mode function 669
file-stat-owner-id function 669
file-stat-size function 669
file-string function 223
file-writable-p function 224
find-database function 514
find-dspec-locations function 169
find-encoding-option function 660
find-external-char function 189
find-filename-pattern-encoding-
  match function 660
find-name-locations function 170
find-object-size function 224
find-process-from-name function 409
find-regexp-in-string function 352
finish-heavy-allocation function 225
flag-not-special-free-action
  function 226
flag-special-free-action function 227
foreign types

p-oci-env 541
p-oci-file 558
p-oci-lob-locator 558
p-oci-svc-ctx 564
ssl-cipher-pointer 52
ssl-cipher-pointer-stack 52
ssl-ctx-pointer 53
ssl-pointer 55
foreign-symbol-address function 304
:free-lob-locator-on-close initarg 524
funcallable-standard-object class 6
function-lambda-list function 354
functions
  add-special-free-action 193
  add-sql-stream 479
  add-symbol-profiler 194
  apply-with-allocation-in-gen-
    num 637
  apropos 63
  apropos-list 64
  attach-ssl 21
  attribute-type 480
  augmented-string-p 639
  avoid-gc 196
  base-character-p 311
  base-char-p 311
  base-string-p 312
  binds-who 197
  break-new-instances-on-access 1
  break-on-access 2
  break-on-unresolved 303
  building-universal-intermediate-
    p 199
  cache-table-queries 482
  call-next-advice 313
  calls-who 200
  call-system 639
  call-system-showing-output 641
  canonicalize-dspec 153
  cdr-assoc 644
  change-directory 201
  change-process-priority 403
  char-external-code 185
  check-fragmentation 202
  clean-down 203
  clean-generation-0 204
  close-registry-key 754
  close-serial-port 471
  coerce 66
  coerce-to-gesture-spec 645
  collect-generation-2 205
  collect-highest-generation 205

```

collect-registry-subkeys 755
collect-registry-values 756
commit 483
compile 67
compile-file 68
compile-file-if-needed 207
compile-system 315
concatenate 72
concatenate-system 317
connect 484
connected-databases 491
copy-preferences-from-older-
version 647
copy-to-weak-simple-vector 208
count-gen-num-allocation 649
create-index 492
create-registry-key 758
create-simple-process 404
create-table 493
create-universal-binary 209
create-view 494
create-view-from-class 496
current-pathname 318
current-stack-length 210
database-name 496
dde-advice-start 770
dde-advice-start* 772
dde-advice-stop 774
dde-advice-stop* 775
dde-execute 778
dde-execute* 779
dde-execute-command 780
dde-execute-command* 781
dde-execute-string 783
dde-execute-string* 784
dde-poke 789
dde-poke* 790
dde-request 792
dde-request* 794
debug-other-process 407
decode-external-string 186
default-eol-style 650
delete-directory 333
delete-records 506
delete-registry-key 759
delete-sql-stream 507
deliver 334
describe 82
destroy-ssl 23
destroy-ssl-ctx 23
detach-ssl 24
detect-eol-style 653
detect-japanese-encoding-in-
file 654
detect-unicode-bom 655
directory 83
disable-sql-reader-syntax 508
disassemble 87
discard-source-info 169
disconnect 508
dismiss-splash-screen 751
dll-quit 338
do-nothing 341
do-rand-seed 25
dotted-list-length 340
dotted-list-p 341
drop-index 511
drop-table 511
drop-view 512
drop-view-from-class 512
dspec-class 163
dspec-defined-p 164
dspec-definition-locations 164
dspec-equal 165
dspec-name 166
dspec-primary-name 166
dspec-progenitor 167
dspec-subclass-p 168
dspec-undefiner 168
dump-form 215
dump-forms-to-file 216
enable-sql-reader-syntax 513
encode-lisp-string 187
enlarge-generation 217
enlarge-static 218
ensure-process-cleanup 408
ensure-ssl 25
enum-registry-value 760
environment-variable 342
errno-value 344
example-compile-file 345
example-file 344
example-load-binary-file 346
execute-command 514
expand-generation-1 219
extend-current-stack 220
extended-character-p 349
extended-char-p 349
extended-time 221
external-format-foreign-type 188
external-format-type 189
false 351
file-directory-p 351
file-stat-blocks 669

file-stat-device 669
 file-stat-device-type 670
 file-stat-group-id 669
 file-stat-inode 669
 file-stat-last-access 669
 file-stat-last-change 669
 file-stat-last-modify 670
 file-stat-links 670
 file-stat-mode 669
 file-stat-owner-id 669
 file-stat-size 669
 file-string 223
 file-writable-p 224
 find-database 514
 find-dspec-locations 169
 find-encoding-option 660
 find-external-char 189
 find-filename-pattern-encoding-match 660
 find-name-locations 170
 find-object-size 224
 find-process-from-name 409
 find-regexp-in-string 352
 finish-heavy-allocation 225
 flag-not-special-free-action 226
 flag-special-free-action 227
 foreign-symbol-address 304
 function-lambda-list 354
 gc-generation 227
 gc-if-needed 231
 gen-num-segments-fragmentation-state 661
 gesture-spec-data 664
 gesture-spec-modifiers 665
 gesture-spec-p 666
 gesture-spec-to-character 668
 get-current-process 409
 get-default-generation 231
 get-file-stat 668
 get-folder-path 670
 get-foreign-symbol 305
 get-form-parser 171
 get-gc-parameters 232
 get-host-entry 26
 get-process 410
 get-serial-port-state 471
 get-socket-address 28
 get-socket-peer-address 29
 get-unix-error 357
 get-user-profile-directory 672
 get-verification-mode 29
 get-working-directory 233
 guess-external-format 673
 hardcopy-system 362
 initialize-database-type 515
 initialize-multiprocessing 411
 insert-records 517
 int32* 676
 int32+ 677
 int32- 677
 int32/ 680
 int32/= 680
 int32< 681
 int32<< 682
 int32<= 682
 int32= 683
 int32> 683
 int32>= 684
 int32>> 685
 int32-1+ 678
 int32-1- 679
 int32-aref 685
 int32-logand 686
 int32-logandc1 687
 int32-logandc2 687
 int32-logbitp 688
 int32-logeqv 689
 int32-logior 690
 int32-lognand 690
 int32-lognor 691
 int32-lognot 692
 int32-logorc1 692
 int32-logorc2 693
 int32-logtest 694
 int32-logxor 694
 int32-minusp 695
 int32-plusp 695
 int32-to-integer 696
 int32-zerop 697
 integer-to-int32 697
 interactive-stream-p 93
 ip-address-string 30
 lisp-image-name 364
 lisp-name-to-foreign-name 306
 list-all-processes 413
 list-attributes 520
 list-attribute-types 519
 list-classes 521
 list-sql-streams 522
 list-tables 523
 load-all-patches 366
 load-data-file 699
 load-logical-pathname-translations 94

load-system 367
 local-dspec-p 172
 locale-file-encoding 700
 locally-disable-sql-reader-
 syntax 525
 locally-enable-sql-reader-
 syntax 525
 lock-name 414
 lock-owner 415
 long-site-name 95
 mailbox-empty-p 416
 mailbox-peek 417
 mailbox-read 417
 mailbox-reader-process 418
 mailbox-send 419
 mailbox-wait-for-event 419
 make-array 97
 make-gesture-spec 701
 make-hash-table 98
 make-lock 421
 make-mailbox 423
 make-mt-random-state 370
 make-named-timer 424
 make-sequence 100
 make-simple-int32-vector 703
 make-ssl-ctx 31
 make-stderr-stream 704
 make-timer 425
 make-typed-aref-vector 704
 make-unregistered-action-list 369
 map 101
 map-all-processes 426
 map-all-processes-backtrace 426
 map-process-backtrace 427
 map-processes 428
 map-query 528
 mark-and-sweep 238
 marking-gc 705
 memory-growth-margin 707
 merge 102
 merge-ef-specs 708
 mt-random 371
 mt-random-state-p 372
 name-defined-dspecs 173
 name-definition-locations 174
 name-only-form-parser 175
 normal-gc 240
 notice-fd 429
 object-address 708
 open 102
 open-pipe 709
 open-registry-key 762
 open-serial-port 469
 openssl-version 35
 open-tcp-stream 32
 open-url 712
 ora-lob-append 531
 ora-lob-assign 532
 ora-lob-char-set-form 532
 ora-lob-char-set-id 533
 ora-lob-close 534
 ora-lob-copy 535
 ora-lob-create-empty 536
 ora-lob-create-temporary 537
 ora-lob-disable-buffering 538
 ora-lob-element-type 539
 ora-lob-enable-buffering 540
 ora-lob-env-handle 540
 ora-lob-erase 541
 ora-lob-file-close 542
 ora-lob-file-close-all 543
 ora-lob-file-exists 543
 ora-lob-file-get-name 544
 ora-lob-file-is-open 545
 ora-lob-file-open 546
 ora-lob-file-set-name 546
 ora-lob-flush-buffer 547
 ora-lob-free 548
 ora-lob-free-temporary 549
 ora-lob-get-buffer 550
 ora-lob-get-chunk-size 552
 ora-lob-get-length 553
 ora-lob-internal-lob-p 554
 ora-lob-is-equal 554
 ora-lob-is-open 555
 ora-lob-is-temporary 555
 ora-lob-load-from-file 556
 ora-lob-lob-locator 557
 ora-lob-locator-is-init 558
 ora-lob-open 559
 ora-lob-read-buffer 560
 ora-lob-read-foreign-buffer 563
 ora-lob-read-into-plain-file 562
 ora-lob-svc-ctx-handle 564
 ora-lob-trim 565
 ora-lob-write-buffer 566, 568
 ora-lob-write-from-plain-file 567
 parse-float 241
 parse-form-dspec 176
 pathname-location 373
 pem-read 36
 pid-exit-status 713
 pointer-from-address 714
 precompile-regexp 374

print-action-lists 375
 print-actions 374
 print-pretty-gesture-spec 715
 print-profile-list 242
 print-query 570
 process-alive-p 429
 process-allow-scheduling 430
 process-arrest-reasons 430
 process-break 431
 process-continue 431
 process-idle-time 432
 process-interrupt 434
 process-kill 435
 process-lock 435
 process-mailbox 436
 process-name 437
 process-p 437
 process-plist 437
 process-priority 438
 process-reset 439
 process-run-function 439
 process-run-reasons 441
 process-run-time 441
 process-send 442
 process-stop 443
 process-stopped-p 444
 process-unlock 444
 process-unstop 445
 process-wait 446
 process-wait-for-event 447
 process-wait-function 448
 process-wait-with-timeout 448
 process-whostate 449
 proclaim 106
 product-registry-path 717
 profiler-tree-from-function 248
 profiler-tree-to-function 249
 ps 450
 query 571
 query-registry-key-info 763
 query-registry-value 764
 quit 377
 read-dhparams 37
 read-foreign-modules 307
 read-serial-port-char 472
 read-serial-port-string 473
 reconnect 572
 record-definition 176
 references-who 250
 regexp-find-symbols 380
 registry-key-exists-p 765
 registry-value 766

remove-advice 381
 remove-special-free-action 251
 remove-symbol-profiler 251
 reset-profiler 252
 restore-sql-reader-syntax-
 state 573
 rollback 574
 room 108
 room-values 718
 round-to-single-precision 384
 run-shell-command 719
 safe-locale-file-encoding 724
 save-argument-real-p 253
 save-image 254
 save-tags-database 179
 save-universal-from-script 261
 sbchar 385
 schedule-timer 451
 schedule-timer-milliseconds 452
 schedule-timer-relative 454
 schedule-timer-relative-
 milliseconds 455
 select 575
 serial-port 472
 serial-port-input-available-p 474
 set-application-themed 753
 set-array-weak 263
 set-automatic-gc-callback 725
 set-blocking-gen-num 726
 set-default-character-element-
 type 385
 set-default-generation 264
 set-default-segment-size 729
 set-delay-promotion 730
 setf cdr-assoc 644
 setf timer-name 457
 set-file-dates 731
 set-gc-parameters 267
 set-gen-num-gc-threshold 731
 set-hash-table-weak 265
 set-make-instance-argument-
 checking 11
 set-maximum-memory 733
 set-maximum-segment-size 734
 set-memory-check 736
 set-memory-exhausted-callback 737
 set-minimum-free-space 269
 set-process-profiling 270
 set-profiler-threshold 272
 set-promotion-count 273
 set-registry-value 767
 set-serial-port-state 475

set-signal-handler 738
 set-spare-keeping-policy 740
 set-ssl-ctx-dh 40
 set-ssl-ctx-options 42
 set-ssl-ctx-password-callback 43
 set-ssl-library-path 44
 sets-who 277
 set-up-profiler 274
 set-verification-mode 38
 short-site-name 113
 simple-augmented-string-p 742
 simple-base-string-p 386
 simple-char-p 387
 simple-process-p 457
 simple-text-string-p 388
 single-form-form-parser 179
 single-form-with-options-form-
 parser 180
 socket-stream-address 50
 socket-stream-ctx 50
 socket-stream-peer-address 51
 socket-stream-ssl 52
 software-type 114
 software-version 115
 source-debugging-on-p 278
 sql 581
 sql-expression 583
 sql-operation 586
 sql-operator 588
 sql-recording-p 588
 sql-stream 589
 ssl-new 54
 start-dde-server 809
 start-profiling 278
 start-sql-recording 591
 start-tty-listener 388
 start-up-server 56
 start-up-server-and-mp 60
 staticip 744
 status 592
 stchar 389
 stop-profiling 280
 stop-sql-recording 593
 string-append 390
 string-ip-address 61
 sweep-all-objects 281
 sweep-gen-num-objects 746
 switch-static-allocation 282
 symeval-in-process 457
 table-exists-p 594
 text-string-p 391
 timer-expired-p 458
 timer-name 459
 toggle-source-debugging 283
 total-allocation 284
 traceable-dspec-p 181
 trace-new-instances-on-access 14
 trace-on-access 16
 tracing-enabled-p 182
 tracing-state 182
 true 392
 truename 129
 try-compact-in-generation 293
 try-move-in-generation 295
 typed-aref 746
 unbreak-new-instances-on-
 access 18
 unbreak-on-access 19
 unnotice-fd 460
 unschedule-timer 461
 untrace-new-instances-on-
 access 19
 untrace-on-access 20
 update-objects-joins 595
 update-records 597
 user-preference 393
 valid-external-format-p 190
 wait-for-input-streams 748
 wait-for-input-streams-returning-
 first 749
 wait-processing-events 462
 wait-serial-port-state 475
 whitespace-char-p 397
 who-binds 296
 who-calls 297
 who-references 297
 who-sets 298
 with-output-to-fasl-file 300
 write-serial-port-char 476
 write-serial-port-string 477
 yield 465
 fundamental-binary-input-stream
 class 605
 fundamental-binary-output-stream
 class 606
 fundamental-binary-stream class 606
 fundamental-character-input-stream
 class 607
 fundamental-character-output-stream
 class 607
 fundamental-input-stream class 608, 609
 fundamental-output-stream class 610
 fundamental-stream class 610

G

gc-generation function 227
gc-if-needed function 231
generic functions
 class-extra-initargs 3
 close 65
 compute-class-potential-initargs 4
 compute-discriminating-function 6
 dde-client-advise-data 776
 dde-server-poke 798
 dde-server-request 799
 dde-server-topic 800
 dde-server-topics 801
 dde-topic-items 802
 delete-instance-records 505, 518
 documentation 88
 get-inspector-values 355
 input-stream-p 92
 make-instance 99
 open-stream-p 104
 output-stream-p 105
 process-a-class-option 7
 process-a-slot-option 9
 slot-boundp-using-class 12
 slot-makunbound-using-class 13
 slot-value-using-class 13
 stream-advance-to-column 611
 stream-check-eof-no-hang 612
 stream-clear-input 612
 stream-clear-output 613
 stream-element-type 119
 stream-file-position 613
 stream-fill-buffer 614
 stream-finish-output 615
 stream-flush-buffer 616
 stream-force-output 617
 stream-fresh-line 617
 stream-line-column 618
 stream-listen 618
 stream-output-width 619
 stream-peek-char 620
 stream-read-buffer 621
 stream-read-byte 622
 stream-read-char 622
 stream-read-char-no-hang 623
 stream-read-line 624
 stream-read-sequence 624
 stream-read-timeout 625
 stream-start-line-p 626
 stream-terpri 627
 stream-unread-char 627

stream-write-buffer 628
stream-write-byte 629
stream-write-char 629
stream-write-sequence 630
stream-write-string 631
update-instance-from-records 595
update-record-from-instance 598
update-record-from-slot 599
update-slot-from-record 599
:gen-num initarg 745
gen-num-segments-fragmentation-state function 661
gesture-spec-accelerator-bit
 variable 663
gesture-spec-control-bit variable 663
gesture-spec-data function 664
gesture-spec-hyper-bit variable 664
gesture-spec-meta-bit variable 665
gesture-spec-modifiers function 665
gesture-spec-p function 666
gesture-spec-shift-bit variable 667
gesture-spec-super-bit variable 667
gesture-spec-to-character
 function 668
get-current-process function 409
get-default-generation function 231
get-file-stat function 668
get-folder-path function 670
get-foreign-symbol function 305
get-form-parser function 171
get-gc-parameters function 232
get-host-entry function 26
get-inspector-values generic
 function 355
get-process function 410
get-serial-port-state function 471
get-socket-address function 28
get-socket-peer-address function 29
get-unix-error function 357
get-user-profile-directory
 function 672
get-verification-mode function 29
get-working-directory function 233
grep-command variable 357
grep-command-format variable 358
grep-fixed-args variable 359
guess-external-format function 673
GUI application 255

H

handle-existing-action-in-action-list variable 359

handle-existing-action-list
 variable 360
handle-existing-defpackage
 variable 234
handle-missing-action-in-action-list
 variable 361
handle-missing-action-list
 variable 360
handle-old-in-package variable 235
handle-old-in-package-used-as-make-package variable 236
handle-warn-on-redefinition
 variable 361
hardcopy-system function 362
hash tables
 weak 98
hidden-packages variable 139
host 26
host name 26
hostname 26

I

init file 363
init-file-name variable 363
initialization file 363
initialize-database-type function 515
initialized-database-types
 variable 516
initialize-multiprocessing
 function 411
initial-processes special
 variable 412
InitLispWorks C function 811
input-stream-p generic function 92
insert-records function 517
inspect-through-gui variable 363
in-static-area macro 674
int32* function 676
int32+ function 677
int32- function 677
int32 type 674
int32/ function 680
int32/= function 680
int32< function 681
int32<< function 682
int32<= function 682
int32= function 683
int32> function 683
int32>= function 684
int32>> function 685
+int32-0+ symbol macro 678
int32-1+ function 678

int32-1- function 679
+int32-1+ symbol macro 678
int32-aref function 685
int32-logand function 686
int32-logandc1 function 687
int32-logandc2 function 687
int32-logbitp function 688
int32-logeqv function 689
int32-logior function 690
int32-logmand function 690
int32-lognor function 691
int32-lognot function 692
int32-logorc1 function 692
int32-logorc2 function 693
int32-logtest function 694
int32-logxor function 694
int32-minusp function 695
int32-plusp function 695
int32-to-integer function 696
int32-zerop function 697
integer-to-int32 function 697
interactive-stream-p function 93
IP Address 26
ip-address-string function 30

L

latin-1-code-pages variable 752
library formats 307
line-arguments-list variable 698
lisp image
 filename 364
 pathname 364
lisp-image-name function 364
lisp-name-to-foreign-name
 function 306
lispworks-directory variable 365
LispWorksDlsym C function 814
LispWorksState C function 815
list-all-processes function 413
list-attributes function 520
list-attribute-types function 519
list-classes function 521
listener 389
 top level commands 651
listener process 413
listener prompt 376
list-sql-streams function 522
list-tables function 523
load-all-patches function 366
load-data-file function 699
load-fasl-or-lisp-file variable 237
load-logical-pathname-translations

```

        function 94
*load-source-if-newer* 368
load-system function 367
*load-when-compile* 316
:lob-locator initarg 524
lob-stream class 523
lob-stream-lob-locator accessor 524
local-dspec-p function 172
locale-file-encoding function 700
locally-disable-sql-reader-syntax
    function 525
locally-enable-sql-reader-syntax
    function 525
location macro 173
lock-name function 414
lock-owner function 415
long-float type 94
long-site-name function 95
loop macro 95, 526

M
Mach-O bundle 258
Mach-O dynamically linked shared
    library 257
macros
    allocation-in-gen-num 195
    appendf 310
    at-location 152
    block-promotion 197
    cd 200
    dde-connect 777
    dde-disconnect 778
    declaim 73
    def 154
    defadvice 320
    defclass 77
    define-action 324
    define-action-list 326
    define-dde-client 796
    define-dde-dispatch-topic 802
    define-dde-server 804
    define-dde-server-function 805
    define-dspec-alias 155
    define-dspec-class 156
    define-form-parser 159
    define-top-loop-command 651
    defpackage 81
    defparser 467
    defsystem 328
    deftransform 133
    def-view-class 498
    delete-advice 213

        do-query 509
        execute-actions 346
        in-static-area 674
        location 173
        loop 95, 526
        profile 246
        rebinding 379
        removef 382
        restart-case 107
        simple-do-query 579
        step 116
        time 121
        trace 122
        undefine-action 392
        undefine-action-list 393
        untrace 129
        when-let 395
        when-let* 396
        with-action-item-error-
            handling 398
        with-action-list-mapping 399
        with-dde-conversation 797
        with-heavy-allocation 299
        with-lock 463
        with-noticed-socket-stream 62
        without-interrupts 464
        without-preemption 465
        with-output-to-string 131
        with-registry-key 769
        with-stream-input-buffer 632
        with-stream-output-buffer 633
        with-transaction 600
        with-unique-names 400
        mailbox-empty-p function 416
        mailbox-peek function 417
        mailbox-read function 417
        mailbox-reader-process function 418
        mailbox-send function 419
        mailbox-wait-for-event function 419
        *main-process* special variable 421
        make-array function 97
        make-gesture-spec function 701
        make-hash-table function 98
        make-instance generic function 99
        make-lock function 421
        make-mailbox function 423
        make-mt-random-state function 370
        make-named-timer function 424
        make-sequence function 100
        make-simple-int32-vector function 703
        make-ssl-ctx function 31
        make-stderr-stream function 704

```

m

- `make-timer` function 425
- `make-typed-aref-vector` function 704
- `make-unregistered-action-list`
 - function 369
- `map` function 101
- `map-all-processes` function 426
- `map-all-processes-backtrace`
 - function 426
- `map-process-backtrace` function 427
- `map-processes` function 428
- `map-query` function 528
- `mark-and-sweep` function 238
- `marking-gc` function 705
- `*max-trace-indent*` variable 239
- `memory-growth-margin` function 707
- `merge` function 102
- `merge-ef-specs` function 708
- Mersenne Twister 371
- metaobject protocol
 - class options 8
 - slot options 10
- MOP
 - class options 8
 - slot options 10
- `mt-random` function 371
- `mt-random-state` type 372
- `*mt-random-state*` variable 372
- `mt-random-state-p` function 372
- `*multibyte-code-page-ef*` variable 753
- `*mysql-library-directories*`
 - variable 529
- `*mysql-library-path*` variable 530

N

- `name-defined-dspecs` function 173
- `name-definition-locations`
 - function 174
- `name-only-form-parser` function 175
- `normal-gc` function 240
- `notice-fd` function 429

O

- `object-address` function 708
- `open` function 102
- opening a URL 712
- `open-pipe` function 709
- `open-registry-key` function 762
- `open-serial-port` function 469
- `openssl-version` function 35
- `open-stream-p` generic function 104
- `open-tcp-stream` function 32

o

- `open-url` function 712
- optimization hints 74
- `:optimize-slot-access` class option 14, 78
- `ora-lob-append` function 531
- `ora-lob-assign` function 532
- `ora-lob-char-set-form` function 532
- `ora-lob-char-set-id` function 533
- `ora-lob-close` function 534
- `ora-lob-copy` function 535
- `ora-lob-create-empty` function 536
- `ora-lob-create-temporary` function 537
- `ora-lob-disable-buffering`
 - function 538
- `ora-lob-element-type` function 539
- `ora-lob-enable-buffering` function 540
- `ora-lob-env-handle` function 540
- `ora-lob-erase` function 541
- `ora-lob-file-close` function 542
- `ora-lob-file-close-all` function 543
- `ora-lob-file-exists` function 543
- `ora-lob-file-get-name` function 544
- `ora-lob-file-is-open` function 545
- `ora-lob-file-open` function 546
- `ora-lob-file-set-name` function 546
- `ora-lob-flush-buffer` function 547
- `ora-lob-free` function 548
- `ora-lob-free-temporary` function 549
- `ora-lob-get-buffer` function 550
- `ora-lob-get-chunk-size` function 552
- `ora-lob-get-length` function 553
- `ora-lob-internal-lob-p` function 554
- `ora-lob-is-equal` function 554
- `ora-lob-is-open` function 555
- `ora-lob-is-temporary` function 555
- `ora-lob-load-from-file` function 556
- `ora-lob-lob-locator` function 557
- `ora-lob-locator-is-init` function 558
- `ora-lob-open` function 559
- `ora-lob-read-buffer` function 560
- `ora-lob-read-foreign-buffer`
 - function 563
- `ora-lob-read-into-plain-file`
 - function 562
- `ora-lob-svc-ctx-handle` function 564
- `ora-lob-trim` function 565
- `ora-lob-write-buffer` function 566, 568
- `ora-lob-write-from-plain-file`
 - function 567
- `output-stream-p` generic function 105

P

- *packages-for-warn-on-redefinition* variable 241
- parameters
 - *default-character-element-type* 324
- parse-float function 241
- parse-form-dspec function 176
- pathname of deliverable 364
- pathname of DLL 364
- pathname of dynamic library 364
- pathname of executable 364
- pathname of lisp image 364
- pathname-location function 373
- pem-read function 36
- pid-exit-status function 713
- pipe
 - open 710
- PL/SQL 514
- platform
 - *features* 89
 - software-type 114
 - software-version 115
- p-oci-env foreign type 541
- p-oci-file foreign type 558
- p-oci-lob-locator foreign type 558
- p-oci-svc-ctx foreign type 564
- pointer-from-address function 714
- pointers
 - weak 263
- precompile-regexp function 374
- print-action-lists function 375
- print-actions function 374
- *print-binding-frames* variable 140
- *print-catch-frames* variable 143
- *print-command* variable 375
- *print-handler-frames* variable 144
- *print-nickname* variable 376
- *print-open-frames* variable 145
- print-pretty-gesture-spec
 - function 715
- print-profile-list function 242
- print-query function 570
- *print-restart-frames* variable 146
- *print-symbols-using-bars*
 - variable 716
- process-a-class-option generic
 - function 7
- process-alive-p function 429
- process-allow-scheduling function 430
- process-arrest-reasons function 430
- process-a-slot-option generic
 - function 9
- process-break function 431
- process-continue function 431
- process-idle-time function 432
- *process-initial-bindings* special
 - variable 432
- process-interrupt function 434
- process-kill function 435
- process-lock function 435
- process-mailbox function 436
- process-name function 437
- process-p function 437
- process-plist function 437
- process-priority function 438
- process-reset function 439
- process-run-function function 439
- process-run-reasons function 441
- process-run-time function 441
- process-send function 442
- process-stop function 443
- process-stopped-p function 444
- process-unlock function 444
- process-unstop function 445
- process-wait function 446
- process-wait-for-event function 447
- process-wait-function function 448
- process-wait-with-timeout
 - function 448
- process-whostate function 449
- proclaim function 106
- product-registry-path function 717
- profile macro 246
 - *profiler-print-out-all* variable 245
 - *profiler-threshold* variable 247
- profiler-tree-from-function
 - function 248
- profiler-tree-to-function
 - function 249
- *profile-symbol-list* variable 248
- *prompt* variable 376
- prompt
 - in listener 376
- ps function 450

Q

- query function 571
- query-registry-key-info function 763
- query-registry-value function 764
- quit function 377
- QuitLispWorks C function 817

R

`read-dhparams` function 37
`read-eval-print` loop 389
`read-foreign-modules` function 307
`read-serial-port-char` function 472
`read-serial-port-string` function 473
`:read-timeout` initarg 45
`rebinding` macro 379
`reconnect` function 572
`record-definition` function 176
`*record-source-files*` variable 177
`*redefinition-action*` variable 178
`references-who` function 250
`regexp` 352, 380
`regexp-find-symbols` function 380
`registry`
 API on Windows 754
`registry-key-exists-p` function 765
`registry-value` function 766
`regular expression` 352, 380
`regular expression matching` 334, 352, 380
`remove-advice` function 381
`removef` macro 382
`remove-special-free-action`
 function 251
`remove-symbol-profiler` function 251
`REPL` 389
`*require-verbose*` variable 383
`reset-profiler` function 252
`restart-case` macro 107
`restore-sql-reader-syntax-state`
 function 573
`rollback` function 574
`room` function 108
`room-values` function 718
`round-to-single-precision`
 function 384
`run-shell-command` function 719

S

`safe-locale-file-encoding`
 function 724
`save-argument-real-p` function 253
`save-image` function 254
`save-tags-database` function 179
`save-universal-from-script`
 function 261
`sbchar` function 385
`schedule-timer` function 451
`schedule-timer-milliseconds`
 function 452

`schedule-timer-relative` function 454
`schedule-timer-relative-milliseconds`
 onds function 455
`select` function 575
`serial-port` function 472
`serial-port-input-available-p`
 function 474
`set-application-themed` function 753
`set-array-weak` function 263
`set-automatic-gc-callback`
 function 725
`set-blocking-gen-num` function 726
`set-default-character-element-type`
 function 385
`set-default-generation` function 264
`set-default-segment-size` function 729
`set-delay-promotion` function 730
`setf cdr-assoc` function 644
`setf timer-name` function 457
`set-file-dates` function 731
`set-gc-parameters` function 267
`set-gen-num-gc-threshold` function 731
`set-hash-table-weak` function 265
`set-make-instance-argument-checking`
 function 11
`set-maximum-memory` function 733
`set-maximum-segment-size` function 734
`set-memory-check` function 736
`set-memory-exhausted-callback`
 function 737
`set-minimum-free-space` function 269
`set-process-profiling` function 270
`set-profiler-threshold` function 272
`set-promotion-count` function 273
`set-registry-value` function 767
`set-serial-port-state` function 475
`set-signal-handler` function 738
`set-spare-keeping-policy` function 740
`set-ssl-ctx-dh` function 40
`set-ssl-ctx-options` function 42
`set-ssl-ctx-password-callback`
 function 43
`set-ssl-library-path` function 44
`sets-who` function 277
`set-up-profiler` function 274
`set-verification-mode` function 38
`*sg-default-size*` variable 741
`short-float` type 112
`short-site-name` function 113
`simple-augmented-string` type 742
`simple-augmented-string-p`
 function 742

simple-base-string type 113
simple-base-string-p function 386
simple-char type 387
simple-char-p function 387
simple-do-query macro 579
SimpleInitLispWorks C function 816
simple-int32-vector type 743
simple-process-p function 457
simple-text-string type 387
simple-text-string-p function 388
single-float type 113
single-form-form-parser function 179
single-form-with-options-form-parser function 180
:size initarg 745
slot-boundp-using-class generic function 12
slot-makunbound-using-class generic function 13
slot-value-using-class generic function 13
:socket initarg 45
socket-error class 45
socket-stream class 45
socket-stream-address function 50
socket-stream-ctx function 50
socket-stream-peer-address function 51
socket-stream-socket accessor 45
socket-stream-ssl function 52
software-type function 114
software-version function 115
source-debugging-on-p function 278
special forms
 declare 74
special variables
 current-process 406
 directory-link-transparency 83
 initial-processes 412
 main-process 421
 process-initial-bindings 432
splash screen 751
SQL
 stored procedure 514
sql function 581
SQL pseudo operators
 sql-boolean-operator 586
 sql-function 586
 sql-operator 586
sql-boolean-operator SQL pseudo operator 586
sql-connection-error condition 581
sql-database-data-error accessor 582
sql-database-data-error condition 582
sql-database-error condition 582
sql-enlarge-static variable 584
sql-error-database-message
 accessor 582
sql-error-secondary-error-id
 accessor 582
sql-expression function 583
sql-fatal-error condition 585
sql-function SQL pseudo operator 586
sql-libraries variable 585
sql-loading-verbose variable 585
sql-operation function 586
sql-operator function 588
sql-operator SQL pseudo operator 586
sql-recording-p function 588
sql-stream function 589
sql-temporary-error condition 590
sql-timeout-error condition 590
sql-user-error condition 591
ssl-cipher-pointer foreign type 52
ssl-cipher-pointer-stack foreign type 52
ssl-closed class 53
ssl-condition class 53
:ssl-configure-callback initarg 45
:ssl-ctx initarg 45
ssl-ctx-pointer foreign type 53
ssl-error class 54
ssl-failure class 54
ssl-new function 54
ssl-pointer foreign type 55
:ssl-side initarg 45
ssl-x509-lookup class 55
stack
 extension 220
 size 210, 741
stack size 650
stack-overflow-behaviour
 variable 743
standard-db-object class 591
start-dde-server function 809
start-profiling function 278
start-sql-recording function 591
start-tty-listener function 388
startup image 751
startup screen 751
startup window 751
start-up-server function 56
start-up-server-and-mp function 60
:static initarg 745

staticp function 744
status function 592
stchar function 389
stderr 704
step macro 116
step-compiled variable 118
step-filter variable 119
step-print-env variable 119
stop-profiling function 280
stop-sql-recording function 593
storage-exhausted class 745
storage-exhausted-gen-num
 accessor 745
storage-exhausted-size accessor 745
storage-exhausted-static accessor 745
storage-exhausted-type accessor 745
:stream-initarg 45
stream-advance-to-column generic
 function 611
stream-check-eof-no-hang generic
 function 612
stream-clear-input generic function 612
stream-clear-output generic
 function 613
stream-element-type generic
 function 119
stream-file-position generic
 function 613
stream-fill-buffer generic function 614
stream-finish-output generic
 function 615
stream-flush-buffer generic
 function 616
stream-force-output generic
 function 617
stream-fresh-line generic function 617
stream-line-column generic function 618
stream-listen generic function 618
stream-output-width generic
 function 619
stream-peek-char generic function 620
stream-read-buffer generic function 621
stream-read-byte generic function 622
stream-read-char generic function 622
stream-read-char-no-hang generic
 function 623
stream-read-line generic function 624
stream-read-sequence generic
 function 624
stream-read-timeout accessor 46
stream-read-timeout generic
 function 625
stream-start-line-p generic
 function 626
stream-terpri generic function 627
stream-unread-char generic function 627
stream-write-buffer generic
 function 628
stream-write-byte generic function 629
stream-write-char generic function 629
stream-write-sequence generic
 function 630
stream-write-string generic
 function 631
stream-write-timeout accessor 46
string type 120
string-append function 390
string-ip-address function 61
sweep-all-objects function 281
sweep-gen-num-objects function 746
switch-static-allocation function 282
symbol macros
 +int32-0+ 678
 +int32-1+ 678
symbol-alloc-gen-num variable 282
symeval-in-process function 457
system
 compile 315
 load 367
 print 362

T

table-exists-p function 594
***terminal-debugger-block-multipro-**
 cessing* variable 147
text-string type 391
text-string-p function 391
time macro 121
timer-expired-p function 458
timer-name function 459
toggle-source-debugging function 283
total-allocation function 284
trace macro 122
traceable-dspec-p function 181
traced-arglist variable 285
traced-results variable 123, 286
trace-indent-width variable 287
trace-level variable 288
trace-new-instances-on-access
 function 14
trace-on-access function 16
trace-print-circle variable 289
trace-print-length variable 289
trace-print-level variable 290

trace-print-pretty variable 291
trace-verbose variable 292
tracing-enabled-p function 182
tracing-state function 182
true function 392
trueName function 129
try-compact-in-generation
 function 293
try-move-in-generation function 295
tty 255
:type initarg 745
typed-aref function 746
types
 16-bit-string 310
 8-bit-string 309
 augmented-string 638
 base-char 311
 base-character 310
 base-string 65
 double-float 89
 extended-char 348
 extended-character 348
 int32 674
 long-float 94
 mt-random-state 372
 short-float 112
 simple-augmented-string 742
 simple-base-string 113
 simple-char 387
 simple-int32-vector 743
 simple-text-string 387
 single-float 113
 string 120
 text-string 391

U

unbreak-new-instances-on-access
 function 18
unbreak-on-access function 19
undefine-action macro 392
undefine-action-list macro 393
universal binaries
 helper functions 199, 253
 saving
 advanced 209
 simply 261
UNIX command
 call-system 640
 call-system-showing-output 641
 open-pipe 710
 run-shell-command 720
unnotice-fd function 460

unresolved-messages 308
unschedule-timer function 461
untrace macro 129
untrace-new-instances-on-access
 function 19
untrace-on-access function 20
update-instance-from-records generic
 function 595
update-objects-joins function 595
update-record-from-slot generic
 function 599
update-records function 597
update-records-from-instance generic
 function 598
update-slot-from-record generic
 function 599

URL
 opening 712
user-preference function 393

V

valid-external-format-p function 190
variables
 active-finders 151
 binary-file-type 72
 binary-file-types 69, 72
 browser-location 313
 cache-table-queries-default 483
 check-network-server 645
 compiler-break-on-error 206
 connect-if-exists 491
 debug-print-length 137
 debug-print-level 138
 default-action-list-sort-time 323
 default-database 497
 default-database-type 497
 default-libraries 307
 default-package-use-list 211
 default-process-priority 407
 default-profiler-collapse 211
 default-profiler-cutoff 212
 default-profiler-limit 212
 default-profiler-sort 213
 default-simple-process-priority 407
 default-stack-group-list-length 650
 default-update-objects-max-len 498
 defsystem-verbose 333
 describe-length 335

```

*describe-level* 335
*describe-print-length* 337
*describe-print-level* 337
*directory-link-transparency* 656
*disable-trace* 215
*dspec-classes* 163
*enter-debugger-directly* 342
*extended-spaces* 398, 657
*external-formats* 350
*features* 89
*file-encoding-detection-
    algorithm* 657
*file-eol-style-detection-
    algorithm* 659
*filename-pattern-encoding-
    matches* 659
gesture-spec-accelerator-bit 663
gesture-spec-control-bit 663
gesture-spec-hyper-bit 664
gesture-spec-meta-bit 665
gesture-spec-shift-bit 667
gesture-spec-super-bit 667
*grep-command* 357
*grep-command-format* 358
*grep-fixed-args* 359
*handle-existing-action-in-
    action-list* 359
*handle-existing-action-list* 360
*handle-existing-defpackage* 234
*handle-missing-action-in-action-
    list* 361
*handle-missing-action-list* 360
*handle-old-in-package* 235
*handle-old-in-package-used-as-
    make-package* 236
*handle-warn-on-redefinition* 361
*hidden-packages* 139
*init-file-name* 363
*initialized-database-types* 516
*inspect-through-gui* 363
*latin-1-code-pages* 752
*line-arguments-list* 698
*lispworks-directory* 365
*load-fasl-or-lisp-file* 237
*max-trace-indent* 239
*mt-random-state* 372
*multibyte-code-page-ef* 753
*mysql-library-directories* 529
*mysql-library-path* 530
*packages-for-warn-on-
    redefinition* 241
*print-binding-frames* 140
*print-catch-frames* 143
*print-command* 375
*print-handler-frames* 144
*print-nickname* 376
*print-open-frames* 145
*print-restart-frames* 146
*print-symbols-using-bars* 716
*profiler-print-out-all* 245
*profiler-threshold* 247
*profile-symbol-list* 248
*prompt* 376
*record-source-files* 177
*redefinition-action* 178
*require-verbose* 383
*sg-default-size* 741
*sql-enlarge-static* 584
*sql-libraries* 585
*sql-loading-verbose* 585
*stack-overflow-behaviour* 743
*step-compiled* 118
*step-filter* 119
*step-print-env* 119
*symbol-alloc-gen-num* 282
*terminal-debugger-block-
    multiprocessing* 147
*traced-arglist* 285
*traced-results* 123, 286
*trace-indent-width* 287
*trace-level* 288
*trace-print-circle* 289
*trace-print-length* 289
*trace-print-level* 290
*trace-print-pretty* 291
*trace-verbose* 292

```

W

```

wait-for-input-streams function 748
wait-for-input-streams-returning-
    first function 749
wait-processing-events function 462
wait-serial-port-state function 475
weak
    arrays 263
    hash tables 98
weak hash tables 98
weak pointers 263
web browser 712
when-let macro 395
when-let* macro 396
whitespace-char-p function 397
who-binds function 296
who-calls function 297

```

who-references function 297
who-sets function 298
Windows registry
 API 754
Windows XP themes 753
with-action-item-error-handling
 macro 398
with-action-list-mapping macro 399
with-dde-conversation macro 797
with-heavy-allocation macro 299
with-lock macro 463
with-noticed-socket-stream macro 62
without-interrupts macro 464
without-preemption macro 465
with-output-to-fasl-file function 300
with-output-to-string macro 131
with-registry-key macro 769
with-stream-input-buffer macro 632
with-stream-output-buffer macro 633
with-transaction macro 600
with-unique-names macro 400
write-serial-port-char function 476
write-serial-port-string function 477
:write-timeout initarg 45

Y

yellow pages 26
yield function 465

