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 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.

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

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
Contents

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
Contents

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
Contents

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
Contents

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
undefined-action 392
undefined-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
Contents

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
Contents

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
Contents

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
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
Contents

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.
Preface

- 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.
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 Editor User Guide for documentation on the LispWorks Editor.
Preface
This chapter describes the LispWorks extensions to CLOS, the Common Lisp Object System.

**break-new-instances-on-access**

**Function**

**Summary**
Breaks to the debugger when a new instance of a class is accessed. Note that this function is deprecated.

**Package**
clos

**Signature**
`break-new-instances-on-access class-designator &key read write slot-names when process trace-output entrycond eval-before before backtrace => t`

**Arguments**
- `class-designator` The class to trap.
  The keyword arguments control which type of access cause a break and are interpreted as described for `trace-on-access`.

**Values**
Returns `t`. 
Description Causes a break when new instances of the class given by class-designator are accessed, according to the keyword arguments.

**Note:** this function is deprecated. You should now call trace-new-instances-on-access with :break t instead.

See also trace-new-instances-on-access

### break-on-access

**Function**

**Summary** Breaks to the debugger when an instance of a class is accessed. Note that this function is deprecated.

**Package** clos

**Signature**

```
break-on-access instance &key read write slot-names when process trace-output entrycond eval-before before backtrace => t
```

**Arguments**

- **instance** A CLOS instance.
  
  The keyword arguments control which type of access cause a break and are interpreted as described for `trace-on-access`.

**Values** Returns t.

**Description** A useful debugging function which causes access to instance to break to the debugger. Accesses include calls to `slot-value` and also accessor functions defined by the class of instance. Other instances of the same class are unaffected.

You can remove the break by calling `unbreak-on-access`.

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

**Note:** this function is deprecated. You should now call `trace-on-access` with :break t instead.
See also  

**class-extra-initargs**  

*Generic Function*

**Summary**  Extends the valid initialization arguments of a class.

**Package**  clos

**Signature**  

class-extra-initargs prototype => initargs

**Arguments**  

- **prototype**  A class prototype.

**Values**  initargs  A list of additional initialization arguments.

**Description**  
The generic function **class-extra-initargs** lets you extend the set of valid initialization arguments for a class and its subclasses. *initargs* should be a list of symbols. Each symbol becomes a valid initarg for the class. By default in a non-delivered LispWorks image, **make-instance** checks that initargs passed to it are valid.

**Note:** **class-extra-initargs** 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 **:extra-initargs** class option, described in **defclass**.

**Example**  
In this session an illegal initarg :my-keyword is passed, causing **make-instance** to signal an error.

Then :my-keyword is added as an extra initarg, after which **make-instance** accepts it.
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.
The generic function `compute-class-potential-initargs` is called to compute the initialization arguments of a class. This set of valid initargs is used by `make-instance` when its arguments are checked.

`class` is the class passed to `make-instance`. That is, `compute-class-potential-initargs` specializes on the metaclass.

`initargs` is either a list of valid initargs, or `t` meaning that any initialization argument is allowed.

There is a supplied method on `t`, which returns `nil`.

The other supplied method is on `standard-class`. This consults the Relevant Methods, which are the applicable methods of `make-instance`, `allocate-instance`, `initialize-instance` and `shared-initialize`. If any of the Relevant Methods have a lambda list containing `&allow-other-keys` then `initargs` is `t`. Otherwise `initargs` is a list containing:

- all the `&key` arguments from Relevant Method lambda lists, and
- the initargs of the slots of `class` and its superclasses, and
- any extra initargs specified via the class option `:extra-initargs` (see `defclass` for details of this), and
- any extra initargs returned by `class-extra-initargs`.

The list `initargs` contains no duplicates, and the result of `compute-class-potential-initargs` is cached so that it is not recomputed unless one of the Relevant Methods, the class or its class precedence list is altered.

See also

- `class-extra-initargs`
- `make-instance`
- `set-make-instance-argument-checking`
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
The metaclass `funcallable-standard-object` provides the default `:direct-superclasses` for instances of `funcallable-standard-class` and its subclasses.

`funcallable-standard-object` is implemented as described in AMOP except for a different order in the class precedence list.

In AMOP the class precedence list is

```
(funcallable-standard-object standard-object function t)
```

whereas in LispWorks the class precedence list is

```
(funcallable-standard-object function standard-object t)
```

LispWorks is like this to be compliant with the rules in the ANSI Common Lisp Standard.

The AMOP class precedence list implies a class precedence for `generic-function` 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](http://www.lisp-works.com/reference/HyperSpec/Body/04_bb.htm).

---

**process-a-class-option**

**Generic Function**

**Summary**
Describes how the value of a class option is parsed.

**Package**
clos

**Signature**
`process-a-class-option metaclass option value => initargs`

**Arguments**
- `metaclass` The metaclass of the class being parsed.
- `option` The `defclass` option name.
- `value` The tail of the `defclass` option form.

**Values**
- `initargs` A plist of initargs describing the option.
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

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

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

```lisp
(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:

```lisp
(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))
  )

(defclass m2 (standard-class)
  ((titles-list :initarg :list-of-possible-titles)))
```
If the titles are to be evaluated, add a method like this:

```lisp
(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:

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

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

See also

```lisp
defclass
process-a-slot-option
```

---

**process-a-slot-option**

*Generic Function*

**Summary**

Describes how a `defclass` slot option is parsed.

**Package**

clos

**Signature**

`process-a-slot-option metaclass option value already-processed-other-options slot => processed-options`

**Arguments**

- `metaclass` The metaclass of the class being parsed.
- `option` The slot option name.
- `value` The value of the slot option.
- `already-processed-other-options`
A plist of initargs for non-standard options that have been processed already.

**slot**
The whole slot description.

**Values**
**processed-options**
A plist of initargs.

**Description**
The generic function `process-a-slot-option` describes how the value of a slot option is parsed. It is called at `defclass` 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.

`processed-options` should be a plist of slot initargs and values containing those from `already-processed-other-options` together with initargs for `option` as required. These are added to any other initargs for the slot.

**Example**

```lisp
(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))
```
To add a slot option \texttt{special-reader} whose value is a non-evaluated symbol naming a reader:

\begin{verbatim}
(deffunction 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)))
\end{verbatim}

To allow repeated \texttt{special-reader} options which are combined into a list:

\begin{verbatim}
(deffunction 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
          (list
           (progn
            (setf (cdr (last (cadr existing))) (list
             value))
            already-processed-options)
           (list* :special-reader
                  ',(value)
                  already-processed-options)))
      (call-next-method)))
\end{verbatim}

See also \texttt{defclass}

\texttt{process-a-class-option}

\textbf{set-make-instance-argument-checking} \emph{Function}

\textbf{Summary} Switches initarg checking in \texttt{make-instance} on or off.

\textbf{Package} clos

\textbf{Signature} \texttt{set-make-instance-initarg-checking on => on}
The CLOS Package

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>on</td>
<td>A boolean.</td>
</tr>
</tbody>
</table>

Description

The function `set-make-instance-initarg-checking` provides control over whether `make-instance` checks its initialization arguments.

Calling `set-make-instance-initarg-checking` with `on` true, causes `make-instance` to check the initargs. This is the initial state of LispWorks.

Initarg checking is switched off globally and dynamically by `(set-make-instance-initarg-checking nil)`.

See also

- `class-extra-initargs`
- `compute-class-potential-initargs`
- `make-instance`

---

**slot-boundp-using-class**

*Generic Function*

Summary

Implements `slot-boundp`.

Package

clos

Signature

`slot-boundp-using-class class object slot-name => result`

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>class</td>
<td>A class metaobject, the class of <code>object</code>.</td>
</tr>
<tr>
<td>object</td>
<td>An object.</td>
</tr>
<tr>
<td>slot-name</td>
<td>A slot name.</td>
</tr>
</tbody>
</table>

Values

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>result</td>
<td>A boolean.</td>
</tr>
</tbody>
</table>

Description

The generic function `slot-boundp-using-class` implements the behavior of the `slot-boundp` 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.
The CLOS Package

Package clos

Signature

(slot-value-using-class class object slot-name => value)

(setf slot-value-using-class) value class object slot-name => value

Arguments

class A class metaobject, the class of object.

object An object.

slot-name A slot name.

Values

value The value of the slot named by slot-name.

Description

The generic function slot-value-using-class implements the behavior of the slot-value 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.

Note: by default, standard slot accessors are optimized to not call slot-value-using-class. This can be overridden with the :optimize-slot-access class option. See defclass for details.

See also defclass

slot-boundp-using-class

slot-makunbound-using-class

trace-new-instances-on-access

Function

Summary Traces new instances of a given class, based on access modes.

Package clos
### Signature

```
(trace-new-instances-on-access class-designator &key read write slot-names break when process trace-output entrycond eval-before before backtrace => t)
```

### Arguments

- **class-designator**  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 `trace-on-access`.

### Values

Returns t.

### Description

Causes new instances of the class given by `class-designator` to be traced for the access modes given by `read`, `write` and `slot-names`.

This function, when used with the :break keyword, replaces the deprecated function `break-new-instances-on-access`.

### Example

```
(trace-new-instances-on-access 'capi:display-pane :
  slot-names nil)
```

Suppose you have a bug whereby the slot `bar` of an instance of your class `foo` 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:

```
(clos:trace-new-instances-on-access
  'foo
  :slot-names '(bar)
  :read nil
  :when '(and (integerp (car *traced-arglist*))
         (< (car *traced-arglist*) 0))
  :break t)
```

and running your program.

### See also

- `break-new-instances-on-access`
- `untrace-new-instances-on-access`
- `trace-on-access`
**Function**

### trace-on-access

**Summary**
Invokes the trace facilities when an instance of a class is accessed.

**Package**
clos

**Signature**
```
trace-on-access instance &key read write slot-names break when process trace-output entrycond eval-before before backtrace => t
```

**Arguments**

- `instance` A CLOS instance.
- `read` A generalized boolean.
- `write` A generalized boolean.
- `slot-names` A list of symbols, or `t`.
- `break` A generalized boolean.
- `when` A form.
- `process` A form.
- `trace-output` A form.
- `entrycond` A form.
- `eval-before` A list of forms.
- `before` A list of forms.
- `backtrace` A keyword, `t` or `nil`.

**Values**
Returns `t`.

**Description**
A useful debugging function which causes access to `instance` to invoke the trace facilities. Accesses include calls to `slot-value` and accessor functions defined by the class of `instance`.

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
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

untrace-on-access
trace-new-instances-on-access
break-on-access

unbreak-new-instances-on-access

Function

Summary  Removes the trapping installed by break-new-instances-on-access. Note that this function is deprecated.

Package  clos

Signature  unbreak-new-instances-on-access  class-designator  =>  t

Arguments  class-designator  The class whose trap you want to remove.
Values

Returns t.

Description

Removes the trapping installed by break-new-instances-on-access. Note that this function is deprecated. You should now use untrace-new-instances-on-access instead.

See also

untrace-new-instances-on-access

unbreak-on-access

Function

Summary

Removes the trapping installed by break-on-access. Note that this function is deprecated.

Package
clos

Signature
unbreak-on-access instance

Arguments
instance A class instance

Values

Returns t.

Description

Removes any break installed on instance by break-on-access. See untrace-on-access for details.

Note: this function is deprecated. You should now use untrace-on-access instead.

See also

untrace-on-access

untrace-new-instances-on-access

Function

Summary

Removes the tracing installed by trace-new-instances-on-access.

Package
clos
untrace-new-instances-on-access

**Signature**
untrace-new-instances-on-access class-designator => t

**Arguments**
class-designator  The class whose trap you want to remove.

**Values**
Returns t.

**Description**
Removes the tracing installed by trace-new-instances-on-access.

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

untrace-on-access

**Function**

**Summary**
Removes the tracing installed by trace-on-access.

**Package**
clos

**Signature**
untrace-on-access instance => t

**Arguments**
instance  A class instance

**Values**
Returns t.

**Description**
Removes any trace installed on instance by trace-on-access.

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

\[
\text{(require "comm")}
\]

**attach-ssl**

*Function*

Attaches SSL to a socket stream.

**Summary**

Attaches SSL to a socket stream.

**Signature**

\[
\text{attach-ssl socket-stream &key ssl-ctx ssl-side ctx-configure-callback ssl-configure-callback => ssl}
\]

**Arguments**

- `socket-stream`: A socket-stream.
- `ssl-ctx`: A symbol or a foreign pointer.
- `ssl-side`: One of the keywords :client, :server or :both.
**ctx-configure-callback**

A function designator or `nil`. The default value is `nil`.

**ssl-configure-callback**

A function designator or `nil`. The default value is `nil`.

**Values**

<table>
<thead>
<tr>
<th></th>
<th>ssl</th>
</tr>
</thead>
<tbody>
<tr>
<td>A foreign pointer of type <code>ssl-pointer</code>.</td>
<td></td>
</tr>
</tbody>
</table>

**Description**

The function `attach-ssl` attaches SSL to the socket-stream.

The allowed values and meaning of the keyword arguments are as described for `socket-stream`.

Note that `attach-ssl` is used by

- `(make-instance 'comm:socket-stream :ssl-ctx ...)`
- `(comm:open-tcp-stream ... :ssl-ctx ...)`

but you can also call it explicitly.

Before starting to create objects, `attach-ssl` ensures the SSL library (by calling `ensure-ssl`) and calls `do-rand-seed` to seed the Pseudo Random Number Generator (PRNG), so normally you do not need to worry about these.

If `ssl-ctx` is a symbol, it creates the `SSL_CTX` and calls `ctx-configure-callback` if this is non-`nil`. If `ssl-ctx` is not a `ssl-pointer`, it creates the `SSL` object, calls `ssl-configure-callback` if this is non-`nil`, and sets the ACCEPT/CONNECT state if the value of `ssl-side` is not `:both`. Then it uses `SSL_set_fd` to attach the `SSL` to the socket, and records this in the socket stream. It returns the `SSL`.

The default value of `ssl-ctx` is `t` and the default value of `ssl-side` is `:server`.

When a `socket-stream` is closed, `detach-ssl` is called with `:retry-count nil`, which, if the stream is attached to SSL,
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`
The function `destroy-ssl-ctx` frees the SSL_CTX pointed to by `ssl-ctx-pointer` and also frees any LispWorks cached values associated with it.

See also `ssl-ctx-pointer`

**Function**

**detach-ssl**

**Summary**
Detaches the SSL from a socket stream.

**Signature**
`detach-ssl socket-stream &key retry-count retry-timeout`

**Arguments**

- `socket-stream` A `socket-stream`.
- `retry-count` A non-negative integer.
- `retry-timeout` A non-negative integer.

**Description**

The function `detach-ssl` detaches the SSL from the socket-stream `socket-stream`. If `socket-stream` is not attached to an SSL, `detach-ssl` just returns immediately. Otherwise, it detaches the SSL from `socket-stream`, tries to shut down the SSL cleanly, and then frees the objects that were allocated by `attach-ssl`.

`retry-count` specifies how many additional times to call `SSL_shutdown` after the second to attempt to get a successful shutdown. The default value of `retry-count` is 5.

`retry-timeout` 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 `retry-timeout` is 0.1.

Note that the shutdown calls happen after the SSL has been detached from `socket-stream` as far as LispWorks is concerned, so if an error occurs at this point and is aborted, `socket-stream`
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 RAND_seed.

**Package**
comm

**Signature**
do-rand-seed

**Arguments**
do-rand-seed takes no arguments.

**Values**
do-rand-seed returns no values.

**Description**
The function do-rand-seed calls the SSL function RAND_seed 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 /dev/urandom, do-rand-seed does nothing.

See also attach-ssl

effect-ssl

**Function**

**Summary**
Initializes SSL.
Signature  ensure-ssl &key library-path already-done

Arguments  library-path  A string or a list of strings.
            already-done  A generalized boolean.

Description  The function ensure-ssl initializes SSL. If it was already called in the image, ensure-ssl does nothing. Otherwise it loads the library, calls SSL_library_init, calls SSL_load_error_strings and performs internal initializations.

If already-done is true, ensure-ssl does only the internal initializations. The default value of already-done is nil.

If library-path is passed, it needs to be either a string, specifying one library, or a list of strings specifying multiple libraries. The default value of library-path is platform-specific. The initial default value is described in the section "Loading the OpenSSL libraries" in the LispWorks User Guide and this default may be changed by calling set-ssl-library-path.

See also  openssl-version
        set-ssl-library-path

get-host-entry

Function

Summary  Returns address or name information about a given host.

Package  comm

Signature  get-host-entry host &key fields => field-values

Arguments  host  A number or a string.
            fields  A list of keywords.
Values

*field-values*

Values, one for each field.

Description

Using whatever host naming services are configured on the current machine, `get-host-entry` returns address or name information about the given host. `nil` is returned if the host is unknown.

The `host` argument can be one of the following:

- a name string, for example "www.foobar.com"
- a dotted inet address string, for example "209.130.14.246"
- a integer representing the inet address, for example `#xD1820EF6`

The `fields` argument is a list of keywords describing what information to return for the host. If `get-host-entry` succeeds, it returns multiple values, one value for each field specified. The following fields are allowed:

- `:address` The primary IP address as an integer.
- `:addresses` A list of all the IP addresses as integers.
- `:name` The primary name as a string.
- `:aliases` The alias names as a list of strings.

**Note:** although the results of `get-host-entry` are not cached by LispWorks, the operating System might cache them.
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
comm

Signature
get-socket-address socket => address, port

Arguments
socket A socket handle.

Values
address The local host address of the socket or nil if not connected.

port The local port number of the socket or nil if not connected.

Description
Connected sockets have two addresses, local and remote.
The get-socket-address function returns the local address.

See also
get-socket-peer-address
socket-stream-address
**get-socket-peer-address**

*Function*

**Summary**

Returns the remote address and port number of a given socket.

**Package**

*comm*

**Signature**

`get-socket-peer-address socket => address, port`

**Arguments**

`socket` A socket handle.

**Values**

`address` The remote host address of the socket or `nil` if not connected.

`port` The remote port number of the socket or `nil` if not connected.

**Description**

Connected sockets have two addresses, local and remote. The `get-socket-peer-address` function returns the remote address.

**See also**

`get-socket-address`

`socket-stream-peer-address`

---

**get-verification-mode**

*Function*

**Summary**

Returns the mode of the SSL.

**Package**

*comm*

**Signature**

`get-verification-mode ssl-or-ssl-ctx => result`

**Arguments**

`ssl-or-ssl-ctx` A foreign pointer of type `ssl-pointer` or `ssl-ctx-pointer`. 
The function `get-verification-mode` returns the mode of the `ssl-pointer` or `ssl-ctx-pointer` as a list of symbols. 

result is a list containing zero or more of the symbols `:verify-client-once`, `:verify-peer` and `:fail-if-no-peer-cert`, corresponding to the C constants `VERIFY_CLIENT_ONCE` `VERIFY_PEER` and `FAIL_IF_NO_PEER_CERT` respectively.

See also `set-verification-mode`

**ip-address-string**

*Function*

Summary Returns the dotted IP address string from the integer IP address.

Package `comm`

Signature `ip-address-string` `ip-address` `=>` `string-ip-address`

Arguments `ip-address` An integer.

Values `string-ip-address` The dotted string format of the given IP address.

Description The `ip-address-string` function converts its argument to a string in the standard dotted IP address notation `a.b.c.d`.

See also `string-ip-address`
**make-ssl-ctx**

**Function**

**Summary**
Makes a **SSL_CTX** object.

**Package**
`comm`

**Signature**

```
make-ssl-ctx &key ssl-ctx ssl-side => ssl-ctx-ptr
```

**Arguments**

- `ssl-ctx` A symbol or a foreign pointer.
- `ssl-side` One of the keywords `:client`, `:server` or `:both`.

**Values**

- `ssl-ctx-ptr` A foreign pointer of type **ssl-ctx-pointer**.

**Description**

The function `make-ssl-ctx` first calls `ensure-ssl`, and returns a foreign pointer of type **ssl-ctx-pointer**.

If the value of `ssl-ctx` is `t`, :default, :v2, :v3, :v23 or :tls-v1, `make-ssl-ctx` creates a **SSL_CTX** object and returns a pointer to it.

The value of `ssl-ctx` can also be a foreign pointer of type **ssl-ctx-pointer**, in which case it is simply returned. If `ssl-ctx` is a foreign pointer of type **ssl-pointer**, then `make-ssl-ctx` signals an error.

The meaning of the keyword arguments `ssl-ctx` and `ssl-side` is as described for **socket-stream**. The default value of `ssl-ctx` is `t` and the default value of `ssl-side` is `:server`.

**See also**

- `ensure-ssl`
- `socket-stream`
- `ssl-ctx-pointer`
open-tcp-stream

Summary
Attempts to connect to a socket on another machine and returns a stream object for the connection.

Package
comm

Signature
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

Arguments

hostname An integer or string.

service A string or a fixnum.

direction One of :input, :output or :io.

element-type base-char or a subtype of integer.

errorp A boolean.

read-timeout A positive number, or nil.

write-timeout A positive number, or nil.

timeout A positive number, or nil.

ssl-ctx A symbol or a foreign pointer.

ctx-configure-callback
A function designator or nil. The default value is nil.

ssl-configure-callback
A function designator or nil. The default value is nil.

local-address nil, an integer or a string.

local-port nil, a string or a fixnum.

nodelay A generalized boolean.

keepalive A generalized boolean.
Values

- `stream-object` A socket stream.

Description

The `open-tcp-stream` function attempts to connect to a socket on another machine and returns `stream-object` for the connection if successful. The server machine to connect to is given by `hostname`, which can be one of the following:

- A string naming the host, for example, "www.nowhere.com"
- A string providing the IP address, for example, "204.71.177.75"
- An integer IP address in network order, for example, \#xCC47B14B

The name of the service to provide is given by `service`. If `service` is a string, the location of the file specifying the names of the 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`

The `service` can also be a fixnum representing the port number of the desired connection.

The direction of the connection is given by `direction`. Its default value is `:io`. The element type of the connection is determined from `element-type`, and is `base-char` by default.

If `errorp` is `nil`, failure to connect (possibly after `timeout` seconds) returns `nil`, otherwise an error is signaled.

`timeout` specifies a connection timeout. `open-tcp-stream` waits for at most `timeout` seconds for the TCP connection to be made. If `timeout` is `nil` it waits until the connection attempt succeeds or fails. On failure, `open-tcp-stream` signals an error or returns `nil` according to the value of `errorp`. To provide a timeout for reads after the connection is made, see `read-timeout`. The default value of `timeout` is `nil`. 
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\r\n\r\n")
  (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
comm

Signature
openssl-version &optional what => result

Arguments
what

Values
result
A string.

Description
The function openssl-version returns a string specifying the version of the loaded OpenSSL library.

The argument what takes these values:
The COMM Package

2.6 The COMM Package

:version

result is the version string, which looks like:
"OpenSSL 0.9.7i 14 Oct 2005" or
"OpenSSL 0.9.8a 11 Oct 2005"

:built-on

Returns a string specifying when it was built.

:directory

Returns where OpenSSL thinks it is installed.

:platform

Returns OpenSSL’s idea of which platforms it is.

:cflags

The compilation command.

The default value of what is :version.

See also

ensure-ssl

pem-read

Function

Summary

An interface to the SSL PEM_read_bio_* functions.

Package

comm

Signature

pem-read thing-to-read filename &key pass-phrase callback errorp => result

Arguments

thing-to-read A string.

filename A pathname designator.

pass-phrase A string, or nil.

callback A function designator, or nil.

errorp A generalized boolean.

Values

result A foreign pointer or nil.
The function `pem-read` is an interface to the `PEM_read_bio_` set of functions. See the manual entry for `pem` for specifications of these functions.

`thing-to-read` defines which function is required. `pem-read` concatenates `thing-to-read` with the string "PEM_read_bio_" to form the name of the `pem` function to call.

`filename` specifies the file to load.

If `pass-phrase` is non-nil, it must be a string, which is passed to the `pem` function. The default value of `pass-phrase` is `nil`.

If `callback` is non-nil, it must be a function with signature:

```
callback maximum-length rwflag => pass-phrase
```

where `maximum-length` is an integer, `rwflag` is a boolean and `pass-phrase` is the pass-phrase to use. The default value of `callback` is `nil`, but you cannot pass non-nil values for both `pass-phrase` and `callback`.

If it succeeds, `pem-read` returns a foreign pointer to the structure that was returned by the `pem` function. If `pem-read` fails, if `errorp` is non-nil it signals an error, otherwise it returns `nil`. The default value of `errorp` is `nil`.

### read-dhparams

**Function**

Reads or uses cached SSL DH parameters.

**Package**

`comm`

**Signature**

```
read-dhparams filename &key pass-phrase callback errorp force =>
  dh-ptr
```

**Arguments**

- `filename` A pathname designator.
- `pass-phrase` A string, or `nil`.

---

37
The function `read-dhparams` reads or uses cached DH parameters. `filename` specifies the file to check.

Unless `force` is true, `read-dhparams` checks if the file `filename` has already been loaded, and if it has been loaded, uses the cached value.

If `force` is true, or if there is no cached value for `filename`, `read-dhparams` loads the file by calling `pem-read` with `thing-to-read` argument "DHparams", `pass-phrase`, `callback` and `errorp`. `read-dhparams` caches and returns a foreign pointer to the resulting DH structure (that is, a pointer corresponding to the C type `DH*`).

If `read-dhparams` fails to load the file `filename`, if `errorp` is true it signals an error, otherwise it returns `nil`. The default value of `errorp` is `t`.

See also `pem-read`

### set-verification-mode

**Function**

**Summary**
Sets the verification mode for CTX.

**Package**
`comm`

**Signature**

```
set-verification-mode ssl-ctx ssl-side mode &optional callback
```
Arguments

- ssl-ctx: A foreign pointer of type ssl-pointer or ssl-ctx-pointer.
- ssl-side: server or client.
- mode: An integer, one of the symbols never, always, once, or a list of keywords.
- callback: A foreign function.

Values

- result: A list of symbols.

Description

The function set-verification-mode sets the verification mode for CTX according to arguments ssl-side and mode.

When ssl-side is server, mode can be:

- An integer: mode is passed directly to SSL_set_verify or SSL_CTX_set_verify.
- :never: The server will not send a client certificate request to the client, so the client will not send a certificate.
- :always: 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.
- :once: Same as :always 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 :verify-client-once, :verify-peer and :fail-if-no-peer-cert. These keywords map to the corresponding C constants VERIFY_CLIENT_ONCE, VERIFY_PEER 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 mode is passed directly as for ssl-side :server.

:never 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.

:always 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 ssl-side :server.

If non-nil callback should be a symbol, function, string or foreign pointer designating a foreign function that is called to perform verification. The default value of callback is nil.

See also get-verification-mode

---

**set-ssl-ctx-dh**

*Function*

**Summary** Sets the DH parameters for a SSL_CTX.

**Package** comm
Signature

```
set-ssl-ctx-dh ssl-ctx filekey dh filename func filename-list pass-
phrase callback => result
```

Arguments

- `ssl-ctx` A foreign pointer.
- `filename` A pathname designator or `nil`.
- `func` A function designator or `nil`.
- `filename-list` An association list.
- `pass-phrase` A string, or `nil`.
- `callback` A function designator, or `nil`.

Values

- `result` A boolean.

Description

The function `set-ssl-ctx-dh` sets the DH parameters for a `SSL_CTX`.

`ssl-ctx` can be either a foreign pointer of type `ssl-ctx-pointer` or a foreign pointer of type `ssl-pointer`.

The value to use is specified by one of the parameters `dh`, `filename`, `func` or `filename-list`.

If `dh` is non-`nil`, it must be a foreign pointer to a DH (corresponding to the C type `DH*`), and this DH is used as is. The default value of `dh` is `nil`.

Otherwise, if `filename` is non-`nil`, it must be a pathname designator for a file containing DH parameters, which is loaded (by `read-dhparams`) and then used. In this case, `pass-phrase` and `callback` can be used, and are passed to `pem-read`.

Otherwise, if `func` is non-`nil`, it must be a function with signature:

```
func is-export keylength => dh-ptr
```

where `is-export` is a boolean, `keylength` is an integer, and `dh-ptr` is a pointer to an appropriate DH structure. `set-ssl-ctx-dh` installs `func` as the DH callback.
Otherwise (that is, if each of \textit{dh}, \textit{filename} and \textit{func} are \texttt{nil}) then \textit{filename-list} must be a non-\texttt{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). \texttt{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 \texttt{read-dhparams}), and returns it. It also loads the first file of the list immediately.

\textit{result} is \texttt{t} on success, \texttt{nil} otherwise.

See also \texttt{pem-read}, \texttt{read-dhparams}, \texttt{ssl-ctx-pointer}, \texttt{ssl-pointer}

\section*{set-ssl-ctx-options}

\textit{Function}

\textbf{Summary} Sets the options in a \texttt{SSL_CTX}.

\textbf{Package} \texttt{comm}

\textbf{Signature} \texttt{set-ssl-ctx-options ssl-ctx \&key microsoft(sess_id bug netscape(challenge bug netscape(reuse_cipher_change bug netsleref2(reuse_cert_type bug microsoft(big_sslv3 buffer msie_sslv2_rsa_padding sssl(eay_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_tlsv1 pkcs1_check_1 pkcs1_check_2 netscape_ca_dn_bug netscape_demo_cipher_change bug)}

\textbf{Arguments} \texttt{ssl-ctx} A foreign pointer.

Each of the keyword arguments is a generalized boolean defaulting to \texttt{nil}.
The function `set-ssl-ctx-options` sets the options in a `SSL_CTX`.

`ssl-ctx` can be either a foreign pointer of type `ssl-ctx-pointer` or a foreign pointer of type `ssl-pointer`.

The option that is set is the `logior` of all the options that are passed to `set-ssl-ctx-options` via the keyword arguments. The value used for each non-nil keyword `keyword` is the value of `SSL_OP_keyword`. The meaning of the options is specified in the OpenSSL manual page for `SSL_set_options`.

See also

- `ssl-ctx-pointer`
- `ssl-pointer`

**set-ssl-ctx-password-callback**

*Function*

**Summary**

Sets the password for a `SSL_CTX`.

**Package**

`comm`

**Signature**

`set-ssl-ctx-password-callback ssl-ctx $key callback password`

**Arguments**

- `ssl-ctx` A foreign pointer.
- `callback` A function designator, or `nil`.
- `password` A string, or `nil`.

**Description**

The function `set-ssl-ctx-password-callback` sets the password for a `SSL_CTX`, either to a callback or a password.

`ssl-ctx` should be a foreign pointer of type `ssl-ctx-pointer`.

If `callback` is non-nil, it must be a function with signature:

`callback maximum-length rwflag => result`
where maximum-length is an integer, rawflag 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
**socket-error**

*Class*

Summary  The condition class for socket errors.

Superclasses  *simple-error*

Subclasses  *ssl-condition*

Initargs  

A socket-stream.

Description  The condition class for socket errors.

**socket-stream**

*Class*

Summary  The socket stream class.

Superclasses  *buffered-stream*

Initargs  

A socket handle.

One of *input*, *output*, or *io*.

An element type.

A positive number or *nil*.

A positive number or *nil*.

A keyword, *t* or *nil*, or a foreign pointer of type *ssl-ctx-pointer* or *ssl-pointer*.

One of the keywords *client*, *server* or *both*. The default value is *server*.

A function designator or *nil*.

A function designator or *nil*.

Accessors  *socket-stream-socket*
The `socket-stream` class implements a buffered stream connected to a socket. The socket handle, specified by `:socket`, and the direction, specified by `:direction`, must be passed for a meaningful stream to be constructed. Common Lisp input functions such as `read-char` will see `end-of-file` if the other end of the socket is closed.

The `:element-type` 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 `socket-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 `socket-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 `socket-stream`: the output is converted from the character data as if by `char-code`. Also, 8-bit binary data can be read and written to a `base-char socket-stream`.

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

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

The `read-timeout` property is intended for use when a socket connection might hang during a call to any Common Lisp input function. The `read-timeout` can be set by `make-instance` or by `open-tcp-stream`. It can also be modified by `(setf stream:stream-read-timeout)`. When `read-timeout` is `nil`, there is no timeout during reads and the call may hang.

When `read-timeout` is not `nil`, and there is no input from the socket for more than `read-timeout` seconds, any reading function returns `end-of-file`. The `read-timeout` does not limit the time inside `read`, but the time between successful extractions
of data from the socket. Therefore, if the reading needs several rounds it may take longer than \textit{read-timeout}.

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

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

The \textit{write-timeout} property is similar to \textit{read-timeout}, but for write operations. If flushing the stream buffer takes too long then \texttt{error} is called.

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

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

The value of \texttt{ssl-ctx} can also be a foreign pointer of type \texttt{ssl-ctx-pointer} (which corresponds to the C type \texttt{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 \texttt{make-ssl-ctx}, but it can also a result of user code, provided that it points to a valid \texttt{SSL_CTX} and has the type \texttt{ssl-ctx-pointer}. 

47
The value of \texttt{ssl-ctx} can also be a foreign pointer of type \texttt{ssl-pointer} (which corresponds to the C type \texttt{SSL*}). This specifies the SSL to use. This maybe a result of a call to \texttt{ssl-new} but can also be the result of user code, provided that it points to a valid SSL object and has the type \texttt{ssl-pointer}. The SSL is used and is not freed when the stream is closed.

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

\texttt{ssl-side} specifies which side the socket stream is. The value of \texttt{ssl-side} is used in two cases:

- When a new \texttt{SSL_CTX} object is created, it is used to select the method:
  \begin{verbatim}
  :client => *client_method
  :server => *server_method
  :both   => *method
  \end{verbatim}

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

If the value of \texttt{ssl-ctx} is a \texttt{ssl-pointer}, \texttt{ssl-side} is ignored.

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

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

\textbf{Example} The following makes a bidirectional stream connected to a socket specified by \texttt{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
socket-stream-address

Function

Summary
Returns the local address and port number of a given socket stream.

Package
comm

Signature
socket-stream-address stream => address, port

Arguments
stream A socket stream.

Values
address The local host address of the socket stream or nil if not connected.
port The local port number of the socket stream or nil if not connected.

Description
Connected socket streams have two addresses, local and remote. The socket-stream-address function returns the local address.

See also
socket-stream-peer-address
get-socket-address

socket-stream-ctx

Function

Summary
Accesses the SSL_CTX attached to a socket stream.

Package
comm

Signature
socket-stream-ctx socket-stream => ssl-ctx-pointer

Arguments
socket-stream A socket-stream.
Values

ssl-ctx-pointer  A foreign pointer of type ssl-ctx-pointer, or nil.

Description

The function socket-stream-ctx accesses the SSL_CTX that is attached to the socket-stream socket-stream.

It returns nil if SSL is not attached.

See also

socket-stream
ssl-ctx-pointer

socket-stream-peer-address  Function

Summary

Returns the remote address and port number of a given socket stream.

Package

comm

Signature

socket-stream-peer-address  stream => address, port

Arguments

stream  A socket stream.

Values

address  The remote host address of the socket stream or nil if not connected.

port  The remote port number of the socket stream or nil if not connected.

Description

Connected socket streams have two addresses, local and remote. The socket-stream-peer-address function returns the remote address.

See also

socket-stream-address
get-socket-peer-address
**socket-stream-ssl**

*Function*

Summary: Accesses the SSL attached to a socket stream.

**Package**: comm

**Signature**: `socket-stream-ssl socket-stream => ssl-pointer`

**Arguments**: `socket-stream` A socket-stream.

**Values**: `ssl-pointer` A foreign pointer of type `ssl-pointer`, or `nil`.

**Description**: The function `socket-stream-ssl` accesses the SSL that is attached to the `socket-stream`. It returns `nil` if SSL is not attached.

See also: `socket-stream`, `ssl-pointer`

---

**ssl-cipher-pointer**

*FLI type specifier*

**Summary**: An FLI type for use with SSL.

**Package**: comm

**Signature**: `ssl-cipher-pointer`

**Description**: The FLI type `ssl-cipher-pointer` corresponds to the C type `SSL_CIPHER*`.

---

**ssl-cipher-pointer-stack**

*FLI type specifier*

**Summary**: An FLI type for use with SSL.
Package: comm

Signature: ssl-cipher-pointer-stack

Description: The FLI type `ssl-cipher-pointer-stack` corresponds to the C type `STACK_OF(SSL_CIPHER)`.

**ssl-closed**

Class

Summary: The class for SSL errors corresponding to `SSL_ERROR_ZERO_RETURN`.

Superclasses: ssl-condition

Description: The condition class `ssl-closed` corresponds to `SSL_ERROR_ZERO_RETURN`. It means the underlying socket is dead.

**ssl-condition**

Class

Summary: The condition class for SSL errors.

Superclasses: socket-error

Subclasses: ssl-closed, ssl-error, ssl-failure, ssl-x509-lookup

Description: The condition class for errors inside SSL.

**ssl-ctx-pointer**

FLI type specifier

Summary: An FLI type for use with SSL.
## Package comm

**Signature**
ssl-ctx-pointer

**Description**
The FLI type `ssl-ctx-pointer` corresponds to the C type `SSL_CTX*`.

### ssl-error

**Class**

**Summary**
The class for SSL errors corresponding to `SSL_ERROR_SYSCALL`.

**Superclasses**
ssl-condition

**Description**
The condition class `ssl-error` corresponds to `SSL_ERROR_SYSCALL`. It means that something got broken.

### ssl-failure

**Class**

**Summary**
The class for SSL errors corresponding to `SSL_ERROR_SSL`.

**Superclasses**
ssl-condition

**Description**
The condition class `ssl-failure` corresponds to `SSL_ERROR_SSL`. 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 SSL.
Package  comm
Signature  ssl-new ssl-ctx-pointer => ssl-pointer
Arguments  ssl-ctx-pointer  A foreign pointer of type ssl-ctx-pointer.
Values  ssl-pointer  A foreign pointer of type ssl-pointer.
Description  The function ssl-new creates a SSL by a direct call to the C function SSL_new.
It returns a pointer to the new SSL.
See also  ssl-ctx-pointer
ssl-pointer

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

ssl-x509-lookup

Class
Summary  The class for SSL errors corresponding to SSL_ERROR_WANT_X509_LOOKUP.
Superclasses  ssl-condition
Description  The condition class ssl-x509-lookup corresponds to
SSL_ERROR_WANT_X509_LOOKUP. It happens when a certificate
is rejected by a user callback.
**start-up-server**

---

**Summary**

Starts a TCP server.

**Package**

`comm`

**Signature**

```
start-up-server &key function announce service address nodelay keepalive process-name wait error => process, startup-condition
```

**Arguments**

- `function`: A function name.
- `announce`: An output stream, `t`, `nil` or a function.
- `service`: An integer, a string or `nil`.
- `address`: An integer, a string or `nil`.
- `nodelay`: A generalized boolean.
- `keepalive`: A generalized boolean.
- `process-name`: A symbol or expression.
- `wait`: A boolean.
- `error`: A boolean.

**Values**

- `process`: A process, or `nil`.
- `startup-condition`: A condition object, or `nil`.

**Description**

The `start-up-server` function starts a TCP server. Use `process-kill` to kill the server, and `open-tcp-stream` to send messages from another client to the server.

The `function` argument provides the name of the function that processes connections. When a connection is made `function` is called with the connected socket handle, at which point you can make a stream using `make-instance` and communicate with the client. The server does not accept more connections until `function` returns, so normally it should create another light-weight process to handle the connection. However, the operating system typically provides a small queue of
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 and 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))))

59
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

**Function**

**Package**
comm

**Signature**

```
start-up-server-and-mp &key function announce service address process-name
```

**Arguments**

- `function`: A function name.
- `announce`: An output stream, `t`, `nil` or a function.
service       An integer, a string or nil.
address       An integer, a string or nil.
process-name  A symbol or expression.

Description The **start-up-server-and-mp** function starts multiprocess-
ing if it has not already been started and then calls **start-up-
server** with the supplied function, announce, service, address
and process-name arguments.

**Note:** **start-up-server-and-mp** is implemented only on
Unix/Linux/Mac OS X platforms.

See also     **start-up-server**

---

**string-ip-address**  
*Function*

**Summary**         Returns the integer IP address from the given dotted IP
                    address string.

**Package**         comm

**Signature**       

```
string-ip-address  ip-address-string  =>  ip-address
```

**Arguments**       

```
string-ip-address  A string denoting an IP address in dotted
                    format.
```

**Values**          

```
ip-address       An integer IP address.
```

**Description**     The **string-ip-address** function takes a string in the stan-
dard dotted IP address notation `a.b.c.d` and returns the cor-
responding integer IP address.

See also     **ip-address-string**
with-noticed-socket-stream  

**Macro**

**Package**  
comm

**Signature**  
with-noticed-socket-stream (stream) &body body

**Arguments**

- **stream**  
A stream created using open-tcp-stream.

- **body**  
Code to be executed while the stream is “noticed”.

**Description**

If you do a process-wait 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 process-wait.

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.

The macro is designed to be used with streams created by the function open-tcp-stream.

**Note:** with-noticed-socket-stream is implemented only on Unix/Linux/Mac OS X platforms.

**See also**  
open-tcp-stream
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.

**apropos**

*Function*

- **Summary**: Searches for interned symbols.
- **Package**: `common-lisp`
- **Signature**: `apropos string &optional package external-only => <no values>`
- **Arguments**:
  - `string`: A string designator.
  - `package`: A package designator or `nil`.
  - `external-only`: A generalised boolean.
The function `apropos` behaves as specified in ANSI Common Lisp. There is an additional optional argument `external-only`, which if true restricts the search to symbols which are external in the searched package or packages. The default value of `external-only` is `nil`.

See also

- `apropos-list`
- `*describe-print-length*`
- `*describe-print-level*`
- `regexp-find-symbols`

---

### apropos-list

**Function**

**Summary**

Searches for interned symbols.

**Package**

`common-lisp`

**Signature**

`apropos-list string &optional package external-only => symbols`

**Arguments**

- `string`: A string designator.
- `package`: A package designator or `nil`.
- `external-only`: A generalised boolean.

**Values**

- `symbols`: A list of symbols.

**Description**

The function `apropos-list` behaves as specified in ANSI Common Lisp. There is an additional optional argument `external-only`, which if true restricts the search to symbols which are external in the searched package or packages. The default value of `external-only` is `nil`.

See also

- `apropos`
**base-string**

*Type*

**Summary**
The base string type.

**Package**
common-lisp

**Signature**
base-string length

**Arguments**
length The length of the string (or *, meaning any).

**Description**
The type of base strings.

---

**close**

*Generic Function*

**Summary**
The close function is implemented as a generic function.

**Package**
common-lisp

**Signature**
close stream &key abort => result

**Arguments**
stream A stream.
abort A generalized boolean.

**Values**
result A boolean.

**Description**
The standard function close 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 close is as per the Common Lisp ANSI specification.

When stream is an instance of a subclass of buffered-stream, if abort is true then any remaining data in the buffer can be discarded. There are two built-in methods on buffered-stream. The primary method specialized on buffered-stream 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**

### Function

**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**

- **object** A Lisp object.
- **result-type** A type specifier.

**Values**

- **result** An object of type `result-type`

**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 `result` cannot be returned as the `result-type` specifies.

**See also**

- **concatenate**
**compile**  

*Function*

**Summary**  
Compiles a lambda expression into a code vector.

**Package**  
*common-lisp*

**Signature**  
`compile name &optional definition => name, function`

**Arguments**  
- `definition`  
  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 `name` must be a non-`nil` symbol with an uncompiled definition).

- `name`  
  If not `nil`, this is the symbol that is to receive the compiled function as its global function definition.

**Values**  
A single value is returned, being the `name` symbol if supplied, or when `name` is `nil` the compiled function definition itself. Such compiled-function objects are not printable (but see `disassemble`) other than as `#<compiled function for SYM-BOL>`.

**Description**  
`compile` 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**  
```lisp
(defun fn (...) ...) ; interpreted definition for fn
(compile 'fn) ; replace with compiled
             ; definition
```
(compile nil '(lambda (x) (* x x)))
; returns compiled squaring function

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

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

See also
compile-file
disassemble
declare

csscale

Function

compile-file

Summary
Compile a Lisp source file into a form that both loads and runs faster.

Package
common-lisp

Signature
compile-file input-file &key output-file verbose print external-format load => output-truename, warnings-p, failure-p

Arguments
input-file A pathname designator.
output-file A pathname designator, or :temp.
verbose A generalized boolean.
print A generalized boolean.
external-format An external format specification.
load A generalized boolean.

Values
output-truename A pathname or nil.
warnings-p A generalized boolean.
failure-p 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
3 The COMMON-LISP Package

- 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

```lisp
(compile-file "devel/fred.lisp")
;; compile fred.lisp to fred.fasl
(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

<table>
<thead>
<tr>
<th>Machine/Implementation</th>
<th>Fasl Extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>x86 Windows/32-bit LispWorks</td>
<td>.ofasl</td>
</tr>
<tr>
<td>x64 Windows/64-bit LispWorks</td>
<td>.64ofasl</td>
</tr>
<tr>
<td>x86 Linux/32-bit LispWorks</td>
<td>.ufasl</td>
</tr>
<tr>
<td>amd64 Linux/64-bit LispWorks</td>
<td>.64ufasl</td>
</tr>
<tr>
<td>x86 FreeBSD/32-bit LispWorks</td>
<td>.ffas1</td>
</tr>
<tr>
<td>HP-PA/32-bit LispWorks</td>
<td>.pfas1</td>
</tr>
</tbody>
</table>
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 `.fsl`. This changed in LispWorks 5.0.

In LispWorks for Linux 4.4 and previous, the fasl file extension is `.ufsl`. This changed in LispWorks 5.0.

### See also

- `compile`
- `compile-file-if-needed`
- `*compiler-break-on-error*`
- `disassemble`

### `concatenate`

**Function**

**Summary**

Extends the standard `concatenate` function allowing it to take any Common Lisp type.

**Package**

`common-lisp`
**concatenate**

**Signature**

```lisp
concatenate result-type &rest sequences => result-sequence
```

**Arguments**

- `result-type`: A type specifier.
- `sequences`: A sequence.

**Values**

- `result-sequence`: A sequence.

**Description**

The `concatenate` function has been extended to take any Common Lisp type. The `result-sequence` will be of type `result-type` unless this is not possible, in which case a `type-error` is signalled.

**See also**

- `coerce`

---

**declaim**

**Macro**

**Summary**

Established a specified declarations.

**Package**

`common-lisp`

**Signature**

```lisp
declaim &rest declarations
```

**Arguments**

- `declarations`: Declaration forms.

**Description**

The macro `declaim` behaves as specified in the ANSI Common Lisp Standard with one exception: for a top-level call to `declaim`, 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**

- `compile-file`
- `declare`
- `proclaim`
**declare**

**Special Form**

**Summary**
Declares a variable as special, provides advice to the Common Lisp system, or helps the programmer to optimize code.

**Package**
common-lisp

**Signature**
declare declaration*

**Arguments**
declaration
A declaration specifier, not evaluated.

**Values**
The special form declare 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 let, do, defun and so on.

(Consult the syntax definition of each special form to see if it takes declare forms and/or documentation strings.)

**Description**
There are three distinct uses of declare: 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 :explain declaration) is to help you optimize your code.

If you use declare 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).

The following are extensions to the Common Lisp definition of declare:

- **lambda-list** 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)
```

```
```

The **:explain** declaration controls messages printed by the compiler while it is processing forms. The declaration can be used with **proclaim** or **declaim** 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:
Set value associated with all optionkeys to nil. This turns off all explanations.

If optionvalue is non-nil, list all the variables of each function, specifying whether they are floating point or not.

If optionvalue is non-nil, print information about compiler transformations that depend on declared or deduced type information.

If optionvalue is non-nil, print information about calls to functions that may allocate floats.

If optionvalue is non-nil, print information about calls to functions that may allocate non-float numbers, for example bignums.

If optionvalue is non-nil, print information about calls to normal functions.

If optionvalue is non-nil, print the argument types for calls to normal functions. Must be combined with :all-calls.

A synonym for :all-calls.

If optionvalue is non-nil, print information about calls to functions that may allocate numbers, for example floats or bignums.

If optionvalue is non-nil, modifies the :all-calls, :floats and :non-floats explanations to include the original source code form that contains the call.

This turns off all explanations.

If optionvalue is non-nil, list all the variables of each function, specifying whether they are floating point or not.

If optionvalue is non-nil, print information about compiler transformations that depend on declared or deduced type information.

If optionvalue is non-nil, print information about calls to functions that may allocate floats.

If optionvalue is non-nil, print information about calls to functions that may allocate non-float numbers, for example bignums.

If optionvalue is non-nil, print information about calls to normal functions.

If optionvalue is non-nil, print the argument types for calls to normal functions. Must be combined with :all-calls.

A synonym for :all-calls.

If optionvalue is non-nil, print information about calls to functions that may allocate numbers, for example floats or bignums.

If optionvalue is non-nil, modifies the :all-calls, :floats and :non-floats explanations to include the original source code form that contains the call.
If `optionvalue` is non-nil, modifies the 
`:all-calls`, `:floats` and `:non-floats` 
explanations to include the macroexpanded 
source code form that contains the call.

`:print-length` Use the `optionvalue` as the value of 
`*print-length*` for `:all-calls`, `:floats` 
and `:non-floats` explanations.

`:print-level` Use the `optionvalue` as the value of 
`*print-level*` for `:all-calls`, `:floats` 
and `:non-floats` explanations.

Example

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

See also

`compile`
`compile-file`
`proclaim`

---

**defclass**

*Macro*

**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**
`common-lisp`

**Description**
The macro `defclass` 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 Note**

When a class is redefined, its extra initargs are always reset. 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*)))
NIL

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*)))
NIL
(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 >
defpackage

Summary
Remains as defined in Common Lisp, but see *handle-existing-defpackage* for an extension.

Package
common-lisp

Signature
defpackage defined-package-name [[option]] => package

Arguments
defined-package-name
A string.

option
Keyword options.

add-use-defaults
A keyword

Values
package
A package.

Description
The macro defpackage is as defined in the ANSI standard, with the inclusion of the :add-use-defaults keyword. However, the standard explicitly declines to define what defpackage does if defined-package-name already exists and is in a state that differs from that described by the defpackage 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 option is supported. :add-use-defaults, with a true argument, causes the package defined-package-name to inherit from the following packages (as well as any explicitly specified by the :use option):

- common-lisp
- lispworks

See also
process-a-class-option
process-a-slot-option
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*

\textbf{describe} \hspace{1cm} \textit{Function}

\textbf{Summary} \hspace{1cm} Remains as defined in ANSI Common Lisp. Additionally, you can control the depth at which slots inside arrays, structures and conses are described.

\textbf{Package} \hspace{1cm} \texttt{common-lisp}

\textbf{Signature} \hspace{1cm} \texttt{describe \ object \ &optional \ stream \ => \ <no-values>}

\textbf{Arguments} \hspace{1cm} \texttt{object} \hspace{1cm} An object.
\hspace{1cm} \texttt{stream} \hspace{1cm} An output stream designator.

\textbf{Description} \hspace{1cm} The function \texttt{describe} displays information about the object \texttt{object} to the stream indicated by \texttt{stream}, as specified in ANSI Common Lisp.

Arrays, structures and conses are \texttt{described} recursively up to the depth given in the value of the variable \texttt{*describe-level*}. Beyond that depth, objects are simply printed.

\textbf{See also} \hspace{1cm} \texttt{*describe-length*}
\hspace{1cm} \texttt{*describe-level*}
\hspace{1cm} \texttt{*describe-print-length*}
\hspace{1cm} \texttt{*describe-print-level*}
**directory**

*Function*

**Summary**
Determines which files on the system have names matching a given pathname.

**Package**
common-lisp

**Signature**
directory pathname &key test directories flat-file-namestring link-transparency non-existent-link-destinations => pathnames

**Arguments**

- **pathname**
  A pathname, string, or file-stream.

- **test**
  Filtering test (only pathnames matching the test are collected).

- **directories**
  A boolean controlling whether non-matching directories are included in the result.

- **flat-file-namestring**
  A generalized boolean.

- **link-transparency**
  If nil, then symbolic links are not followed.
  This means that returned names are not necessarily truenames, but has the useful feature that the pathname-directory of each pathname returned is the directory supplied as argument.
  The default value of link-transparency is given by the special variable, *directory-link-transparency*, which has initial value t on UNIX/Linux/Mac OS X. By setting this variable to nil, you can get the old behavior of directory. On Windows, where the file system does not normally support symbolic links, this variable is initially nil.
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

pathnames

A list of physical pathnames.

Description
directory collects all the pathnames matching the given pathname.

directory returns truenames, conforming to the ANSI specification for Common Lisp. Some programs may depend on the old behavior, however (and directory 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: link-transparency and non-existent-link-destinations.

Because truenames are now returned, the entries . and .. no longer show up in the output of directory. This means, for instance, that

(directory #P"/usr/users/")

does not include #P"/usr", which is the truename of #P"/usr/users/.."

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 subdirectories 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 :check-for-subs 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 directory 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`

### disassemble

**Function**

**Summary**

Prints the machine code of a compiled function.

**Package**

`common-lisp`

**Signature**

```
disassemble name-or-function => nil
```

**Arguments**

`name-or-function` 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 `*standard-output*`.

On UNIX and Mac OS X, the number of instructions in the disassembly is also printed, at the end.

If the function denoted by `name-or-function` is not compiled then it is first compiled using the function `compile`. This happens if `name-or-function` is a lambda expression or a symbol naming an interpreted function.

An error is signalled if `name-or-function` is not suitable.

**Examples**

```
(disassemble #'(lambda (x) (progn x)))
(disassemble 'cons)
(disassemble #'map)
```

**Notes**

The output from `disassemble` 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

Generic Function

Summary
Returns the documentation string if available.

Package
common-lisp

Signature

documentation x doc-type => documentation
(setf documentation) new-value x doc-type => new-value

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

Type

Summary
A subtype of float.

Package
common-lisp

Signature
double-float

Description
double-float is disjoint from short-float and single-float 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 double-float. However, there are distinct specialized array types (array single-float), with single precision, and (array double-float), with double precision.

See also
long-float
parse-float
short-float
single-float

*features*

Variable
Summary
The features list.
3 The COMMON-LISP Package

Package common-lisp

Initial Value A list containing :lispworks. 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 LispWorks implementation.

:solaris2 Solaris2
:hp-ux HP-UX
:svr4 System 5 Release 4 machine (for example Solaris2)
:linux Linux
:darwin The variant of FreeBSD underlying Mac OS X.
:unix Unix, including all of the above.
:mswindows Microsoft Windows, including 32-bit and 64-bit.
:lispworks-64bit 64-bit LispWorks.
:x86 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.

:amd64, :x86-64, :x64
Images that run on the amd64/x86_64/x64 architecture have each of these features. 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:

```lisp
#+(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.
Every LispWorks 5 image has exactly one of the features :lispworks-32bit and :lispworks-64bit.

For :sparc, :powerpc and :mswindows, there 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**

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

**Package**
common-lisp
The predicate `input-stream-p` is implemented as a generic function. The default method returns `t` if `stream` is an input stream. If the user wants to implement a stream with no inherent directionality (and thus does not include `fundamental-input-stream` or `fundamental-output-stream`) but for which the directionality depends on the instance, then a method should be provided for `input-stream-p`.

See also `fundamental-input-stream` `output-stream-p`

---

**interactive-stream-p**

A generic function that determines if an object is an interactive stream.

- **Package**: `cl`
- **Signature**: `interactive-stream-p stream => bool`
- **Arguments**: `stream` A stream.
- **Values**: `bool` A generalized boolean.
- **Description**: The predicate `interactive-stream-p` is implemented as a generic function. The `fundamental-stream` class provides a default method that returns `nil`.
- **See also**: `input-stream-p` `output-stream-p`
3 The COMMON-LISP Package

**load-logical-pathname-translations**

*Function*

Summary

Searches for and loads the definition of a logical host, if not already defined.

Package

`cl`

Signature

`load-logical-pathname-translations host => just-loaded`

Arguments

`host`  
A logical host, expressed as a string.

Values

`just-loaded`  
A generalized boolean

Description

This function loads the translations for `host` by loading the file `host.lisp` from the LispWorks directory `translations`.

Example

```
(load-logical-pathname-translations "EDITOR-SRC")
```

**long-float**

*Type*

Summary

A subtype of `float`.

Package

`common-lisp`

Signature

`long-float`

Description

`long-float` is the same type as `double-float` in LispWorks, on all platforms.

See also

`double-float`
`parse-float`
`short-float`
`single-float`
**long-site-name**

*Function*

**Summary**
Identifies the physical location of the computer.

**Package**
common-lisp

**Signature**

long-site-name => description

(setf long-site-name) description => description

**Arguments**
description A string or nil.

**Description**
The function long-site-name returns a string identifying the physical location of the computer. This should be set using (setf long-site-name) when you configure your Lisp-Works image.

**See also**
short-site-name

---

**loop**

*Macro*

**Summary**
A macro that performs iteration.

**Package**
cl

**Signature**

loop {for|as} var [type-spec]

being {the|each}{records|record}

(in|of) query-expression => result

**Arguments**
var A variable.

query-expression An SQL query-statement

**Values**
result An object.

**Description**
The Common Lisp loop 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-
Each iteration of the loop assigns the next record of the table to the variable \textit{var}. The record is represented in Lisp as a list. Destructuring can be used in \textit{var} to bind variables to specific attributes of the records resulting from \textit{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.

\textbf{Example}

This extended \texttt{loop} example, on each record returned as a result of the query, binds \texttt{name}, 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.

\begin{verbatim}
(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)))
\end{verbatim}
Function

make-array

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 non-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 non-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.
nil  Same meaning as :new. This is the default value.
:static  Allocate the array in a static segment.
:long-lived  Allocate the array assuming it is going to be long-lived.
:old  Same meaning as :long-lived. A fixnum n  Allocate the array in generation n.

Arrays (including strings) that are passed by address to foreign functions must be static, and so must should be created with :allocation :static.

Allocation with :old or :long-lived 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**  common-lisp

**Signature**  
make-hash-table &key test size rehash-size rehash-threshold hash-function weak-kind => hash-table

**Arguments**

test  A designator for a function of two arguments, which returns t if they should be regarded as the same and nil otherwise.

hash-function  A designator for a function of one argument, which returns a hash value.
weak-kind One of :value, :key, :both, :one, :either, nil. The default is nil.

Description
The standard definition of make-hash-table is extended such that test can be any suitable user-defined function, except that it must not call process-wait or similar mp package functions which suspend the current process. If test is not one of the standard test functions (eq, eql, equal and equalp), and if hash-function is not supplied, then the hash value is the same as would be used if test were equalp.

hash-function may be supplied only if test 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.

If weak-kind is non-nil, it makes hash-table weak. Its semantics are the same as the second argument of set-hash-table-weak, that is:

(make-hash-table :weak-kind weak-kind <other-args>)

is equivalent to

(let ((ht (make-hash-table <other-args>)))
  (set-hash-table-weak ht weak-kind)
  ht)

See also set-hash-table-weak

make-instance

Generic Function

Summary Creates and returns a new instance of a class.

Package common-lisp

Signature make-instance class &rest initargs &key &allow-other-keys => instance

Arguments class A class, or a symbol that names a class.
### initargs
- **An initialization argument list.**

### Values
- **instance** - A fresh instance of class *class*.

### Description
- **make-instance** behaves as specified in ANSI Common Lisp. In particular it checks the initialization arguments as calculated by *compute-class-potential-initargs*.
- This check can be suppressed by passing `:allow-other-keys t`. In addition, LispWorks provides global control over the initarg checking via `set-make-instance-argument-checking` and per-class control via `class-extra-initargs`.

**Note:** in a delivered image, **make-instance** does not check the initialization arguments.

### Compatibility
- **Note** - In LispWorks 4.2 and previous versions, **make-instance** does not check the initargs. If your code contains invalid initargs, you could use one of the techniques mentioned above to resolve it.

### See also
- **class-extra-initargs**
- **compute-class-potential-initargs**
- **set-make-instance-argument-checking**

---

### make-sequence

#### Function

#### Summary
- Extends the standard **make-sequence** function allowing it to take any type specifier.

#### Package
- **common-lisp**

#### Signature
- `make-sequence result-type size &key initial-element => sequence`

#### Arguments
- **result-type** - A type specifier.
- **size** - A non-negative integer.
initial-element  An object.

Values

sequence  A sequence.

Description

The make-sequence function has been extended to take any Common Lisp type. The sequence will be of type result-type unless this is not possible, in which case a type-error is signalled.

See also

concatenate
map
merge

map

Function

Summary

Redefines the standard map function allowing it to take any type specifier.

Package

common-lisp

Signature

map result-type function &rest sequences => result

Arguments

result-type  A sequence type specifier or nil.
function  A function designator.
sequence  A sequence.

Values

result  A sequence.

Description

The map function has been extended to take any Common Lisp type. The result will be of type result-type unless this is not possible, in which case a type-error is signalled.

See also

concatenate
make-sequence
merge
### merge Function

**Summary**
Redefines the standard `merge` function allowing it to take any type specifier.

**Package**
`common-lisp`

**Signature**
```
merge result-type sequence1 sequence2 predicate &key key => sequence
```

**Arguments**
- `result-type` A type specifier.
- `sequence1` A sequence.
- `sequence2` A sequence.
- `predicate` A designator for a function.
- `key` A designator for a function or `nil`.

**Values**
- `sequence` A sequence.

**Description**
The `merge` function has been extended to take any Common Lisp type. The `sequence` will be of type `result-type` unless this is not possible, in which case a `type-error` is signalled.

**See also**
- `concatenate`
- `make-sequence`
- `map`

---

### open Function

**Summary**
Creates, opens, and returns a file stream that is connected to a specified file.

**Package**
`common-lisp`

**Signature**
```
open filespec &key direction element-type external-format if-exists if-does-not-exist => stream
```
Arguments
filespec  A file designator.
direction  If direction is :probe, external-format is ignored. The element type and external format of the returned stream are undefined.
element-type  By default, the value of *default-character-element-type* (the ANSI standard default is character).
external-format  An external file format designator. By default, this is :default.
if-exists  What to do if the file stream already exists. The possible values for this are as in the ANSI standard.
if-does-not-exist  What to do if the file stream does not already exist. The possible values for this are as in the ANSI standard.

Values
stream  A file stream, or nil.

Description
If external-format has a name which is not :default and the parameters include :eol-style, it is used as is.
Otherwise, the system decides which external format to use via guess-external-format. By default, this finds a match based on the filename; or (if that fails), looks in the EMACS-style (-*) attribute line for an option called encoding or external-format; 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.

After the external-format has been determined, it is verified using valid-external-format-p; and an error is signalled if this check fails.
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

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

Package
common-lisp

Signature
open-stream-p stream => result
Arguments

stream A stream.

Values

result A generalized boolean.

Description

The function open-stream-p is generic. The default method provided by the class fundamental-stream returns \textit{t} if close has not been called on the stream.

See also

close
fundamental-stream

\textbf{output-stream-p} \hspace{1cm} \textit{Generic Function}

Summary

A generic function that determines if an object is an output stream.

Package

common-lisp

Signature

output-stream-p stream => result

Arguments

stream A stream.

Values

result A generalized boolean.

Description

The predicate output-stream-p is implemented as a generic function. The default method returns \textit{t} if \textit{stream} is an output stream. If the user wants to implement a stream with no inherent directionality (and thus does not include fundamental-input-stream or fundamental-output-stream) but for which the directionality depends on the instance, then a method should be provided for output-stream-p.

See also

fundamental-output-stream
input-stream-p
proclaim

Summary
Established a specified declaration in the global environment.

Package
common-lisp

Signature
proclaim declaration-list => nil

Arguments
declaration-list A list of declaration forms to be put into immediate and pervasive effect.

Values
Returns nil.

Description
Unlike declare, proclaim 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:

(proclaim '(optimize (speed 3) (space 0) (debug 0)))

means the rest of the file is compiled with these optimization levels in effect. Other ways of doing this are:

- use the :optimize option in defsystem to establish default optimization qualities for every member of the system, when compiled via compile-system.
- add appropriate declare declarations in every function in the file.

Note: For a top-level call to proclaim or declaim, 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 proclaim (so it is not a top-level call).

See the LispWorks User Guide for a more extended description of the compiler optimize qualities.
Examples

(proclaim '(special *fred*))
(proclaim '(type single-float x y z))
(proclaim '(optimize (safety 0) (speed 3)))

Notes

As proclaim 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.

Remember to quote the argument list if it is a constant list.

(proclaim (special x)) attempts to call function special.

Exercise caution if you declare or proclaim variables to be special without regard to the naming convention that surrounds their names with asterisks.

See also

compile
compile-file
declam
declare

restart-case

Macro

Summary

Evaluates a restartable form in a special dynamic environment.

Package

common-lisp

Signature

restart-case restartable-form {clause} => result*

clause ::= (case-name lambda-list [[:interactive interactive-expression | :test test-expression]]
declaration* form*)

Description

The macro restart-case behaves as specified in the ANSI Common Lisp standard.

In addition to that specification, report-expression may be a form whose car is list. Such a form is evaluated when the
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 \texttt{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.

\textbf{room} \hfill \textit{Function}

\begin{itemize}
\item \textbf{Summary} \hspace{2em} Print information about the state of internal storage and its management.
\item \textbf{Package} \hspace{2em} \texttt{common-lisp}
\item \textbf{Signature} \hspace{2em} \texttt{room \&optional x}
\item \textbf{Arguments} \hspace{2em} \texttt{x} \hspace{2em} One of \texttt{nil}, \texttt{t}, or \texttt{:default}.
\item \textbf{Values} \hspace{2em} \texttt{room} returns no values.
\item \textbf{Description} \hspace{2em} This function provides statistics on the current state of the storage, including the amount of space currently allocated, and the amount available for allocation.
\end{itemize}

As outlined in the Common Lisp Hyperspec, the \texttt{room} function takes an optional argument which controls the level of detail it produces.

Given an argument of \texttt{nil}, 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 `t`, 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
      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 OK,
  Waiting promotion = OK, sweeps
before promotion = 4
  Segment 2004AFA8: Total Size 242K, Allocated
  105K, Free 132K
  minimum free space OK, static
Generation 2: Total Size 24324K, Allocated 20730K,
  Free 3572K
  Segment 222A8548: Total Size 12992K, Allocated
  9527K, Free 3460K
  minimum free space OK,
  Waiting promotion = OK, sweeps
before promotion = 4
  Segment 21D78548: Total Size 4224K, Allocated
  4110K, Free 109K
  minimum free space OK,
  Waiting promotion = OK, sweeps
before promotion = 4
  Segment 21418548: Total Size 2816K, Allocated
  2811K, Free 0K
  minimum free space OK,
  Waiting promotion = OK, sweeps
before promotion = 4
  Segment 217E8548: Total Size 4224K, Allocated
  4218K, Free 1K
  minimum free space OK,
  Waiting promotion = OK, sweeps
before promotion = 4
  Segment 20DBD8C8: Total Size 68K, Allocated
  63K, Free 0K
  minimum free space 117K,
  Waiting promotion = OK, 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,
  Waiting promotion = OK, sweeps
before promotion = 10
  Segment 20DCEE40: Total Size 6437K, Allocated
  6321K, Free 112K
  minimum free space OK,
  Waiting promotion = OK, sweeps
before promotion = 10
  Segment 207177E8: Total Size 6809K, Allocated
The COMMON-LISP Package

3

6805K, Free OK
minimum free space OK, awaiting promotion = 0K, sweeps before promotion = 10
Total Size 50752K, Allocated 43083K, Free 7307K

See also
find-object-size
room-values
total-allocation

short-float

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
**short-site-name**  
*Function*

**Summary**  
Identifies the physical location of the computer.

**Package**  
`common-lisp`

**Signature**  
`short-site-name => description`

`(setf short-site-name) description => description`

**Arguments**  
`description`  
A string or `nil`.

**Description**  
The function `short-site-name` returns a string briefly identifying the physical location of the computer. This should be set using `(setf short-site-name)` when you configure your LispWorks image.

**See also**  
`long-site-name`

---

**simple-base-string**  
*Type*

**Summary**  
The simple base string type.

**Package**  
`common-lisp`

**Signature**  
`simple-base-string length`

**Arguments**  
`length`  
The length of the string (or `*`, meaning any).

**Description**  
The type of simple base strings.

---

**single-float**  
*Type*

**Summary**  
A subtype of `float`.

**Package**  
`common-lisp`
3 The COMMON-LISP Package

**Signature**

`single-float`

**Description**

A `single-float` is an immediate object in 64-bit LispWorks, A `single-float` is a boxed object in 32-bit LispWorks. `single-float` is disjoint from `double-float` in all LispWorks implementations, version 5.0 and later. `single-float` is disjoint from `short-float` in all 32-bit LispWorks implementations in version 5.0 and later. In 64-bit LispWorks `single-float` is the same type as `short-float`.

**Compatibility Note**

In LispWorks 4.4 and previous on Windows and Linux platforms, `single-float` is the same type as `double-float`. However, there are distinct specialized array types `(array single-float)`, with single precision, and `(array double-float)`, with double precision.

**See also**

double-float  
long-float  
parse-float  
short-float

---

**software-type**

*Function*

**Summary**

Identifies the Operating System.

**Package**

`common-lisp`

**Signature**

`software-type => description`

**Values**

`description` A string or `nil`.

**Description**

The function `software-type` returns a string representing a generic name of the Operating System, or `nil` if this cannot be determined.

See also `software-version`

**software-version**  
*Function*

**Summary**  
Identifies the version of the Operating System.

**Package**  
`common-lisp`

**Signature**  
`software-version => description`

**Values**  
`description`  
A string or `nil`.

**Description**  
The function `software-version` returns a string giving the version of the Operating System, or `nil` if this cannot be determined.


**Example**  

```
(software-version)
=>
"Windows Vista: 6.0 (build 6000) "

(software-version)
=>
"Windows XP: 5.1 (build 2600) Service Pack 2"
```
(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 subforms.

: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.
Redo one of the previous commands.

Get an item from the history list and put it in a variable.

List available commands.

Replace one form with another form in previous command and redo it.

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
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) -> t
[2] 1 -> 1
[2] 2 -> 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 `stream-element-type` as a generic function.

**Package**

`common-lisp`

**Signature**

`stream-element-type stream => type`

**Arguments**

`stream` A stream.

**Values**

`type` A type specifier.

**Description**

The function `stream-element-type` 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**

*Type*

**Summary**
The string type.

**Package**
`common-lisp`

**Signature**
`string length element-type`

**Arguments**
- `length` - The length of the string (or *, meaning any).
- `element-type` - The type of string element. The default is the value of `*default-character-element-type*` rather than *.

**Description**
The union of all string types as specified in the standard, but extended with an extra parameter: `element-type`.

`(string length element-type)` means all string types whose element type is a subtype of `element-type`. That is:

- `(string * base-char) = (vector base-char *)`
- `(string * lw:simple-char) = (or (vector base-char *) (vector lw:simple-char *))`
(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 consing in your code **code** you need to do:

```lisp
(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"))`

```
start end
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**

```lisp
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.

: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*).

: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
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,

\begin{verbatim}
(trace (print :entrycond nil 
:exitcond nil 
:allocation $\text{print-allocation}$))
\end{verbatim}

results in $\text{print-allocation}$ containing the sum of the
allocation made inside print.

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

\texttt{: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:

\begin{itemize}
\item \texttt{:quick} Like the \texttt{:bq} debugger command.
\item \texttt{t} Like the \texttt{:b} debugger command.
\item \texttt{:verbose} Like the \texttt{:b :verbose} debugger command.
\item \texttt{:bug-form} Like the \texttt{:bug-form} debugger command.
\item \texttt{: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 \texttt{*traced-
arglist*} then the values received by the body of the func-
tion are changed accordingly (see also \texttt{*traced-
arglist*}).
\item \texttt{:eval-after} and \texttt{:eval-before} are similar to \texttt{:after} and
\texttt{:before}, without output.
\item \texttt{:break} is followed by a form. This is evaluated after printing
the standard information caused by entering the function,
and after executing any \texttt{:before} forms; if it returns \texttt{nil} then
tracing continues normally, otherwise \texttt{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

| trace-result | If trace 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

trace 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 trace 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.

The arguments to trace 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 trace is called with a function that is already being traced, then the new tracing specification for that function replaces the old version.

Note: trace works by tracing function names, not function objects. Therefore tracing function objects, for example by

(trace #'foo)

will not yield any trace output. Also, if the symbol foo is traced, then invoking the function foo by

(funcall (symbol-function 'foo) ...)

will not yield any trace output.
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)
```
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*
**truename**

*Function*

Summary

Returns the truename of a pathname.

Package

`common-lisp`

Signature

`truename filespec => truename`

Arguments

`filespec`  A pathname designator.

Values

`truename`  A fully-specified physical pathname.

Description

The function `truename` 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 `*default-pathname-defaults*`). Note that this means that the paths returned by `directory` are always fully specified.

See also

`directory`

**untrace**

*Macro*

Summary

Turns off the Common Lisp tracing facility on the named functions.

Package

`common-lisp`
Signature \( \text{untrace} \ (\text{function-name} \mid \text{method-desc})^{*} \Rightarrow \text{untrace-list} \)

Arguments

- **function-name**: A symbol whose symbol-function is no longer to be traced.
- **method-desc**: Is a method description, as described in the entry for \textbf{trace}. See \textbf{trace} for more details.

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, \textbf{untrace} returns a list of the symbols of those functions. Thus, in all situations, \textbf{untrace} returns a list of the symbols of those functions being untraced.

Description

\textbf{untrace} 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 \textbf{untrace} is called with a function that is not being traced.

Examples

\begin{verbatim}
USER 12 > (progn (untrace) (trace + - / *))
*
USER 13 > (+ 2 3)
 0 + > (2 3)
 0 + < (5)
 5

USER 14 > (untrace + -)
(*) [/]

USER 15 > (+ 2 3)
 5
\end{verbatim}

To untrace a method:

\begin{verbatim}
(untrace (clos:method foo (t)))
\end{verbatim}
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 `element-type` 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
3 The COMMON-LISP Package
This chapter describes symbols available in the **COMPILER** package.

### `deftransform`  
**Macro**

**Summary**  
Defines a function that computes the expansion of a form.

**Package**  
`compiler`

**Signature**  
```
deftransform name transform-name lambda-list &body body => list-of-transforms
```

**Arguments**

- **`name`**  
  A symbol naming the function to which the transform is to be applied.

- **`transform-name`**  
  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.

- **`lambda-list`**  
  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.

**body**

The body of the expander function, the result of which replaces the original form (unless it evaluates to `compiler::%pass%`, in which case no transformation is applied).

**Values**

*list-of-transforms* A list of the names of transforms defined for the function, including the one just added.

**Description**

`deftransform`, like `defmacro`, defines a function that computes the expansion of a form. Transforms are only used by the compiler and not by the interpreter. `deftransform` allows you to add to the optimizations performed by the compiler.

**Examples**

```
(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

(compiler:deftransform three-list () (constant (list 1 2 3))))
```
;; 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
deftransform differs from defmacro in various ways:

The evaluation of the body can return compiler:%pass% 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).

The compiler only calls the expander function if the arguments match the lambda list — macros are unconditionally expanded.

There can be several deftransforms 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.)

If a transform takes keyword arguments the compiler preserves the correct order of evaluation.

A carelessly written deftransform may lead the compiler to transform valid Common Lisp into incorrect code — there is no semantic checking of the transform.

See also
compile
compile-file
4 The COMPILER Package
This chapter describes symbols available in the `dbg` package, used to configure the debugging information produced by LispWorks.

* `*debug-print-length*`  
  
  **Variable**

  **Summary**  
  Controls the number of object components printed in debugger output.

  **Package**  
  `dbg`

  **Initial Value**  
  `40`

  **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 `nil` then all the parts of an object are shown.

  **Examples**  
  ```lisp
  USER 83 > (setq dbg:*debug-print-length* 3)
  ```
5 The DBG Package

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*

Variable

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* Variable

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

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, 851/905 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 >

Notes

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

*print-binding-frames*

Variable

Summary Controls whether binding 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. Binding frames are formed when special variables are bound, but are normally not shown by the debug-
ger. However if the value of \texttt{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)
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 catch 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 *print-catch-frames* is set to nil then the catch frames are no longer shown.
**Examples**

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 :

**Notes**

*print-catch-frames* is an extension to Common Lisp.

**print-handler-frames**

**Variable**

**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 LispWorks User Guide), 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
5 The DBG Package

Initial Value nil

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 *print-open-frames* is set to t then the open frames are displayed. It is unlikely that you need to examine open frames: their use is connected with implementation details.

Examples 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 :

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

*print-restart-frames* Variable

Summary Controls whether restart 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. Restart frames are formed when restarts are established (see the LispWorks User Guide), but are normally not shown by the debugger. However if *print-restart-frames* is set to t 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
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.

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 multiprocessing activities.
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
This chapter describes symbols available in the DSPEC package.

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

*active-finders*  

**Variable**

**Summary** Controls how source finding operates.

**Package** dspec

**Initial Value** (:internal)

**Description** The *active-finders* variable controls how the functions find-name-locations and find-dspec-locations operate. This in turn controls 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 *active-finders* are:
The internal database of definitions performed in this image.

Prompt for a tags file, when first used.

Either a tags file or a tags database.

A tags database is a fast file generated by `save-tags-database`.

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`

---

**at-location**

**Macro**

**Summary**

Tells the dspec system of the source location.

**Package**

`dspec`

**Signature**

`at-location (location) &body body => result`

**Arguments**

`location` A pathname or a keyword.

`body` Forms, including defining forms.

**Values**

`result` The result of `body`.

**Description**

The macro `at-location` informs the dspec system that the source for definitions done during the execution of `body` are at the location `location`. 
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

CL-USER 12 > (dspec:canonicalize-dspec 'foo)
(FUNCTION FOO)

CL-USER 13 > (dspec:canonicalize-dspec '(defmethod bar
(list t)))
(METHOD BAR (LIST T))

See also

define-dspec-alias

---

def

Macro

Summary

Informs the system of a name for a definition.

Package

dspec

Signature

def dspec &body body => result

Arguments

dspec A dspec.
body Lisp forms, evaluated as an implicit progn.

Values

result The result of body.

Description

The macro def informs the system that any definitions within
body should be recorded as being within the dspec dspec. This
means that when something attempts to locate such a
definition, it should look for a definition named dspec.

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.

dspec can be non-canonical.

You can also use def to provide a dspec for a definition that
has its own class that has been defined with define-dspec-
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

```
(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.
The macro `define-dspec-alias` works rather like `deftype`. Dspecs whose `car` is `name` should have parameters that match `lambda-list`. They will be canonicalized into the dspec returned by `body`.

`define-dspec-alias` 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. `define-dspec-alias` is used to convert one of them to the other during canonicalization by `canonicalize-dspec`.

Example

`defparameter` is pre-defined as an alias for `defvar`.

See also

`canonicalize-dspec`

---

**define-dspec-class**

Macro

Summary

Defines a dspec class.

Package

`dspec`

Signature

```
define-dspec-class name superspace documentation &key pretty-name undefiner canonicalize prettify definedp object-dspec defined-parts aggregate-class
```

Arguments

- `name`: A symbol naming the dspec class
- `superspace`: A symbol naming the superspace
- `documentation`: A string describing the dspec class
- `undefiner`: A function that generates the undefining form for the class
- `canonicalize`: A function to canonicalize a dspec if it belongs to the class
prettify A function to return a prettier form of a dspec of the class

defindedp A function to decide if a dspec of the class currently has a definition

object-dspec A function to return the dspec from an object if it was defined by the class

defined-parts A function to return all the currently defined parts in the class for a given a primary-name

aggregate-class The aggregate dspec class for a part dspec

Description

The macro define-dspec-class defines a dspec class, providing handlers for definitions in that dspec class.

define-dspec-class defines name as a dspec class, inheriting from the dspec class superspace. superspace should be nil to define a new top-level dspec class.

documentation should be a string documenting the dspec class. For example "My Objects".

After evaluating a define-dspec-class form, name can be used by defining forms to record locations of definitions of that dspec class name by calling record-definition.

All of the remaining arguments described below can be omitted if not needed. The most important arguments for the LispWorks IDE are definedp and undefiner.

If undefiner 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 name. 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 undefiner is omitted, then definitions of this class cannot be undefined.

If canonicalize 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
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 func-
tion returns the dspec if it matches the form

(dspect-class symbol)

If prettify is given, its value must be a function of one argu-
ment. 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 argu-
ment. 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 pri-
mary 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 cre-
ated 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 oth-
erwise. 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 `cl: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 definer.

**Package** dspec

**Signature**

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

**Arguments**

`definer-and-options`
A symbol `definer` naming a definer of functions, macros, variables and so on, or a list (`definer options`) where `options` is a plist of keys and values.

**Parameters**
- `nil`, or list of parameters `params` in the top level form, optionally ending with `&rest param-getter`.

**Body**
- The body of a parser function.

**Values**
- `parser` A form parser function.

**Description**

The macro `define-form-parser` defines a form parser for forms beginning with `definer`.

`options` is a property list with the following keys allowed:

- `:parser` A parser function `parser-function`.
- `:alias` A dspec class or alias `alias`.
- `:anonymous` A boolean.

The parser function defined is named by `parser-function`. 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 `definer` with `"-FORM-PARSER"` appended.

If `parameters` and `body` are given, then `parser-function` is defined as a global function that is expected to return a dspec for the defining form or `nil` if this is not possible. Within `body`, `definer` is bound to the `car` of the actual form being parsed. In simple cases, this is just `definer`, 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.

The `params` 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.
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:

```lisp
(dspre:define-form-parser def-foo (name)
  `(,def-foo ,name))
```

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

```lisp
(dspre: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:

(dsparse: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))

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

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

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

A simpler way to write the last form is:

(dspec:define-form-parser (my-defmethod (:alias defmethod)))

See also

get-form-parser
parse-form-dspec
**dspec-class**

*Function*

Summary
Returns the dspec class of a dspec.

Package
`dspec`

Signature
`dspec-class dspec => class`

Arguments
`dspec` A dspec.

Values
`class` A dspec class name.

Description
The function `dspec-class` returns the dspec class name for `dspec`.

Example
```
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
```

See also
`dspec-name`

**dspec-classes**

*Variable*

Summary
Lists all the dspec classes.

Package
`dspec`

Signature
`*dspec-classes*`

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

**dspec-definition-locations**  
*Function*  
Summary: Returns the locations of the known definitions.  
Package: dspec  
Signature:  
dspec-definition-locations  
dspec => locations  
Arguments:  
dspec  
A dspec.  
Values:  
locations  
A list of pairs (recorded-dspec location).  
Description: The function dspec-definition-locations returns the locations of the definitions recorded for the dspec dspec.
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`.
The DSPEC Package

Example

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

**dspec-name**

*Function*

Summary

Extracts the name from a canonical dspec.

Package

dspec

Signature

dspec-name dspec => name

Arguments

dspec  
A canonical dspec.

Values

name  
A dspec name.

Description

The function `dspec-name` extracts the name from the canonical dspec `dspec`.

Note that for part classes this is a list starting with the primary name.

If `dspec` is not canonicalized, `dspec-name` signals an error.

See also

dspec-class

**dspec-primary-name**

*Function*

Summary

Extracts the primary name from a canonical dspec.

Package

dspec

Signature

dspec-primary-name dspec => name

Arguments

dspec  
A canonical dspec.

Values

name  
A dspec name.
The function `dspec-primary-name` extracts the primary name from the canonical `dspec`.

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 `dspec-class`

### `dspec-progenitor`  

**Function**

**Summary**  
Returns the ultimate parent of a `subfunction` `dspec`.

**Signature**  
`dspec-progenitor ` `dspec` `=>` `result`

**Package**  
`dspec`

**Arguments**  
`dspec`  
A `dspec`.

**Values**  
`result`  
A `dspec`.

**Description**  
The function `dspec-progenitor` returns a `dspec` `result` which is the ultimate parent of a `subfunction` `dspec` argument `dspec`.

If the argument `dspec` is not a local `dspec`, it is simply returned.

Note that `result` is not necessarily a canonical `dspec`.

**Example**  

```
(dspec-progenitor
 ' (subfunction 1 (subfunction (flet a) (defun foo)))
) =>
 (defun foo)
```

See also `local-dspec-p`
dspec-subclass-p  

Function

Summary  Tests whether one dspec class is a subclass of another.

Package  dspec

Signature  dspec-subclass-p class1 class2 => result

Arguments  class1, class2  Symbols naming dspec classes.

Values  result  A boolean.

Description  The function dspec-subclass-p determines whether the dspec class denoted by class1 is a subclass of that denoted by class2.

Example  CL-USER 55 > (dspec:dspec-subclass-p 'defmacro 'type)  NIL

          CL-USER 56 > (dspec:dspec-subclass-p 'defmacro 'function)  T


dspec-undefiner  

Function

Summary  Returns an undefining expression for a dspec.

Package  dspec

Signature  dspec-undefiner dspec => form

Arguments  dspec  A dspec.

Values  form  A Lisp form.

Description  The function dspec-undefiner returns a form which would undefine dspec, whether or not dspec is currently defined.
If no such form can be constructed, \texttt{nil} is returned.

Example

\begin{verbatim}
CL-USER 66 > (dspec:dspec-undefiner '(defun foo))
(PROGN (FMAKUNBOUND (QUOTE FOO)) (SETF (DOCUMENTATION (QUOTE FOO) (QUOTE FUNCTION)) NIL))
\end{verbatim}

\textbf{discard-source-info} \hspace{1cm} \textit{Function}

\begin{itemize}
\item \textbf{Summary}: Clears the internal dspec database.
\item \textbf{Package}: dspec
\item \textbf{Signature}: \texttt{discard-source-info} \Rightarrow \texttt{nil}
\item \textbf{Arguments}: None.
\item \textbf{Values}: Returns \texttt{nil}.
\item \textbf{Description}: The function \texttt{discard-source-info} removes all source location information from the internal dspec database.
\item \textbf{Example}: To build \texttt{my-image} which does not contain source locations for the definitions loaded, but retaining a tags database of those definitions:

\begin{verbatim}
(load-all-patches)
(load "my-code")
(dspec:save-tags-database #P"my-tags-database.ofasl")
(dspec:discard-source-info)
(save-image "my-image")
\end{verbatim}
\item \textbf{See also}: \texttt{save-tags-database}
\end{itemize}

\textbf{find-dspec-locations} \hspace{1cm} \textit{Function}

\begin{itemize}
\item \textbf{Summary}: Returns the locations of the definitions of a dspec.
\end{itemize}
Package dspec

Signature find-dspec-locations  dspec => locations

Arguments dspec  A dspec.

Values locations  A list of pairs (recorded-dspec location).

Description The function find-dspec-locations returns the locations of the relevant definitions for the dspec dspec.

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.

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

The location information is collected from all finders on *active-finders*, that is, the relevant definitions are those known to at least one of these finders.

If two or more finders return the same pair (recorded-dspec location), as compared by dspec-equal and location equality, then only the first occurrence of the pair (in the order of *active-finders*) appears in locations.

See also *active-finders*
dspec-definition-locations
dspec-equal

find-name-locations

Function

Summary Returns the locations of the definitions of a name.

Package dspec

Signature find-name-locations  classes name => locations
### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>classes</td>
<td>A list of dspec class names.</td>
</tr>
<tr>
<td>name</td>
<td>A name.</td>
</tr>
</tbody>
</table>

### Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>locations</td>
<td>A list of pairs (recorded-dspec location).</td>
</tr>
</tbody>
</table>

### Description

The function `find-name-locations` returns the locations of the relevant definitions for `name` in the classes listed in `classes`.

For each known definition `recorded-dspec` names the definition that defined `name` in `location`, and `location` is a pathname or keyword as described in `at-location`.

The location information is collected from all finders on `*active-finders*`, that is, the relevant definitions are those known to at least one of these finders.

If two or more finders return the same pair `(recorded-dspec location)`, as compared by `dspec-equal` and location equality, then only the first occurrence of the pair (in the order of `*active-finders*`) appears in `locations`.

### See also

- `*active-finders*`
- `name-definition-locations`
- `dspec-equal`

---

### get-form-parser

**Function**

<table>
<thead>
<tr>
<th>Summary</th>
<th>Returns the form parser associated with a definer.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package</td>
<td>dspec</td>
</tr>
<tr>
<td>Signature</td>
<td><code>get-form-parser definer =&gt; parser</code></td>
</tr>
<tr>
<td>Arguments</td>
<td>definer A symbol naming a definer.</td>
</tr>
<tr>
<td>Values</td>
<td>parser A form parser function, or <code>nil</code>.</td>
</tr>
</tbody>
</table>
Description

The function `get-form-parser` returns a form parser function if there is one associated with `definer`.

This is the case for predefined definers and for those for which you have established a form parser using `define-form-parser`.

If there is no associated form parser, `nil` is returned.

Example

```
CL-USER 1 > dspec:get-form-parser 'defun
DSPEC:NAME-ONLY-FORM-PARSER
```

See also `define-form-parser` `parse-form-dspec`

**local-dspec-p**

*Function*

Summary

The predicate for local dspecs.

Package `dspec`

Signature

`local-dspec-p dspec => localp`

Arguments

`dspec` A dspec.

Values `localp` A boolean.

Description

The function `local-dspec-p` determines whether the dspec `dspec` is a local dspec.

Local dspecs name local definitions, such as local functions. Currently a local dspec is a list whose `car` is `subfunction`.

See also `dspec-progenitor`
**location**

*Macro*

**Summary**
Returns the source location.

**Package**
dspec

**Signature**
`location => location`

**Values**

| location | A pathname or a keyword. |

**Description**
The macro `location` returns a location suitable for passing to `record-definition`. This is usually done via a separate defining function. You will need to use `location` only if you create your own ways of making definitions (and not if your definers call only system-provided definers).

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

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

**See also**

- `at-location`
- `def`

---

**name-defined-dspecs**

*Function*

**Summary**
Returns defined dspecs matching a name.

**Package**
dspec

**Signature**
`name-defined-dspecs classes name => dspecs`

**Arguments**

| classes | A list of dspec class names. |
The function **name-defined-dspecs** looks in each of the dspec classes \textit{classes} for definitions of \textit{name}.

For each definition found (as if by \textit{dspec-defined-p}), the result \textit{dspecs} contains the canonical dspec.

\textbf{See also} \textit{dspec-defined-p}

### name-definition-locations

\textit{Function}

\textbf{Summary} Returns the locations of the known definitions.

\textbf{Package} \textit{dspec}

\textbf{Signature} \texttt{name-definition-locations \textit{classes name => locations}}

\textbf{Arguments} \textit{classes} A list of dspec class names.  
\textit{name} A name.

\textbf{Values} \textit{locations} A list of pairs (\textit{recorded-dspec location}).

\textbf{Description} The function \texttt{name-definition-locations} returns the locations of the definitions recorded for the name \textit{name} in any of the dspec classes in \textit{classes}.

For each known definition \textit{recorded-dspec} names the definition that defined \textit{name} in \textit{location}, and \textit{location} is a pathname or keyword as described in \texttt{at-location}.

\textbf{Example} \begin{verbatim}
CL-USER 7 > (dspec:name-definition-locations '(function) 'foo-bar)
(((DEFSTRUCT FOO) #P"C:/temp/hack.lisp")
\end{verbatim}
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))

(dspe:define-form-parser
  (my-definer (:parser
dspe:name-only-form-parser)))

See also  define-form-parser
The DSPEC Package

**parse-form-dspec**  
Function  
Summary  
Parses the dspec from a defining form.  
Package  
dspec  
Signature  
`parse-form-dspec form => result`  
Arguments  
form  
A form.  
Values  
result  
A dspec or nil.  
Description  
The function `parse-form-dspec` invokes the defined form parser for `form` and returns the resulting dspec.  
Example  
```
(parse-form-dspec '(def-foo my-foo (arg) (foo-it arg)))
```

```
=>
(def-foo my-foo)
```

See also  
define-form-parser  
get-form-parser  
record-definition  
Function  
Summary  
Checks for existing definitions and records a new definition.  
Package  
dspec  
Signature  
`record-definition dspec location &key check-redefinition-p => result`  
Arguments  
A dspec.  
A pathnamen or keyword.  
check-redefinition-p  
A boolean.
Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>result</td>
<td>A generalised boolean.</td>
</tr>
</tbody>
</table>

Description

The function `record-definition` tells the system that `dspec` is defined at `location`.

The system-provided definer macros call the function `record-definition` with the current location.

`location` should be a pathname or keyword as returned by `location`.

When `check-redefinition-p` is true, it checks for existing definitions according to the value of `*redefinition-action*`. The default value of `check-redefinition-p` is `t`.

If the definition is made, then `result` is true. If the definition is not made then `result` 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*

<table>
<thead>
<tr>
<th>Summary</th>
<th>Package</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controls whether the locations of definitions are recorded.</td>
<td><code>dspec</code></td>
</tr>
</tbody>
</table>
The variable *record-source-files* controls whether locations of definitions are recorded in the internal tags database.

*record-source-files* 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 *active-finders*

---

**redefinition-action**

Variable

Summary Specifies the action on some redefinitions.

Package dspec

Initial value :warn

Description *redefinition-action* specifies what action should be taken when methods and other forms are redefined. If *redefinition-action* is set to :warn then you are warned. If it is set to :quiet or nil, the redefinition is done quietly. If, however, it is set to :error, then LispWorks signals an error.

*redefinition-action* 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 *handle-warn-on-redefinition*
save-tags-database  

**Function**

Summary  Saves the current internal dspec database to a given file.

Package  dspec

Signature  

```
save-tags-database pathname => pathname
```

Arguments  

```
pathname
```

A filename.

Values  

```
pathname
```

The filename that was supplied.

Description  

The `save-tags-database` function saves the current internal dspec database into the file given by `pathname`. The file can then be used in the variable `*active-finders*`.

See also  

`*active-finders*`

---

single-form-form-parser  

**Function**

Summary  A pre-defined form parser.

Package  dspec

Signature  

```
single-form-form-parser top-level-form getter => dspec
```

Arguments  

```
top-level-form
getter
```

A top level defining form.

The subform getter function.

Values  

```
dspec
```

A dspec.

Description  

The function `single-form-form-parser` is a predefined form parser for use with `define-form-parser`. 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 def-var, 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

single-form-with-options-form-parser Function

Summary A pre-defined form parser.

Package dspec

Signature single-form-with-options-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 single-form-with-options-form-parser is a predefined form parser for use with define-form-parser. 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.


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, accord-
ing to dspec-defined-p. The result does not depend on
whether *dspec* is currently traced.
Example

CL-USER 67 > (dspec:traceable-dspec-p '(subfunction foo bar))
NIL

CL-USER 68 > (dspec:traceable-dspec-p '(defun open))
OPEN

tracing-enabled-p

Function

Summary

Gets and sets the global tracing state.

Package
dspec

Signature

tracing-enabled-p => enabledp

(setf tracing-enabled-p) enabledp => enabledp

Values

enabledp A generalized boolean.

Description

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.

The function (setf tracing-enabled-p) switches tracing on or off according to the value of enabledp. This does not affect the list of functions that are currently traced.

See also

trace
tracing-state

tracing-state

Function

Summary

Gets the current trace details.

Package
dspec

Signature

tracing-state &optional dspec => state
### Signature

\[
\text{(setf tracing-state) state &optional dspec => state}
\]

### Arguments

- **dspec**: A dspec.

### Values

- **state**: A list.

### Description

The function `tracing-state` returns a listing describing the current state of the tracing system. It shows the current tracing state for the dspec `dspec`, or for all traced definitions if `dspec` is not supplied.

The result `state` 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 `tracing-state` returns more information than is returned by `trace`. It is useful for preserving a complex set of traces.

The function `(setf tracing-state)` sets the state of the tracing system. It changes the current tracing state for the dspec `dspec`, or for all traced definitions if `dspec` is not supplied.

`(setf tracing-state)` can be used to switch between different sets of traces. Note however that turning tracing on or off is better done using `tracing-enabled-p`.

### See also

- `trace`
- `tracing-enabled-p`
6 The DSPEC Package
This chapter describes symbols available in the `EXTERNAL-FORMAT` package.

**char-external-code**  
*Function*

<table>
<thead>
<tr>
<th>Summary</th>
<th>Returns the code of a character in the specified character set.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package</td>
<td><code>external-format</code></td>
</tr>
<tr>
<td>Signature</td>
<td><code>char-external-code char set =&gt; code</code></td>
</tr>
<tr>
<td>Arguments</td>
<td><code>char</code> The character whose code you wish to return.</td>
</tr>
<tr>
<td>Values</td>
<td><code>code</code> The code of <code>char</code> in the character set <code>set</code>. An integer.</td>
</tr>
</tbody>
</table>
Description
Returns the code of the character \texttt{char} in the coded character set specified by \texttt{set}, or \texttt{nil}, if there is no encoding. Note that a coded character set is not the same thing as an external format.

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

\begin{verbatim}
(+ (* 100 row) column)
\end{verbatim}

\texttt{:euc-jp} is the complete two-byte format encoded as

\begin{verbatim}
(+ (* 256 first-byte) second-byte)
\end{verbatim}

\texttt{: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 \texttt{find-external-char}

\textbf{decode-external-string} \hfill \textit{Function}

\textbf{Summary}
Decodes a binary vector to make a string.

\textbf{Package} \texttt{external-format}

\textbf{Signature}
\texttt{decode-external-string vector external-format &key start end => string}

\textbf{Arguments}
- \texttt{vector} A binary vector.
- \texttt{external-format} An external format spec.
- \texttt{start, end} Bounding index designators of \texttt{vector}.

\textbf{Values}
- \texttt{string} A string.
**Description**  
The function `decode-lisp-string` decodes the integers in the part of the vector `vector` bounded by `start` and `end` using encoding `external-format` to make a string `string`.

The element type of `vector` does not need to match the `external-format-foreign-type` of `external-format`.

**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**  
`encode-lisp-string`

---

**encode-lisp-string**  

*Function*

**Summary**  
Converts a string to an encoded binary vector.

**Package**  
`external-format`

**Signature**  

```
encode-lisp-string string external-format &key start end => vector
```

**Arguments**  

- `string`  
  A string.
- `external-format`  
  An external format spec.
- `start, end`  
  Bounding index designators of `string`.

**Values**  

- `vector`  
  A binary vector.

**Description**  
The function `encode-lisp-string` converts the part of `string` bounded by `start` and `end` to a binary vector `vector` encoded in encoding `external-format`.

The element type of `vector` matches the `external-format-foreign-type` of `external-format`. 
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  

A type specifier describing the integer types handled by `external-format`.

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  
`external-format-type`

`external-format-type` Function

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

Package  
`external-format`

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

Arguments  
`external-format`  
An external character format.

Values  
A type specifier describing the character types handled by `external-format`.

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  
`external-format-foreign-type`

`find-external-char` Function

Summary  
Returns the character of a given code in a specified character set.

Package  
`external-format`
Signature  find-external-char code set => char

Arguments  
  code A character code. This is an integer.

Values  char The character represented by code. If code is not a legal code in the specified set, the return value is undefined.

Description  Returns the character that has the code code (an integer) in the coded character set specified by set, or nil, 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.

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  char-external-code

valid-external-format-p

Function

Summary  Tests whether an external format spec is valid.
Package  external-format

Signature  valid-external-format-p ef-spec &optional env => bool

Arguments  ef-spec An external format spec.  
            env An environment across which the spec should apply.

Values  bool t if ef-spec is a valid spec; nil otherwise.

Description  This predicate tests whether the external format spec given in ef-spec is valid (in the environment env).

Example  (valid-external-format-p '(:Unicode :eol-style :lf))
7 The EXTERNAL-FORMAT Package
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.

### add-special-free-action

**Function**

| Summary | Adds a function to perform a special action during garbage collection. |
| Package | hcl |
| Signature | add-special-free-action function => function-list |
| Arguments | function | A symbol naming a function of one argument. |
| Values | function-list | 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. add-special-free- |
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**
hcl

**Signature**
`allocation-in-gen-num gen-num &body body => result`

**Arguments**
- `gen-num` An integer, which if out of range for a valid generation number is rounded either to the youngest or oldest generation.
  
  If `gen-num` is negative, the specified generation is: the highest generation number + 1 – `gen-num`, so that an argument of –1 specifies the highest generation number.

- `body` The forms to be evaluated while the allocation generation has been temporarily set to `gen-num`.

**Values**
- `result` The result of evaluating `body`.

**Description**
Allocates objects from a specified generation during the extent of the evaluation of the body forms.

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.

**Note:** this macro is implemented only in 32-bit LispWorks. In 64-bit implementations, use `apply-with-allocation-in-gen-num` or the ':allocation' argument to `make-array` instead.
Examples

```lisp
(allocation-in-gen-num
  1
  (setq tab (make-hash-table :size 1200
                              :test 'eq)
       arr (make-array 20)))
```

See also
- apply-with-allocation-in-gen-num
- make-array
- set-default-generation
- get-default-generation
- *symbol-alloc-gen-num*

### avoid-gc

**Function**

**Summary**
Avoids garbage collection if possible in 32-bit LispWorks.

**Package**
hcl

**Signature**
`avoid-gc => previous-results`

**Arguments**
None.

**Values**
The function returns the previous settings of `minimum-for-sweep`, `maximum-overflow` and `minimum-overflow` (see `set-gc-parameters` for details of these.)

**Description**

`avoid-gc` sets various internal parameters so that garbage collection is avoided as far as possible.

This can be useful with non-interactive programs.

If you use `avoid-gc`, use `normal-gc` later to reset the parameters to their default settings.

**Note:** `avoid-gc` 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` to increase the default size of seg-
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       hcl

Signature     block-promotion &body body => result

Arguments     body              Forms executed as an implicit progn.

Values        result            The result of evaluating the final form in body.

Description   The macro block-promotion executes body and prevents promotion of objects into generation 2 during this execution. After body is executed, generations 0 and 1 are collected.

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, (mark-and-sweep 2) will free a large amount of space in generation 2.

block-promotion can be thought of as doing set-promotion-count on generation 1 with an infinite count, for the duration of body.

block-promotion 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 compile-file, to ensure the transient compile-time data gets collected.

block-promotion has global scope and hence may not be useful in an application such as a multi-threaded server. During the execution of body, generation 1 grows to accommodate all the allocated data, which may have some negative effects on the behaviour of the system, in particular on its interactive response.

Note: symbols and process stacks are allocated in generation 2 or 3 (see *symbol-alloc-gen-num*) hence block-promotion cannot prevent these getting into that generation.
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
building-universal-intermediate-p => intermediatep

Arguments
None

Values
intermediatep  A boolean.

Description
The function building-universal-intermediate-p can be
used in a build script to determine if it is being executed to
build one of the architectures of a universal binary.

The return value intermediatep is nil in most cases. It will be t
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

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 *features*. 
On architectures that do not have universal binaries, this function always returns \texttt{nil}.

See also \texttt{save-universal-from-script} \texttt{save-argument-real-p}

**calls-who** \textit{Function}

Summary

Lists functions called by a function.

Package \texttt{hcl}

Signature \texttt{calls-who \hspace{1em} dspec \rightarrow callees}

Arguments

\hspace{1em} \texttt{dspec} \hspace{1em} A dspec.

Values \hspace{1em} \texttt{callees} \hspace{1em} A list.

Description

The function \texttt{calls-who} returns a list of the dspecs naming the functions called by the function named by \texttt{dspec}.

See also the editor commands \texttt{List Callees}, and \texttt{Show Paths From}.

\textbf{Note:} The cross-referencing information used by \texttt{calls-who} is generated when code is compiled with source-level debugging switched on.

Example

\texttt{(calls-who '(method foo (string)))}

See also \texttt{toggle-source-debugging} \texttt{who-calls}

**cd** \textit{Macro}

Summary

Changes the current directory.
Package hcl

Signature \texttt{cd \texttt{optional directory} \texttt{=} \texttt{current-dir}}

Arguments \texttt{directory} A pathname designator specifying the new directory.

Values \texttt{current-dir} A physical pathname.

Description The macro \texttt{cd} changes the current directory to that specified by \texttt{directory}. \texttt{directory} may be an absolute or relative pathname, and defaults to the string 

See also change-directory get-working-directory

\textbf{change-directory} \textit{Function}

Summary Changes the current directory.

Package hcl

Signature \texttt{change-directory \texttt{directory} \texttt{=} \texttt{current-dir}}

Arguments \texttt{directory} A pathname designator specifying the new directory.

Values \texttt{current-dir} A physical pathname.

Description \texttt{change-directory} changes the current directory to that specified by \texttt{directory}. \texttt{directory} may be an absolute or relative pathname.

Use \texttt{get-working-directory} to find the current directory.

See also \texttt{cd} \texttt{get-working-directory}
The HCL Package

check-fragmentation  

**Function**

**Summary**
Provides information about the fragmentation in a generation in 32-bit LispWorks.

**Package**
hcl

**Signature**
check-fragmentation gen-num => total-free, total-small-blocks, total-large-blocks

**Arguments**
gen-num 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**
total-free Total free space in the generation.
total-small-blocks Amount of free space in the generation which is available in blocks of 512 bytes or larger.
total-large-blocks 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 try-move-in-generation.

**Note:** check-fragmentation is implemented only in 32-bit LispWorks. It is not relevant to the Memory Management API in 64-bit implementations, where gen-num-segments-fragmentation-state is available instead.

**See also**
try-compact-in-generation
try-move-in-generation
**clean-down**

*Function*

**Summary**
Frees memory and reduces the size of the image, if possible.

**Package**
hcl

**Signature**
clean-down &optional full => new-size

**Arguments**
full controls whether to operate on the highest generation. The default is t.

**Values**
new-size The new size of the image, after reduction.

**Description**
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.

If full is t, clean-down 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.

If full is nil, clean-down 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.

clean-down may fail to delete empty segments if there are static segments in high address space.

**Note:** in 64-bit LispWorks, clean-down is implemented as if by

(gc-generation 7 :coalesce t)

though you can use gc-generation directly for better control.

**See also**
gc-generation
save-image
clean-generation-0 \hspace{1cm} Function

Summary
Attempts to promote all objects from generation zero into generation one, thereby clearing generation zero, in 32-bit LispWorks.

Package
hcl

Signature
clean-generation-0 => 1

Arguments
None

Values
Returns the value 1.

Description
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.

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 \texttt{allocation-in-gen-num} or \texttt{set-default-generation}.

Note: \texttt{clean-generation-0} 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 \texttt{(gc-generation 0)}.

Example
; allocate lots of non-ephemeral objects
; ...
(clean-generation-0)

See also
\texttt{allocation-in-gen-num}
\texttt{collect-generation-2}
\texttt{collect-highest-generation}
\texttt{expand-generation-1}
\texttt{gc-generation}
\texttt{set-promotion-count}
**collect-generation-2**

**Function**

**Summary**
Controls whether generation 2 is garbage collected in 32-bit LispWorks.

**Package**
hcl

**Signature**
collect-generation-2 on => size

**Arguments**
on
If on is nil, generation 2 is not garbage collected. If on is t, the generation is garbage collected.

**Values**
size
The current size of the image.

**Description**
Controls whether generation 2 is garbage collected. (Generation 2 normally holds long-lived objects created dynamically.)

Note: collect-generation-2 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-blocking-gen-num instead.

**See also**
clean-generation-0
collect-highest-generation
expand-generation-1
set-blocking-gen-num
set-promotion-count

---

**collect-highest-generation**

**Function**

**Summary**
Controls whether the top generation is garbage-collected in 32-bit LispWorks.

**Package**
hcl
Signature

`collect-highest-generation flag`

Arguments

flag If flag is non-nil, the top generation is collected; if flag is any other value, the top generation is not collected. The default is nil.

Values

`collect-highest-generation` returns no values.

Description

Note: `collect-highest-generation` is implemented only in 32-bit LispWorks. It is not relevant to the Memory Management API in 64-bit implementations.

See also

`avoid-gc`
`clean-generation-0`
`collect-generation-2`
`expand-generation-1`
`normal-gc`

*`compiler-break-on-error`* Variable

Summary

Controls whether `compile-file` handles compilation errors.

Package

hcl

Initial Value

nil

Description

If an error occurs during compilation of a form by `compile-file`, an error handler normally causes the compilation of that form to be skipped, and the error is reported later.

When `*compiler-break-on-error*` is non-nil, an error during compilation by `compile-file` is signaled and the debugger is entered.

See also

`compile-file`
**Function**: compile-file-if-needed

**Summary**: Compiles a Lisp source file if it is newer than the corresponding fasl file.

**Package**: hcl

**Signature**: compile-file-if-needed input-pathname &key output-file load &allow-other-keys => output-truename, warnings-p, failure-p

**Arguments**
- **input-pathname**: A pathname designator.
- **output-file**: A pathname designator.
- **load**: A generalized boolean.

**Values**
- **output-truename**: A pathname or nil.
- **warnings-p**: A generalized boolean.
- **failure-p**: A generalized boolean.

**Description**

The function `compile-file-if-needed` compares the file-write-date of the source file named by `input-pathname` with the file-write-date of the appropriate fasl file (as computed by `compile-file-pathname` from `input-pathname` and `output-file`).

If the fasl file does not exist or is older than `input-pathname`, then `compile-file` is called with `input-pathname`, `output-file`, `load` and any other arguments passed., and the values returned are those returned from `compile-file`.

Otherwise, if `load` is true `compile-file-if-needed` loads the fasl file and returns nil, and if `load` is nil it simply returns nil.
Example

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

See also
compile-file

copy-to-weak-simple-vector

Function

Summary
Creates a weak vector with the same contents as the supplied vector.

Package
hcl

Signature
copy-to-weak-simple-vector vector-t => weak-vector

Arguments
vector-t
An array of type (vector t).

Values
weak-vector
A weak array of type (vector t).
The function `copy-to-weak-simple-vector` creates and returns a weak vector with the same contents as the argument `vector-t`.

Apart from the checking of arguments, this is equivalent to:

```
(replace (make-array (length vector-t) :weak t) vector-t)
```

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

See also
- `make-array`
- `set-array-weak`

---

**Function**

### create-universal-binary

**Summary**
Creates a universal binary from two mono-architecture LispWorks images.

**Package**
hcl

**Signature**
```
create-universal-binary target-image src-image1 src-image2 => target-image
```

**Arguments**
- `target-image` A pathname designator.
- `src-image1` A pathname designator.
- `src-image2` A pathname designator.

**Values**
- `target-image` A pathname designator.

**Description**
This function is intended for advanced use. See the function `save-universal-from-script` for a simpler way to create a universal binary.

The function `create-universal-binary` writes a universal binary to the file `target-image` from the saved image files `src-image1` and `src-image2`. The value of `target-image` is returned.
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, my-application-intel and my-application-powerpc, 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 my-application that will run on both kinds of Macintosh:

```
(create-universal-binary "my-application"
    "my-application-intel"
    "my-application-powerpc")
```

**See also**

save-image

save-universal-from-script

---

**current-stack-length**

*Function*

**Summary**

Returns the size of the current stack.

**Package**

hcl

**Signature**

current-stack-length => stack-size
Arguments: None

Values:

- **stack-size**: The current size of the stack, in 32 bit words (in 32-bit implementations) or 64-bit words (in 64-bit implementations).

Example:

```
(current-stack-length) => 16000
```

Compatibility Note:
In LispWorks 4.4 and previous on Windows and Linux platforms, `current-stack-length` was not implemented. This is fixed in LispWorks 5.0 and later.

See also:
- `extend-current-stack`
- `*sg-default-size*`

---

### *default-package-use-list* Variable

**Summary**
List of packages that newly created packages use by default.

**Package**
hcl

**Initial Value**
```
("CL" "LW" "HCL")
```

**Description**
This variable is the default value of the `:use` keyword to `defpackage`, which specifies which existing packages the package being defined inherits from.

---

### *default-profiler-collapse* Variable

**Summary**
Controls collapsing of the profile tree.

**Package**
hcl

**Initial Value**
nil
The variable *default-profiler-collapse* is a boolean indicating whether the profile tree should collapse functions with only one child function. The default value is nil.

See also

print-profile-list
set-up-profiler

*default-profiler-cutoff*  

Variable

Summary  The minimum percentage that the profiler will display in the output tree.

Package  hcl

Initial Value  0

Description  The variable *default-profiler-cutoff* 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

print-profile-list
set-up-profiler

*default-profiler-limit*  

Variable

Summary  The maximum number of lines of output that are printed during profiling.

Package  hcl

Initial Value  100,000,000

Description  *default-profiler-limit* 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  
\textit{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.

\textit{name}  
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  
\textbf{delete-advice} returns \texttt{nil}.

Description  
\textbf{delete-advice} 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 \textbf{defadvice}. 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).

\textbf{remove-advice} is a function, identical in effect to \textbf{delete-advice}, except that you need to quote the arguments.

Notes  
\textbf{delete-advice} is an extension to Common Lisp.

See also  
\textbf{defadvice} 
\textbf{remove-advice}
### *disable-trace*  
**Variable**

<table>
<thead>
<tr>
<th>Summary</th>
<th>Controls tracing.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package</td>
<td>hcl</td>
</tr>
<tr>
<td>Initial Value</td>
<td>nil</td>
</tr>
<tr>
<td>Description</td>
<td><em>disable-trace</em> controls tracing without affecting the tracing state. If it is set to t then tracing is switched off, but this does not call untrace. When the value of <em>disable-trace</em> is restored to nil, tracing continues as before.</td>
</tr>
<tr>
<td>Notes</td>
<td><em>disable-trace</em> is an extension to Common Lisp.</td>
</tr>
<tr>
<td>See also</td>
<td>trace</td>
</tr>
</tbody>
</table>

### dump-form  
**Function**

<table>
<thead>
<tr>
<th>Summary</th>
<th>Dumps selected forms to a stream.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package</td>
<td>hcl</td>
</tr>
<tr>
<td>Signature</td>
<td>dump-form form stream =&gt; nil</td>
</tr>
<tr>
<td>Arguments</td>
<td>form Form to be dumped.</td>
</tr>
<tr>
<td></td>
<td>stream Stream form is to be dumped to.</td>
</tr>
<tr>
<td>Values</td>
<td>Returns nil.</td>
</tr>
<tr>
<td>Description</td>
<td>dump-form is used in conjunction with with-output-to-fasl-file to dump selected forms. A dumped form is evaluated when loaded using load-data-file. See with-output-to-fasl-file for more details.</td>
</tr>
</tbody>
</table>
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

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**  
hcl

**Signature**  
extend-current-stack &optional how-much => size

**Arguments**  
how-much  
What percentage the stack should be extended by. The default is 50.

**Values**  
size  
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:

```
(hcl:extend-current-stack 100)
```

**Compatibility**  
In LispWorks 4.4 and previous on Windows and Linux platforms, extend-current-stack is not implemented. This is fixed in LispWorks 5.0 and later.

**Note**  
See also current-stack-length  
*stack-overflow-behaviour*
**extended-time**

*Macro*

**Summary**
Prints useful timing information, including information on garbage collection (GC) activity.

**Package**
hcl

**Signature**
extended-time &body body

**Arguments**
body
The forms to be timed.

**Description**
The macro extended-time runs the forms in body. It then prints a summary of the time taken followed by a breakdown of time spent in the GC.

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.

In 32-bit LispWorks these rows begin:

*main promote* indicates promotions from generation 0.

*internal promote*
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.

*fixup* is a part of the compaction and promotion process.

In 64-bit LispWorks these rows begin:

*Standard gen-num (n calls)*
indicates n Standard GCs (includes automatic GCs and calls to gc-generation) in which the highest generation collected was gen-num.

*Marking gen-num (n calls)*
indicates \( n \) Marking GCs (includes calls to \texttt{marking-gc}) in which the highest generation collected was \texttt{gen-num}.

Thus in the example below

\textbf{Standard 1 (6 calls) ...}

indicates that there were 6 Standard GCs in which the highest generation collected was 1.

\textbf{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 2300075
```

\begin{verbatim}
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
\end{verbatim}

This example illustrates output in 64-bit LispWorks:
Timing the evaluation of (FOO)

User time = 4.468
System time = 0.208
Elapsed time = 4.716
Allocation = 96030696 bytes
0 Page faults

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.
The HCL Package

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  

object  

Any Common Lisp form.

Values  

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”.

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.

Description  

Certain Common Lisp objects are not represented by a single heap object; for instance, using find-object-size 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  

USER 37 > (hcl:find-object-size
(make-string 1000 :initial-element #\A))
1012

See also  

room  

total-allocation

finish-heavy-allocation  

Function

Summary  

Tells the system that allocation of many long-lived objects is over.

Package  

hcl

Signature  

finish-heavy-allocation
The function `finish-heavy-allocation` 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 `finish-heavy-allocation` is after loading something.

See also `with-heavy-allocation`

---

**flag-not-special-free-action**

*Function*

<table>
<thead>
<tr>
<th>Description</th>
<th>Unflags an object for special action on garbage collection.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package</td>
<td>hcl</td>
</tr>
<tr>
<td>Signature</td>
<td><code>flag-not-special-free-action object =&gt; nil</code></td>
</tr>
<tr>
<td>Arguments</td>
<td>object The object on which the special actions are to be removed.</td>
</tr>
<tr>
<td>Values</td>
<td>Returns nil.</td>
</tr>
</tbody>
</table>
| Example     | `CL-USER 29 : 1 > (make-instance 'capi:title-pane)
#<CAPI:TITLE-PANE "" 20F9898C>
CL-USER 30 : 1 > (flag-not-special-free-action *)
NIL` |
| See also    | `add-special-free-action` `flag-special-free-action` `remove-special-free-action` |
flag-special-free-action

Function

Summary Flags an object for special action on garbage collection.

Package hcl

Signature flag-special-free-action object => t

Arguments object The object on which the special actions are to be performed. This cannot be a symbol.

Values Returns t.

Description Note that all the current special-free-action functions are performed on the object. Use flag-not-special-free-action to unflag an object.

Example
CL-USER 29 > (make-instance 'capi:title-pane)
#<CAPI:TITLE-PANE "" 20F9898C>
CL-USER 30 > (flag-special-free-action *)
T

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

gc-generation

Function

Summary Does a Copying GC.

Package hcl

Signature gc-generation gen-num &key coalesce promote block => allocation

Arguments gen-num An integer between 0 and 7 inclusive, or t.
coalesce A generalized boolean.

promote A generalized boolean.

block An integer between 0 and 7, inclusive, or one of the keywords :blocking-gen-num and :all.

Values allocation The total allocation in generation gen-num and younger generations.

Description The function gc-generation does a Garbage Collection of a specific generation. The actual operation is different between 64-bit LispWorks and 32-bit LispWorks.

gen-num should be a valid generation number, or t. The value t is mapped to the blocking generation number in 64-bit LispWorks, and to 2 in 32-bit LispWorks. For backwards compatibility the keyword :blocking-gen-num is also accepted, with the same meaning as t.

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.

Another situation which may require gc-generation is when objects are marked for special free action (by flag-special-free-action). 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 gen-num t).

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 \textit{block} by copying them to the next higher generation.

If \textit{promote} is non-\texttt{nil}, the live objects in generation \textit{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 \textit{promote} is \texttt{nil}.

If \textit{coalesce} is non-\texttt{nil}, all non-static live objects in lower generations are promoted to generation \textit{gen-num}. That is what \texttt{clean-down} does (with \textit{gen-num} being the highest generation). It may be useful directly in some cases. The default value of \textit{coalesce} is \texttt{nil}.

\textit{block} specifies a generation number up to which to promote. An integer value specifies the generation number. If \textit{block} is \texttt{:blocking-gen-num}, then \texttt{gc-generation} promotes up to the blocking generation. If \textit{block} is \texttt{:all}, then \texttt{gc-generation} promotes nothing. The default value of \textit{block} is \texttt{:blocking-gen-num}.

\texttt{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 \texttt{gc-generation} of generation 2 or generation 3. If you know a time when the application can spend time GCing, a call to \texttt{gc-generation} with a higher value of \textit{gen-num} may be useful. It is probably never really useful to use \texttt{gc-generation} on generation 0 or 1.

To decide on which \textit{gen-num} to call \texttt{gc-generation}, check which generation gets full by making periodic calls to \texttt{room}.

\texttt{gc-generation} with \texttt{promote} or \texttt{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:

\[(\text{gc-generation 4 :coalesce t})\]

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**

\texttt{gc-generation} marks and sweeps the generation \texttt{gen-num} and all generations below, and then does some additional cleanups. \texttt{coalesce}, promote and \texttt{block} are ignored.

**Compatibility Note**

In 32-bit LispWorks, \texttt{gc-generation} simply calls \texttt{mark-and-sweep}. This has a similar effect, but two significant differences must be noted:

1. by default, \texttt{gc-generation} promotes the young generations, so repeated calls to \texttt{gc-generation} will promote everything to generation \texttt{gen-num} or generation \texttt{block} (whichever is lower). In contrast \texttt{mark-and-sweep} 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 Note**

\[(\text{gc-generation t})\]

is intended as the replacement of

\[(\text{mark-and-sweep 2})\]

**See also**

\texttt{clean-down}
\texttt{mark-and-sweep}
\texttt{marking-gc}
\texttt{set-blocking-gen-num}
gc-if-needed  
Function

Summary  Garbage collects if the previous call requires more space that is actually available in 32-bit LispWorks.

Package  hcl

Signature  gc-if-needed => nil

Arguments  None.

Values  Returns nil.

Description  This function checks to see if the amount of allocation from the previous call is more than system:*allocation-interval*, 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  
avoid-gc  
get-gc-parameters  
mark-and-sweep  
normal-gc  
set-gc-parameters  
without-interrupts  
with-heavy-allocation

get-default-generation  
Function

Summary  Returns the current default generation.

Package  hcl
Signature  get-default-generation => default-gen

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, get-default-generation 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  hcl

Signature  get-gc-parameters parameters => values

Arguments  parameters  A keyword representing a single GC parameter. Any other value means all parameters.

Values  values  If parameters specifies a single GC parameter, the value of that parameter is returned. Otherwise values is an alist containing every GC parameter, together with its current value.
Description

See \texttt{set-gc-parameters} for a full description of these parameters.

With keyword argument, of one of the parameters, the corresponding value is returned.

\textbf{Note:} \texttt{get-gc-parameters} is implemented only in 32-bit LispWorks. It is not relevant to the Memory Management API in 64-bit implementations.

Example

\begin{verbatim}
CL-USER 1 > (get-gc-parameters :minimum-overflow)
500000
CL-USER 2 > (pprint (get-gc-parameters t))

\((:\text{ENLARGE-BY-SEGMENTS} \ . \ 10)
\(:\text{MINIMUM-FOR-PROMOTE} \ . \ 1000\)
\(:\text{MAXIMUM-OVERFLOW} \ . \ 1000000\)
\(:\text{MINIMUM-OVERFLOW} \ . \ 5000000\)
\(:\text{MINIMUM-BUFFER-SIZE} \ . \ 200\)
\(:\text{NEW-GENERATION-SIZE} \ . \ 262144\)
\(:\text{PROMOTE-MAX-BUFFER} \ . \ 100000\)
\(:\text{PROMOTE-MIN-BUFFER} \ . \ 200\)
\(:\text{MAXIMUM-BUFFER-SIZE} \ . \ 131072\)
\(:\text{MINIMUM-FOR-SWEEP} \ . \ 8000\)
\(:\text{BIG-OBJECT} \ . \ 131072\))
\end{verbatim}

See also \texttt{set-gc-parameters}

\textbf{get-working-directory} \hspace{1cm} \textit{Function}

Summary

Finds the current working directory.

Package \hspace{1cm} \texttt{hcl}

Signature

\texttt{get-working-directory \Rightarrow cwd}

Arguments \hspace{1cm} None.
Values

<table>
<thead>
<tr>
<th>Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cwd</td>
<td>The current working directory, as a pathname.</td>
</tr>
</tbody>
</table>

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

```
CL-USER 1 > (get-working-directory)
#P"/u/dubya/"
```

See also

cd  
change-directory

`*handle-existing-defpackage*`

Variable

Summary

Controls LispWorks’ response when `defpackage` is used on an existing package that is different from the definition given.

Package

hcl

Initial value

`:warn :modify`

Description

The standard explicitly declines to define what `defpackage` does if the named package already exists and is in a different state to that described by the `defpackage` form. The variable `*handle-existing-defpackage*` is an extension to Common Lisp which allows you to select between alternative behaviors that are known to be useful.

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 `defpackage` but missing from the package. You can also control whether a condition is signalled.

The variable consists of a list of any of the following:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>:error</td>
<td>Signal an error.</td>
</tr>
</tbody>
</table>
:warn  Signal a warning.
:add  Add the new symbols to the externals, imports, and so on.
:modify  Modify the package to have only these externals.
:verbose  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  *handle-existing-defpackage* is an extension to Common Lisp.

See also  defpackage

*handle-old-in-package*  
Variable

Summary  Controls the handling of CLtL1-style in-package forms.

Package  hcl

Initial Value  :warn
The variable \texttt{*handle-old-in-package*} controls what happens when a CLtL1-style \texttt{in-package} form is processed. This refers to the specification in Common Lisp the Language, first Edition, which preceded ANSI Common Lisp and specified \texttt{in-package} as a function with keyword arguments.

The allowed values are as follows:

- \texttt{:quiet} Quietly use the CLtL1 definition of the \texttt{in-package} function.
- \texttt{:warn} Signal a warning and use the old definition.
- \texttt{:error} Signal a continuable error.

See also \texttt{*handle-old-in-package-used-as-make-package*}

\texttt{*handle-old-in-package-used-as-make-package*} Variable

Summary Controls the handling of CLtL1-style \texttt{in-package} forms.

Package \texttt{hcl}

Initial Value \texttt{:quiet}

Description The variable \texttt{*handle-old-in-package-used-as-make-package*} controls what happens when a CLtL1-style \texttt{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 \texttt{in-package} as a function with keyword arguments.

The allowed values are as follows:

- \texttt{:quiet} Handle according to the value of \texttt{*handle-old-in-package*}.
- \texttt{:warn} Signal a warning and create the package.
- \texttt{:error} 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

**Function**

**Summary**

Garbage collects a specified generation in 32-bit LispWorks.

**Package**

hcl

**Signature**

`mark-and-sweep gen-number => bytes`

**Arguments**

`gen-number`

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**

`bytes`

The number of bytes allocated in that generation.

**Description**

`mark-and-sweep` 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.

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 `mark-and-sweep` takes, but also the more space recovered.

**Note:** `mark-and-sweep` 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 `gc-generation` or `marking-gc`. 
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
Function

normal-gc

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.

Notes
*max-trace-indent* is an extension to Common Lisp.

See also
trace
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-floating**  

**Function**

**Summary**
Parses a float from a string and returns it as float.

**Package**
hcl
Signature  \texttt{parse-float string &key start end default-format => float}

Arguments  \begin{itemize}
  \item \textit{string}  A string
  \item \textit{start, end}  Bounding index designators for \textit{string}
  \item \textit{default-format}  One of the atomic type specifiers \texttt{short-float, single-float, double-float,} or \texttt{long-float}.
\end{itemize}

Values  \texttt{float}  A float

Description  The function \texttt{parse-float} parses a float from the substring of \texttt{string} delimited by \textit{start} and \textit{end} and returns it as \texttt{float}.

If the substring represents an integer or the exponent marker is \texttt{E} or is omitted, then \texttt{float} will be of type \texttt{default-format}, which defaults to the value of \texttt{*read-default-float-format*}. Otherwise, its type will match the exponent marker as specified by 2.3.2.2 "Syntax of a Float" in the Common Lisp standard.

If the substring does not represent an integer or a float, then an error of type \texttt{parse-error} is signalled.

Examples  \begin{verbatim}
(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
\end{verbatim}

\textbf{print-profile-list}  \textit{Function}

Summary  Prints a report of symbols that have been profiled.

Package  \texttt{hcl}
Signature: \texttt{print-profile-list &key sort limit cutoff collapse => nil}

Arguments:
- \texttt{sort}: \texttt{:call, :profile or :top}
- \texttt{limit}: An integer.
- \texttt{collapse}: A generalized boolean.
- \texttt{cutoff}: A real number.

Values: \texttt{print-profile-list} returns \texttt{nil}.

Description:
The function \texttt{print-profile-list} prints a report of symbols, after profiling using \texttt{profile}, or \texttt{start-profiling} followed by \texttt{stop-profiling}.

If the profiler was set up with \texttt{style :tree}, then a tree of calls is printed first, according to \texttt{limit, cutoff} and \texttt{collapse}. 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 \texttt{sort}.

If the profiler was set up with \texttt{style :list}, then only the columnar report is printed.

\texttt{sort} can take these values:

- \texttt{:call}: Sort by the number of times the function was called.
- \texttt{:profile}: Sort by the number of times the function was found on the stack.
- \texttt{:top}: Sort by the number of times the function was found at the top of the stack.

If \texttt{sort} is not passed then the results are printed as after the profiling run. The default is the value of the variable \texttt{*default-profiler-sort*}.

\texttt{limit} is the maximum number of lines printed in the columnar report as described for \texttt{*default-profiler-limit*}. The
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:

```lisp
CL-USER 1 > (set-up-profiler
  :symbols
    '((cadr car eql fixnump + 1+ caadr cddr))

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

<table>
<thead>
<tr>
<th>Symbol</th>
<th>called</th>
<th>profile</th>
</tr>
</thead>
<tbody>
<tr>
<td>CADR</td>
<td>5000012</td>
<td>13</td>
</tr>
<tr>
<td>CDDR</td>
<td>3000000</td>
<td>3</td>
</tr>
<tr>
<td>EQL</td>
<td>2000202</td>
<td>4</td>
</tr>
<tr>
<td>FIXNUMP</td>
<td>2000003</td>
<td>2</td>
</tr>
<tr>
<td>CAR</td>
<td>1000000</td>
<td>1</td>
</tr>
<tr>
<td>+</td>
<td>1000000</td>
<td>3</td>
</tr>
<tr>
<td>CAADR</td>
<td>1000000</td>
<td>2</td>
</tr>
<tr>
<td>1+</td>
<td>1000000</td>
<td>2</td>
</tr>
</tbody>
</table>

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

Summary
Runs the specified forms, and prints a performance profile.

Package
hcl

Signature
profile &body forms => final

Arguments
forms
The forms making up the program being profiled.

Values
final
The result of evaluating the final form.

Description
This macro starts up the LispWorks program profiler. This tool is useful for determining the time critical elements of a program.

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 *profile-symbol-list*. 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.

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.
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
304409320171337804361260816606476884437764156896051200
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 hcl

Description *profiler-threshold* is used with repeated profiling runs, to control which symbols are profiled. It is set by set-profiler-threshold.
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

The function `profiler-tree-from-function` prints a tree with root `function-name` whose children are the callees of `function-name` and their callees.

`profiler-tree-from-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`.

If `max-depth` is a number it limits the depth of tree that is printed to that value. The default value of `max-depth` is `nil`, meaning no limit on the depth that is printed.

See also

`profile`
`start-profiling`
`stop-profiling`

---

**profiler-tree-to-function**

*Function*

**Summary**

Prints a reversed call tree of profiled code below a given function.

**Package**

`hcl`

**Signature**

`profiler-tree-to-function function-name &optional max-depth`

**Arguments**

`function-name` A symbol naming a function.

`max-depth` A number or `nil`.

**Description**

The function `profiler-tree-to-function` prints a tree with root `function-name` whose children are the callers of `function-`
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`

---

**references-who**

<table>
<thead>
<tr>
<th>Summary</th>
<th>Lists special variables referenced by a definition.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package</td>
<td>hcl</td>
</tr>
<tr>
<td>Signature</td>
<td><code>references-who function =&gt; result</code></td>
</tr>
<tr>
<td>Arguments</td>
<td><code>function</code> A symbol or a function dspec.</td>
</tr>
<tr>
<td>Values</td>
<td><code>result</code> A list.</td>
</tr>
<tr>
<td>Description</td>
<td>The function <code>references-who</code> returns a list of the special variables referenced by the definition named by <code>function</code>.</td>
</tr>
</tbody>
</table>
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
remove-symbol-profiler  symbol  =>  nil

Arguments  symbol  A symbol to be removed from the *profile-symbol-list*.

Values  Returns nil.

Description  remove-symbol-profiler removes a symbol from *profile-symbol-list*, the list of profiled symbols.

See also  add-symbol-profiler
*profile-symbol-list*

reset-profiler  Function

Summary  Resets the profiler so that symbols below a given threshold are no longer profiled.

Package  hcl

Signature  reset-profiler &key according-to  =>  nil

Arguments  according-to  One of two values — :profile or :top. This refers to which column of the profiling results reset-profiler uses to determine which symbols to delete from *profile-symbol-list*. The default is :profile.

Values  reset-profiler returns nil.

Description  This function updates the list of symbols being profiled according to the results of the previous profiling run. reset-profiler runs down the list of symbols being profiled and removes any symbols whose appearance in the previous profiling run falls below the value *profiler-threshold*. 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 \texttt{t}. When using \texttt{save-universal-from-script}, any required application bundle should be created before calling that function (see the \texttt{save-macos-application.lisp} example below). When using the Application Builder, any required application bundle should be created in the build script only when \texttt{save-argument-real-p} returns \texttt{t}.

On architectures that do not have universal binaries, this function always returns \texttt{t}.


dirname = "examples/configuration"
save-macos-application.lisp
delete dir

Example examples/configuration/save-macos-application.lisp
See also save-universal-from-script
building-universal-intermediate-p
deliver
save-image

\textbf{save-image} \hspace{1cm} \textit{Function}

Summary Saves the image to a new file.

Package \texttt{hcl}

Signature \texttt{save-image filename &key dll-exports dll-added-files automatic-init gc type normal-gc restart-function multiprocessing console environment remarks clean-down image-type => nil}

The \texttt{console} argument is available only in LispWorks for Windows and LispWorks for Macintosh.
Arguments

- **filename**: 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.

- **dll-exports**: A list of strings, or the keyword :default.

- **dll-added-files**: A list of strings.

- **automatic-init**: A generalized boolean.

- **gc**: If non-nil, there is a garbage collection before the image is saved. The default value is t.

- **type**: Determines if some global variables are cleared before the image is saved. You can generally use the default value, which is :user.

- **normal-gc**: If this is t the function normal-gc is called before the image is saved. The default is t.

- **restart-function**: A function to be called on restart.

- **multiprocessing**: Controls whether multiprocessing is enabled on restart.

- **console**: On Windows console 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 console controls when, if ever, to make a console window.

  Possible values are discussed below.

- **environment**: environment controls whether the LispWorks environment is started on restart. Possible values are discussed below.

- **remarks**: remarks adds a comment to the save history. The value should be a string.

- **clean-down**: When t, calls (clean-down t).
The function `save-image` saves the LispWorks image to a new executable or dynamic library containing any modifications you have made to the supplied image.

For information about the sort of changes you might want to save in a new image, see "Customization of LispWorks" in the *LispWorks User Guide*.

Do not use `save-image` 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.

You cannot use `save-image` on Windows, Linux and Mac OS X when multiprocessing is running. It signals an error in this case.

On Cocoa you can combine a call to `save-image` with the creation of an application bundle containing your new LispWorks image, as in the example shown below.

`dll-exports` 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 `dll-exports` should be list (potentially `nil`) of strings. In this case a dynamic library is saved, and each string in `dll-exports` names a function which becomes an export of the dynamic library and should be defined as a Lisp function using `fli:define-foreign-callable`. Each exported name can be found by `GetProcAddress` (on Windows) or `dlsym` (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.
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.

custom is implemented only in Lispworks for Windows and LispWorks for Macintosh. The possible values for custom are as follows:

:default Unchanged since previous save.

t On the Macintosh, the value t has the same effect as the value :always.

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*.

:init At startup, if input and output are not redirected

:always At startup, even if input and output are redirected.

The LispWorks for Windows and LispWorks for Macintosh images shipped have custom set to :input.

The possible values for environment are as follows:
The HCL Package

:default

Unchanged since previous save.

nil

Start with just the TTY listener.

t

Start the environment automatically, no TTY listener.

:with-tty-listener

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 hcl

Signature `save-universal-from-script` target-image script-name &key output-stream => target-image

Arguments
- target-image: A pathname designator.
- script-name: A pathname designator.
- output-stream: A stream or nil.

Values
- target-image: A pathname designator.

Description
The function `save-universal-from-script` provides a convenient way to create a universal binary on an Intel Macintosh, using a script designed for saving a mono-architecture image.

The `script-name` 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 `deliver` or `save-image` as appropriate.

The function `save-universal-from-script` runs the current LispWorks image in two subprocesses, once for the PowerPC architecture (under Rosetta) and once for the native Intel architecture, passing `-build script-name` on the command line. The script is evaluated as normal, except that the file-name that is passed to any call to `save-image` or `deliver` 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 `target-name` in the same way as `create-universal-binary`.

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 `deliver` or `save-image` work in the script.
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

```lisp
(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:

```lisp
(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**

`hcl`

**Signature**

```
set-array-weak array weakp => weakp
```

**Arguments**

- `array` A non-displaced array, with
  `array-element-type` `t`.
- `weakp` If `weakp` is non-`nil`, the array is made weak. If `weakp` is `nil`, the array is made non-weak.
Values

Returns `weakp`.

Description

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 `array` with `nil`.

Pointers are replaced by `nil` 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.

The weakness state of an array can be changed many times.

In all implementations, `array` must not be a displaced array, and the `array-element-type` of `array` must be `t`.

In 64-bit LispWorks, an additional requirement is that `array` must be an adjustable array.

`set-array-weak` can be called at any moment.

Note: An array can be made weak at creation time using the `:weak` argument to `make-array`.

See also

`copy-to-weak-simple-vector`

`set-hash-table-weak`

`make-array`

`mark-and-sweep`

---

**set-default-generation**

*Function*

**Summary**

Set the current generation for storage allocation in 32-bit LispWorks.
Package: hcl

Signature: set-default-generation num => num

Arguments: num

The number of the generation from which to do future allocation.

Values: Returns num.

Description
Set the current generation for storage allocation. By default the system allocates memory from the youngest generation (generation 0).

Note: set-default-generation is useful only in 32-bit LispWorks. In 64-bit implementations it does nothing and returns 0.

Examples
(set-default-generation 1)
    ;; allocate from an older generation
(set-default-generation 0)
    ;; return to normal

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

Package: hcl

Summary
Sets the weakness state of a hash-table.

Function
Signature  \texttt{set-hash-table-weak hash-table weak => weakness-state}

Arguments  \texttt{hash-table}  \\
\texttt{weak}  \\
A hash-table.

\texttt{weak}  \\
Sets the weakness state of \texttt{hash-table}. Value may be:

\texttt{:value or t}  — An entry is kept if there is a pointer to the value from another object.

\texttt{:key}  — An entry is kept if there is a pointer to the key from another object.

\texttt{:both}  — An entry is kept if there are pointers to both the key and the value.

\texttt{:one or :either}  — An entry is kept if there is a pointer to either the key or the value.

\texttt{nil}  — Make the hash-table non-weak. All entries are kept.

Values  Returns \texttt{weak}, unless \texttt{t} was passed, when \texttt{:value} is returned.

Description  By default, hash-tables are not weak, which means that they keep alive all the keys and the values in the table.

A weak hash-table allows entries to be removed if there are no other pointers to them. The \texttt{weakness-state} tells the system which entries may be removed like this.

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.
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:** *set-gc-parameters* is implemented only in 32-bit LispWorks. It is not relevant to the Memory Management API in 64-bit implementations.

**See also**

*get-gc-parameters*

---

**set-minimum-free-space**

**Function**

**Summary**

Sets the minimum free space for a segment of the specified generation in 32-bit LispWorks.

**Package**

`hcl`

**Signature**

```
set-minimum-free-space gen-num size &optional segment => generation-size
```

**Arguments**

- **gen-num**
  The generation to be affected.
- **size**
  The size (in bytes) to set the segment to.
- **segment**
  An integer specifying the segment to be affected. The default value is 0, meaning the first segment of the generation.
### Values

**Values**

| generation-size | A list showing information for the generation just specified in the call. |

### Description

**Description**

Sets the minimum free space for a segment of the specified generation.

By default, affects the first segment — pass `segment` to affect a different segment of the generation.

The minimum free space is shown by `room`.

**Note:**

`set-minimum-free-space` is implemented only in 32-bit LispWorks. It is not relevant to the Memory Management API in 64-bit implementations.

### See also

- `clean-generation-0`
- `collect-generation-2`
- `collect-highest-generation`
- `expand-generation-1`
- `room`
- `set-promotion-count`

---

### set-process-profiling

#### Function

**Summary**

Controls the set of processes that are profiled.

**Package**

`hcl`

**Signature**

`set-process-profiling flag processes`

**Arguments**

- **flag**: `:add`, `:remove` or `:set`.
- **processes**: One of `:current`, `:all`, a `mp:process` object, or a list of `mp:process` objects which may also contain `:current`.
The function `set-process-profiling` modifies the set of processes for which profiling information is (or will be) collected.

If `set-process-profiling` is called while profiling (that is after a call to `start-profiling` and before the next call to `stop-profiling` with `print non-nil`) the system immediately starts collecting profile information for the new set of processes.

When `start-profiling` is called without passing `processes`, it sets the processes to profile according to the last call to `set-process-profiling`.

`flag` determines how the set of processes to profile is modified:

: `add` The given processes are added to the set.
: `remove` The given processes are removed from the set.
: `set` The given processes are used as the set.

`processes` controls which processes are added to the set, removed from the set or are contained in the set, as follows:

: `current` Means the current process. When `start-profiling` is called it interprets `current` to mean the current process at the time it is called. If `set-process-profiling` is called while profiling, `current` is interpreted as the current process when `set-process-profiling` is called.

: `all` Means all processes, including those which are created after profiling started.

A `mp:process` object

Means that process.
A list means the processes in that list. The list can contain the symbol :current, which is interpreted as described above.

(set-process-profiling) can be called whether or not the profiler is collecting information. See start-profiling and stop-profiling.

Examples

Add process1 to the set:

(set-process-profiling :add process1)

Turn off profiling for the current process:

(set-process-profiling :remove :current)

Turn off all profiling:

(set-process-profiling :remove :all)

Set all processes for later profiling:

(set-process-profiling :set :all)

See also

profile
start-profiling
stop-profiling

(set-profiler-threshold) Function

Summary
Sets the percentage threshold for symbols to be profiled in a subsequent run.

Package
hcl

Signature
set-profiler-threshold value => value

Arguments
value must be a fixnum between 0 and 100.

Values
set-profiler-threshold returns value.
This function sets the value of *profiler-threshold* 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 *profiler-threshold* is set to 40 then by running reset-profiler with argument :top 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

(set-profiler-threshold 40)

See also
reset-profiler
profile
*profiler-threshold*

---

**set-promotion-count**

Function

Controls when objects can be promoted to the next generation in 32-bit LispWorks.

**Summary**

**Package**
hcl

**Signature**

set-promotion-count gen-num count &optional segment => count

**Arguments**

- **gen-num**
  The generation number affected.

- **count**
  The number of garbage collections survived by objects in that generation, before promotion. If count is nil, the function returns the current promotion count setting.

- **segment**
  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 count.

Description

Controls how many garbage collections an object in a segment must survive before promotion to the next generation.

**Note:** The `set-promotion-count` 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 `set-delay-promotion`.

See also

- `block-promotion`
- `clean-generation-0`
- `collect-generation-2`
- `collect-highest-generation`
- `expand-generation-1`

---

**set-up-profiler**

**Function**

**Summary**

Declares the parameter values of the profiling function.

**Package**

`hcl`

**Signature**

```
(set-up-profiler &key symbols packages kind interval limit cutoff collapse style gc call-counter show-unknown-frames)
```

**Arguments**

- `symbols` A symbol or a list of symbols.
- `packages` A valid package name, or a list of package names, or `:all`.
- `kind` `:profile`, `:virtual` or `:real`.
- `interval` An integer greater than or equal to 10000.
- `limit` An integer or `nil`.
- `cutoff` An integer or `nil`.
- `collapse` A generalized boolean.
- `style` `:tree`, `:list` or `nil`. 
Values

The time interval is returned.

Description

`set-up-profiler` is used to declare the values of the parameters of the profiling function. Three values are required, as follows.

`symbols`, if non-nil, specifies which symbols are to be monitored by the profiler. Each symbol in `symbols` is checked to see if it is suitable for profiling and if so it is added to the list `*profile-symbol-list*`.

If `symbols` is not passed then `packages` specifies which symbols are to be monitored. If `packages` is :all, then all packages are monitored. All the symbols in the packages are checked as above. If a `symbols` argument is present then `packages` is ignored.

`kind` specifies the way that the time between samples is measured on Unix-like platforms:

`:profile` Process time only.

`:virtual` Process time and system time for the process.

`:real` Real time.

The default value of `kind` is :profile.

Note: `kind` is ignored on Microsoft Windows platforms.

`interval` specifies the interval in microseconds between profile samples. The minimum value of `interval` is 10000, that is 10 ms. The default value of `interval` is 10000.
limit, when non-nil, sets \texttt{*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 \texttt{*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 \texttt{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 \texttt{*default-profiler-collapse*}. The default value of \texttt{collapse} is \texttt{nil}.

style controls the format of output. If \texttt{style} is not passed or passed as \texttt{nil}, the format does not change. If \texttt{style} is passed, it can take these values:

\texttt{:list} The profiler will show the functions seen on the stack.

\texttt{:tree} The profiler will generate a tree of calls seen in the profiler, as well as the output shown by \texttt{:list}.

The default value of \texttt{style} is \texttt{: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 \texttt{gc} is \texttt{nil}.

call-counter whether to add extra code to count calls. The counting is done dynamically. If \texttt{call-counter} is \texttt{nil}, call counters are not added, and the call counter of all functions is displayed as 0. The default value of \texttt{call-counter} is \texttt{t}.

\textbf{Note:} Call counting can affect performance significantly on some platforms. To get accurate timing (in scales of a few percentage points), pass \texttt{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.
### sets-who

**Values**

| result | A list. |

**Description**

The function `sets-who` returns a list of the special variables set by the definition named by `function`.

**Note:** The cross-referencing information used by `sets-who` is generated when code is compiled with source-level debugging switched on.

**See also**

- `who-sets`
- `toggle-source-debugging`

### source-debugging-on-p

**Function**

**Summary**

Tests if source level debugging is on for compiled code.

**Package**

`hcl`

**Signature**

`source-debugging-on-p => bool`

**Arguments**

None.

**Values**

| bool | If `t`, source level debugging is on. |

**Description**

Returns `t` if source level debugging is on for compiled code; otherwise returns `nil`.

**See also**

- `toggle-source-debugging`

### start-profiling

**Function**

**Summary**

Starts collecting profiling information.

**Package**

`hcl`
Signature  
\texttt{start-profiling \&key initialize \ processes} 

Arguments  
\textit{initialize} \hspace{1cm} A boolean.  
\textit{processes} \hspace{1cm} One of \texttt{:current, :all, \texttt{mp:process}} or a list of \texttt{mp:process} objects. 

Description  
The function \texttt{start-profiling} starts collecting profiling information. 

If \textit{initialize} is non-\texttt{nil} any profiling information collected so far is discarded. The default value of \textit{initialize} is \texttt{t}. 

If \textit{processes} is supplied, the set of processes that will be profiled is set as if by calling: 

\begin{verbatim}
(set-process-profiling :set :processes processes)
\end{verbatim} 

Otherwise, the set of processes remains unchanged, so is controlled by any previous calls to \texttt{set-process-profiling}. Only processes that are active are profiled. 

\texttt{start-profiling} can be repeatedly called without intervening calls to \texttt{stop-profiling}, for example to change the profiled processes. 

\texttt{start-profiling} cannot be used while \texttt{profile} is used or while the Profiler tool is profiling (on any thread). Between the call to \texttt{start-profiling} and the next call to \texttt{stop-profiling} with \texttt{print \ t} (or omitted), \texttt{profile} and the Profiler tool cannot be used. 

Various parameters which are set by \texttt{set-up-profiler} control the behavior of the profiler. See the documentation for \texttt{set-up-profiler}. 

Examples  
The following sequence of calls to \texttt{start-profiling} and \texttt{stop-profiling} can be used to profile only interesting work and print the results: 

Start profiling the current process:
(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

(stop-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**
hcl

**Signature**
`switch-static-allocation flag => previous-flag`

**Arguments**
flag
If flag is non-nil, subsequent objects are allocated in the static area; if flag has any other value, objects are allocated conventionally.

**Values**
`switch-static-allocation` returns the previous setting of flag.

**Description**
Objects in the static area are garbage-collected, but not moved.
You should avoid using this function.

See also `enlarge-static`
`in-static-area`

**symbol-alloc-gen-num**

*Variable*

**Summary**
Specifies the generation in which interned symbols and their symbol names are allocated.
Package: hcl
Initial Value: 2 in 32-bit LispWorks, 3 in 64-bit LispWorks
See also: allocation-in-gen-num, get-default-generation, set-default-generation

toggle-source-debugging

Summary: Changes compiler settings affecting production of source level debugging information.

Package: hcl
Signature: toggle-source-debugging &optional on => bool

Arguments: on
Flag (t or nil) to control the resulting setting of the variables. The default is t.

Values: bool
The current state of source level debugging: t if source level debugging is on.

Description: toggle-source-debugging sets certain compiler parameters, and also turns leaf case optimizations on (when called with nil) or off (when called with t). For all these parameters, the value nil reduces compilation speed.

toggle-source-debugging is called in the configuration file a-dot-lispworks.lisp, and the initial state of LispWorks such that source level debugging is on.

The parameters relate to information required for source level debugging, cross-referencing and finding all changed definitions.

The parameters (all in the compiler package) are:
*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
hcl

### Signature
total-allocation

### 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
find-object-size
room

### *traced-arglist*  

**Variable**

### Summary
The list of arguments given to the function being traced.

### Package
hcl

### Initial Value
nil

### Description
Upon entering a function that is being traced, *traced-arglist* is bound to the list of arguments given to the function. *traced-arglist* is then printed after the function name in the output from tracing. It is accessible in the :before and :after forms to trace. However care should be used when manipulating this variable, since it is the value of *traced-arglist* itself that is used when calling the traced function. Thus if this value is altered by the :before forms then the function receives the altered argument list.

### Example
USER 14 > (trace (+ :before
  ((setq *traced-arglist*
    (mapcar #'1+
      *traced-arglist*)))))

---

285
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))
*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**

```lisp
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
```
*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**  

**Variable**

**Summary**  
Controls how circular structure are printed in trace output.

**Package**  
hcl

**Initial Value**  
nil

**Description**  
*trace-print-circle* 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 *print-circle*. *print-circle* is bound to the value of *trace-print-circle* while printing tracing information.

**Example**  
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

**Notes**  
*trace-print-circle* is an extension to Common Lisp.

**See also**  
trace

---

**trace-print-length**  

**Variable**

**Summary**  
The number of components of an object that are printed in trace output.

**Package**  
hcl
**The HCL Package**

**Initial Value**
100

**Description**
*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.

*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.

**Example**

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)

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

**See also**
trace

**trace-print-level**

*Variable*

**Summary**
The depth to which nested objects are printed in trace output.

**Package**
hcl

**Initial value**
5
Description  

*trace-print-level* 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 subcomponents are at level 2, and so on.

*print-level* is bound to the value of *trace-print-level* while printing tracing information. If *trace-print-level* is nil then objects are printed without regard to depth.

Examples

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)

Notes  

*trace-print-level* is an extension to Common Lisp.

See also  

See also  

*trace-print-pretty*  

Variable

Summary  

Controls the amount of whitespace in trace output.

Package  

hcl

Initial Value  

nil

Description  

*trace-print-pretty* controls the amount of whitespace printed during output from tracing. If it is not nil 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*  

Summary  Controls how arguments and values are printed in trace output.

Package  hcl

Initial Value  :only
Description *trace-verbose* controls the way arguments and values are printed in trace output.

If the value is not nil then trace attempts to decode the arguments and values, and prints them.

When the value is :only, trace does not print the lists of arguments and values after the function name.

Notes *trace-verbose* is an extension to Common Lisp.

See also trace

**try-compact-in-generation**

*Function*

Summary Compacts the most fragmented segment(s) in a generation in 32-bit LispWorks.

Package hcl

Signature try-compact-in-generation generation-number time-threshold &optional fraction-threshold => result

Arguments generation-number

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.

time-threshold A real number.

fraction-threshold A real number between 0 and 1, defining the minimum fragmentation to actually compact. The default is 0.25.

Values result A boolean.
try-compact-in-generation finds the most fragmented segment in the generation specified. If time-threshold is positive, it compacts this segment, and repeats this operation until time-threshold seconds have elapsed. At this point try-compact-in-generation returns, with value t if at least one segment was compacted and value nil otherwise. Because the operation cannot be stopped in the middle, the actual time taken will always be larger than time-threshold.

If fraction-threshold is 1, try-compact-in-generation does nothing. If fraction-threshold is 0, try-compact-in-generation will compact all uncompacted segments (unless it runs out of time). With the default (0.25) try-compact-in-generation compacts only moderately fragmented segments.

If time-threshold is negative, then try-compact-in-generation does not actually compact any segments. result is a boolean indicating whether try-compact-in-generation would actually try to compact 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-compact-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-move-in-generation
**try-move-in-generation**  

**Function**

**Summary**
Moves objects out of the most fragmented segment(s) in a generation, leaving them empty in 32-bit LispWorks.

**Package**
`hcl`

**Signature**
```
try-move-in-generation generation-number time-threshold
&optional fraction-threshold => result
```

**Arguments**
- `generation-number`
  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.
- `time-threshold`
  A real number.
- `fraction-threshold`
  A real number between 0 and 1, defining the minimum fragmentation to actually move. The default is 0.25.

**Values**
- `result`
  A boolean.

**Description**
`try-move-in-generation` finds the most fragmented segment in the generation specified. If `time-threshold` is positive, it moves objects out of this segment, leaving it empty, and repeats this operation until `time-threshold` seconds have elapsed. At this point `try-move-in-generation` returns, with value `t` if at least one segment was moved and value `nil` otherwise. Because the operation cannot be stopped in the middle, the actual time taken will always be larger than `time-threshold`.

If `fraction-threshold` is 1, `try-move-in-generation` does nothing. If `fraction-threshold` is 0, `try-move-in-generation` will move all uncompacted segments (unless it runs out of time).
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

**Function**

**Summary**
Returns the definitions which bind a special variable.

**Package**
hcl

**Signature**

\[\text{who-binds} \ symbol \Rightarrow result\]

**Arguments**

- **symbol**
  A special variable.

**Values**

- **result**
  A list.

**Description**
The function **who-binds** returns a list of dspecs naming the definitions which bind the special variable **symbol**.
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**

```latex
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 | hcl
---|---
Signature | who-references symbol => result
Arguments | symbol | A special variable.
Values | result | A list.
Description | The function **who-references** returns a list of dspecs naming the definitions which reference the special variable **symbol**.

**Note:** The cross-referencing information used by **who-references** is generated when code is compiled with source-level debugging switched on.

See also | references-who
toggle-source-debugging

---

**who-sets**

*Function*

Summary | Returns the definitions which set a special variable.
---|---
Package | hcl
Signature | who-sets symbol => result
Arguments | symbol | A special variable.
Values | result | A list.
Description | The function **who-sets** returns a list of dspecs naming the definitions which set the value of the special variable **symbol**.

**Note:** The cross-referencing information used by **who-sets** is generated when code is compiled with source-level debugging switched on.
with-heavy-allocation

**Macro**

**Summary**
Slows up garbage collection during the execution of code that allocates a lot of space.

**Package**
hcl

**Signature**
```
with-heavy-allocation &rest body => result
```

**Arguments**

- `body`
The forms for which you want the garbage collector to behave differently from normal.

**Values**

- `result`
The result of executing `body`.

**Description**
The macro `with-heavy-allocation` 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 `with-heavy-allocation` is implemented only in 32-bit LispWorks. In version 5.1 and later it is implemented in 64-bit LispWorks as well.

**See also**
sets-who
toggle-source-debugging

avoid-gc
gc-if-needed
get-gc-parameters
mark-and-sweep
normal-gc
set-gc-parameters
finish-heavy-allocation
without-interrupts
with-output-to-fasl-file

Function

Summary Sends output to a fasl file on disk.

Package hcl

Signature

\[
\text{with-output-to-fasl-file (stream pathname &rest options)}
\]

\[
&\text{body body} \Rightarrow \text{nil}
\]

Arguments

\[
\begin{align*}
\text{stream} & \quad \text{Stream to be bound to the fasl file to be created.} \\
\text{pathname} & \quad \text{Name of the fasl file to be created.} \\
\text{body} & \quad \text{Forms, some of which may be dumped.}
\end{align*}
\]

Values Returns \text{nil}.

Description

\text{with-output-to-fasl-file} is used in conjunction with \text{dump-form}. The \text{body} forms are executed, and during the execution, \text{dump-form} may be called to dump selected forms. Dumped forms are evaluated if the file \text{pathname} is later loaded by \text{load-data-file}.

Supply an appropriate fasl extension in \text{pathname}. A simple way to achieve this is by calling \text{compile-file-pathname}. A complete list of fasl extensions for supported platforms may be found in \text{compile-file}.

If the file \text{pathname} already exists, it is superseded.

A fasl file created using \text{with-output-to-fasl-file} must be loaded only by \text{load-data-file}, and not by \text{load}. 


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
8 The HCL 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).

### break-on-unresolved-functions

#### Function

**Package**

`link-load`

**Signature**

`break-on-unresolved-functions &optional stream`

**Arguments**

`stream`

An output stream for message reporting. If set to `nil`, then no output will be produced. By default this is `t`.

**Description**

The `break-on-unresolved-functions` 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
This chapter applies only to LispWorks for UNIX

`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

### lisp-name-to-foreign-name

**Function**

**Package**
- link-load

**Signature**
- lisp-name-to-foreign-name name &key language

**Arguments**
- 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**

```lisp
(lisp-name-to-foreign-name 'lisp-name-with-hyphens)
"_lisp_name_with_hyphens"
```

**See also**
- get-foreign-symbol
### read-foreign-modules

**Function**

**Package**  
link-load

**Signature**  
read-foreign-modules &rest module-names => t

**Arguments**  
module-names  
A sequence of strings or pathnames.

**Values**  
t

**Description**  
The `read-foreign-modules` 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, `read-foreign-modules` scans the default libraries looking for definitions of referenced but undefined symbols.

The `module-names` 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 `*default-library-name-search-paths*`, the environment variable `LD_LIBRARY_PATH` and the variable `*default-library-names*`. Object files of various formats and library files can be handled by `read-foreign-modules`.

**Example**  
```
(read-foreign-modules "/usr/users/clc/projects/head.o" 
                    "~clc/projects/libs.a" 
                    "-lW")
```

**Notes**  
The `read-foreign-modules` function actually adds the module-names to the list of modules in the variable `*default-libraries*` and then tries to resolve any undefined symbols using this list. The function `get-foreign-symbol` may be
called to explicitly force a symbol onto the undefined list or
the act of defining a foreign function (\texttt{fli:\texttt{define-foreign-}
function}) will do it implicitly.

\texttt{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  \texttt{*unresolved-
messages*} is set to \texttt{t} (the default is \texttt{nil}). By default messages
are printed out about which object modules are being loaded.
This may be switched off by setting  \texttt{*coff-loading-
verbose*} to \texttt{nil}.

\textbf{See also} \texttt{get-foreign-symbol}
This chapter describes symbols available in the LISPWORKS package. This package is used by default. Its symbols are visible in the CL-USER package.

**8-bit-string**

Type

**Summary**
The 8 bit string type.

**Package**
lispworks

**Signature**
8-bit-string length

**Arguments**
length 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.
10 The LISPWORKS Package

16-bit-string

Summary The 16 bit string type.

Package lispworks

Signature 16-bit-string length

Arguments length 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

Summary Appends lists to the end of a given list.

Package lispworks

Signature appendf place &rest lists => result

Arguments place A place.
lists A set of lists.

Values result An object.

Description The modify macro appendf appends the lists given by lists to the end of the list in place. See append for more details.

See also removef

base-character

Summary The base character type.
Package lispworks
Signature base-character
Description The type of base characters.
base-character is a synonym for the Common Lisp type base-char.
See also base-char-code-limit

base-character-p Function
Summary Tests if an object is a base character
Package lispworks
Signature base-character-p object => bool
Arguments object The object to be tested.
Values bool t if object is a base character; nil otherwise.
Description This is the predicate for base characters.
See also base-character

base-char-p Function
Summary Tests if an object is a base character
Package lispworks
Signature base-char-p object => bool
Arguments object The object to be tested.
The LISPWORKS Package

Values  

bool  

t if object is a base character; nil 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 char) among base characters.

base-string-p  

Function

Summary  

Tests if an object is a base string.

Package  

lispworks

Signature  

base-string-p object => bool

Arguments  

object  

The object to be tested.

Values  

bool  

 t if object is a base string; nil otherwise.

Description  

This is the predicate for base strings.

See also  

base-string
*browser-location*  

**Variable**

**Signature**  
*browser-location*

**Package**  
lispworks

**Initial Value**  
nil

**Description**  
Controls how the online documentation interface and the function `open-url` find a web browser executable (either Netscape, Firefox, Mozilla or Opera) to use. The value should be `nil` or a string.

If the value is `nil`, LispWorks attempts to find the browser using the value of the environment variable `PATH`.

If the value is a string, it specifies the directory in which the browser is installed. Typical values are `"/usr/bin/"` and `"/usr/local/bin/"`.

**Note:** do not omit the trailing slash.

**Note:** *browser-location* is used only in the Motif-based IDE.

**See also**  
open-url

**call-next-advice**  

**Function**

**Summary**  
Calls the next piece of advice associated with a function.

**Package**  
lispworks

**Signature**  
call-next-advice `args`

**Arguments**  
`args` 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

call-next-advice 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

call-next-advice 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 defadvice 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 call-next-advice. A call to this from within the body of the around advice invokes the next piece of around advice with the arguments given to call-next-advice. The last piece of around advice in the ordering invokes the sequence of before advice, the original definition, and after advice if it calls call-next-advice. Around advice may contain any number of calls to call-next-advice, including no calls.

Notes

call-next-advice is an extension to Common Lisp. See the LispWorks User Guide for a broader discussion of advice.

See also

defadvice
### Function: compile-system

**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 system-name &key force simulate load args
```

**Arguments**
- `system-name` (required): A symbol representing the name of the system. The system must have been defined already using the `defsystem` macro.
- `force` (optional): If `t` then all the files in the system are compiled regardless. (This argument was formerly called `force-p`. The old name is currently still accepted for compatibility.)
- `simulate` (optional): 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 `simulate`:
  - `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.
  - `:simulate` may be abbreviated as `:sim`.

---

315
If `t` then `load-system` is called after `compile-system` has finished. If `:no` then no files are loaded at all. The default is `nil`.

Args

Arguments to be passed directly to the compiler.

target-directory

This must be a string representing a valid directory. It defaults to the `:default-pathname` option to `defsystem`. This is the directory where the object files created are put. If the `target-directory` is given then dependency information expressed in the system rules is ignored. `:target-directory` may be abbreviated as `:t-dir`.

Values

`compile-system` returns `nil`.

Examples

```lisp
(compile-system 'blackboard :simulate :ask)

(compile-system 'tms :load t)

(compile-system 'packages :load :no
  :target-directory "/usr/users/386i/"
```

Notes

If the `:load` keyword is set to `t` then by default `load-system` is called after `compile-system`. This behavior can be changed to loading any file immediately after it is compiled by setting the variable `defsystem::*load-when-compile*` to non-nil.

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 `foo.o` where `n` is a platform-specific integer. On Mac OS X the object file name is `foo.dylib` and on Windows the object file name is `foo.dll`.

See also

`concatenate-system`  
`defsystem`  
`load-system`
**concatenate-system**

**Function**

**Summary**

Produces a single, concatenated fasl from a `defsystem` system or systems.

**Package**

`scm`

**Signature**

`concatenate-system output system &key force simulate sim source-only args target-directory t-dir script-p => result`

`system ::= system-name*`

**Arguments**

- `output` The name of the required concatenated fasl.
- `system-name` The name of a system defined using `defsystem`.
- `simulate` Verbosity conditions, see Description for more detail.
- `sim` Same as `simulate`.
- `force` If `t`, then all files in the system will be concatenated.
- `source-only` If `t`, the source files of the system are concatenated.
- `target-directory` The directory to search for the object files.
- `t-dir` Same as `target-directory`.

**Values**

- `result` A list containing the name or names of the concatenated systems.

**Description**

This function produces a single, concatenated fasl, `output-file`, from a list of individual systems (named amongst the `args`).

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
files. However, output-file must, in such cases, be a MS-Win-
dow-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 sys-
tem’s default directory will be loaded instead.

See also compile-system
defsystem
load-system

current-pathname

Summary Computes a pathname relative to the current path.

Package lispworks

Signature current-pathname &optional relative-pathname type => pathname

Arguments relative-pathname A pathname designator.
type A string or nil. The default is nil.
Values  

\textit{pathname}  

A pathname.

Description  

The function \texttt{current-pathname} is useful for loading other files relative to a file.

\texttt{current-pathname} computes a pathname from the current operation as follows:

When loading a file

Uses \texttt{*load-pathname*}.

When compiling a file

Uses \texttt{*compile-file-pathname*}.

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 \texttt{relative-pathname} is merged with this physical pathname. The \texttt{pathname-type} of the result \texttt{pathname} is set to \texttt{type} if supplied, the \texttt{pathname-version} is set to \texttt{:newest}, and \texttt{pathname} is returned.

A useful value for \texttt{type} is \texttt{nil}, which can be used to allow \texttt{load} to choose between lisp or fasl regardless of the type of the current pathname.

\textbf{Note:} \texttt{defsystem} uses \texttt{current-pathname} with its \texttt{:default-host} argument.

Examples  

Suppose you want the file \texttt{foo} to load the file \texttt{bar}.

While loading the source file \texttt{foo.lisp}:

\begin{verbatim}
(current-pathname "bar")
\end{verbatim}
While loading the binary file \texttt{foo.ofasl}:

\begin{verbatim}
(current-pathname "bar")
\end{verbatim}

\Rightarrow

\begin{verbatim}
#P"C:/temp/bar.ofasl"
\end{verbatim}

To load \texttt{bar.lisp} or \texttt{bar.ofasl} according to the value of \texttt{*load-fasl-or-lisp-file*}, regardless of whether \texttt{foo.lisp} or \texttt{foo.ofasl} is being loaded, specify \texttt{type nil}:

\begin{verbatim}
(load (current-pathname "bar" nil))
\end{verbatim}

See also \texttt{defsystem}

\texttt{pathname-location}

\section*{defadvice}

\textbf{Macro}

\textbf{Summary} \hspace{1cm} Defines a new piece of advice.

\textbf{Package} \hspace{1cm} \texttt{lispworks}

\textbf{Signature} \hspace{1cm} \texttt{defadvice (dspec name advice-type &key where documentation)}

\begin{verbatim}
lambda-list &body body => nil
\end{verbatim}

\begin{verbatim}
dspects ::= fn-name |
         macro-name |
         (method generic-fn-name [(class*)])
\end{verbatim}

\begin{verbatim}
advice-type ::= :before | :after | :around
\end{verbatim}

\textbf{Arguments} \hspace{1cm} \texttt{dspec} \hspace{1cm} 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.

- **name**: 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.

- **advice-type**: A keyword specifying the kind of advice wanted.

- **where**: 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 `:end` then the advice is instead placed at the end of its section. The other permissible value for this argument is `:start`, which places the advice at the start of its section in the ordering (as in the default behavior).

- **documentation**: A string providing documentation on the piece of advice.
lambda-list 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.

body The main body of the advice.

Values defadvice returns nil.

Description defadvice 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 defadvice.

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.

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.
(or the original function if there is no after advice). The code associated with before and after advice should not destruc-
tively 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 specifi-
ers). 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**

```lisp
define-action name-or-list action-name data &rest specs =>
```

**Arguments**

- `name-or-list` A list or action list object.
- `action-name` A general lisp object.
- `data` An object.
- `specs` A list.

**Description**

The `define-action` macro adds a new action to the specified list; this action will be executed according to the action-list's execution-function (see `execute-actions`) when executed. If the action-list specified by `name-or-list` does not exist, then this is handled according to the value of `*handle-missing-action-list*`.

`name-or-list` is evaluated to give either a list UID (to be looked up in the global registry of lists) or an action list object. `action-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).

`data` specifies an object referring to data relevant to the action.

`specs` is a free-form list of ordering specifiers and extra keywords, used to control more details of how and when this action is executed.

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 `equalp` to the action-name), then the notification and subsequent handling of this attempt is controlled by the values in the list `*handle-existing-action-in-action-list*`. This is to prevent problems due to re-evaluating an action definition inappropriately. Notification and redefine behavior can be overridden by using the `:force` keyword argument. In this case, any required redefinition is performed unconditionally and without notification.
The following keywords are recognized in the specs argument:

- **:after**
  The following element in specs 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**

- **define-action**
- **undefine-action**

### define-action-list

**Macro**

**Summary**

Defines a registered action list.

**Package**
lispworks
define-action-list uid &key documentation sort-time dummy-actions default-order execution-function =>

uid
A Lisp object.
documentation
A string.
sort-time
One of :execute or :define-action.
dummy-actions
A list.
default-order
A list.
execution-function
A function.

The define-action-list macro defines an action list.

uid is a unique identifier, and must be a general Lisp object, to be compared by equalp. It names the list in the global registry of lists. See make-unregistered-action-list to create unnamed, “unregistered” action-lists. The uid may be quoted, but is not required to be. It is possible, but not recommended, to define an action-list with unique identifier nil. If a registered action-list with the uid already exists (that is, one which returns t when compared with equalp), then notification and subsequent handling is controlled by the value of the *handle-existing-action-list* variable.

The documentation string allows you to provide documentation for the action list.

sort-time is a keyword specifying when added actions are sorted for the given list — either :execute or :define-action (see *default-action-list-sort-time*).

dummy-actions 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

'(:beginning :middle :end)

default-order specifies default ordering constraints for subsequently defined action-items where no explicit ordering constraints are specified. An example is
'(: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*
undefined-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 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 sub-systems) of `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 :lsp-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

load-system
compile-system
concatenate-system
current-pathname
*defsystem-verbose*

*defsystem-verbose*

Summary

Controls the amount of messages printed by defsystem about system (re)definition.

Package

lispworks

Initial value

Description

The variable *defsystem-verbose* is a generalized boolean controlling the amount of messages printed by defsystem. When the value is true, the system prints messages about system definition and redefinition. The default value is t.

See also

defsystem

delete-directory

Summary

Deletes a directory.

Package

lispworks
The LISPWORKS Package

Signature

```
(delete-directory directory &optional error => result)
```

Arguments

- `directory`: A pathname designator.
- `error`: nil, :error or :no-error.

Value

- `result`: t or nil.

Description

The function `delete-directory` attempts to delete the directory `directory`. It returns t on success, and on failure either returns nil or signals an error.

`error` determines what happens when `delete-directory` fails. When `error` is nil (the default), if `directory` does not exist `delete-directory` returns nil, otherwise any failure causes an error to be signaled. If `error` is :no-error, `delete-directory` returns nil on any failure. If `error` is :error, any failure causes an error to be signaled.

Typical reasons for failures in `delete-directory` are that `directory` is not empty, or that the user does not have the right permissions.

---

**deliver**

*Function*

Summary

The main interface to the Delivery tools.

Package

lispworks

Signature

```
deliver function file level &rest keywords
```

Description

The function `deliver` is the main interface to the LispWorks delivery tools. You use it to create LispWorks executable applications and dynamic libraries.

For more information about Delivery including a detailed description of `deliver`, see the *LispWorks Delivery User Guide*. 
For information about invoking \texttt{deliver} using the IDE, see "The Application Builder" in the \textit{Common LispWorks User Guide}.

See also \texttt{save-image}

\*\texttt{describe-length}\* \textbf{Variable}

\begin{description}
\item[Summary] Determines how many attributes of a composite object are described.
\item[Package] \texttt{lispworks}
\item[Initial Value] 20
\item[Description] The variable \*\texttt{describe-length}\* controls how many attributes of a composite object the function \texttt{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 \*\texttt{describe-length}\* is \texttt{nil} then \texttt{describe} describes all of the attributes. Use this value only with care.

\textbf{Note}: the \texttt{describe} functionality is load-on-demand in the LispWorks image as shipped. Therefore if you have not done \texttt{(require \"describe\") or called \texttt{describe}, \*\texttt{describe-length}\* may be unbound.

See also \texttt{describe}
\end{description}

\*\texttt{describe-level}\* \textbf{Variable}

\begin{description}
\item[Summary] Controls the depth to which \texttt{describe} describes arrays, structures and conses.
\end{description}
<table>
<thead>
<tr>
<th>Package</th>
<th>lispworks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial Value</td>
<td>1</td>
</tr>
</tbody>
</table>

**Description**
The variable `*describe-level*` controls the depth to which the function `describe` describes arrays, structures and conses.

**Note:** the `describe` functionality is load-on-demand in the LispWorks image as shipped. Therefore if you have not do `(require "describe")` or called `describe`, `*describe-level*` may be unbound.

**Example**
```lisp
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)
```

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  
Function

Summary  
Makes a LispWorks dynamic library quit.

Package  
lispworks

Signature  
dll-quit &key kill-all-processes timeout output force => result, quit-output

Arguments  
kill-all-processes A generalized boolean.
timeout A positive integer or nil.
output An output stream designator.
force A generalized boolean.

Values  
result t or nil.
quit-output A string or nil.

Description  
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:

- 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.

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 Freelibray, or you can re-use it (without re-loading).
By default kill-all-processes is nil which means that, if there are other running processes, dll-quit just returns nil. If kill-all-processes is non-nil, dll-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.

dll-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 dll-quit does nothing and returns nil. If force is non-nil, dll-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, dll-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, dll-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.
The LISPWORKS Package

quit-output contains the output which was generated when output nil was passed. Otherwise quit-output is nil.

If dll-quit is called inside a recursive foreign callback, the LispWorks dynamic library quits only when the outermost callback returns.

Note: dll-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 dll-quit, and return.

Note: dll-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

dotted-list-length

Function

Summary
Similar to list-length

Package
lispworks

Signature
dotted-list-length list => result

Arguments
list A list.

Value
result An integer.
The function `dotted-list-length` performs the same action as `list-length`, except that if the last `cdr` is not `nil` then instead of signalling an error, it returns the number of `cons`es plus 1.

See also `dotted-list-p`

---

**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  unspecified  An unspecified value.

Description  The function **do-nothing** ignores its arguments and returns an unspecified value. It is useful as a function argument.

See also  **false**  **true**

*enter-debugger-directly*  Variable

Summary  Controls direct entry into the Debugger tool.

Package  lispworks

Initial value  nil

Description  The variable *enter-debugger-directly* is a generalised boolean which affects the behavior of the LispWorks IDE when an error is signalled outside of the Listener REPL.

Value **nil** causes an error notifier window to be displayed (from which you can abort, report a bug, or raise a Debugger tool).

A true value causes the Debugger tool to be displayed immediately, and no error notifier appears.

**Note:** Errors signalled in a Listener Read-Eval-Print loop are handled in the REPL and therefore *enter-debugger-directly* has no effect on the behavior in this case.

environment-variable  Function

Summary  Reads the value of an environment variable from the environment table of the calling process.
Package              lispworks
Signature             environment-variable name => result
Arguments             name             A string.
Values                result           A string or nil.
Description           The function environment-variable reads the environment variable specified by name and returns its value, or nil if the variable could not be found.
                       A setf method is also defined, allowing you to set the value of an environment variable:
                       (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 PATH is returned:
                       (environment-variable "PATH")
                       The result is a string of all the defined paths:
                       "c:\hqbin\nt\x86;c:\hqbin\nt\x86\perl;c:\hqbin\win32;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:

```lisp
(setf (environment-variable "MYTZONE") "GMT")
```

### errno-value

**Summary**
Returns the current value of the UNIX variable `errno`.  

**Package**
lispworks  

**Signature**
`errno-value => value`  

**Arguments**
None.  

**Values**

| value | The current value of `errno`. |  

**Description**
The function `errno-value` returns the current value of the UNIX variable `errno`.  

**Note:** this is implemented only on UNIX/Linux/Mac OS X.  

**Example**

```lisp
USER 10 > (errno-value)
2

USER 11 > (get-unix-error 2)
"no such file or directory"
```

**See also**
`get-unix-error`

### example-file

**Summary**
Returns a path in the `examples` folder.  

**Package**
lispworks  

**Signature**
`example-file file => path`
Arguments  

file  
A pathname designator.

Values  

path  
A pathname.

Description  

The function example-file returns an absolute path to a file file in the examples folder of the LispWorks library.

It does not actually test for the existence of the file.

Example  

```lisp
(example-file "capi/applications/othello.lisp")
=>
#P"C:/Program Files/LispWorks/lib/5-1-0-0/examples/capi/applications/othello.lisp"
```

See also  

example-compile-file

---

### example-compile-file

**Function**

**Summary**  

Compiles a file in the examples folder to a temporary output file.

**Package**  

lispworks

**Signature**  

```
example-compile-file file &rest args => output-truename, warnings-p, failure-p
```

**Arguments**  

file  
A pathname designator.

args  
Arguments passed to compile-file.

**Values**  

output-truename  
A pathname or nil.

warnings-p  
A generalized boolean.

failure-p  
A generalized boolean.

**Description**  

The function example-compile-file constructs the path to file in the examples 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  
\texttt{execute-actions name-or-list \&rest keyword-value-pairs \&rest other-args =>}

Arguments  
\begin{itemize}
  \item \texttt{name-or-list} \hspace{1cm} An action list
  \item \texttt{keyword-value-pairs} \hspace{1cm} See description.
  \item \texttt{other-args} \hspace{1cm} A list.
\end{itemize}

Description  
The \texttt{execute-actions} macro executes, in sequence, the actions on the specified list. If the action-list specified by \texttt{name-or-list} does not exist, then this is handled according to the value of \texttt{*handle-missing-action-list*}. Note that \texttt{name-or-list} is evaluated.

If a user-defined execution function was specified when the action list was defined, then it should accept the following arguments:

\begin{verbatim}
(action-list other-args \&rest keyword-value-pairs)
\end{verbatim}

Note that \texttt{other-args} is passed as a single list.

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 apply on \texttt{other-args}:

\begin{verbatim}
(apply data other-args)
\end{verbatim}

This behavior is modified by the keyword-value-pairs, thus:

- If the keyword parameter \texttt{:ignore-errors-p} is \texttt{non-nil}, 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 \texttt{:ignore-errors-p} is \texttt{nil} (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:
  :collect collect values into list
  :and return t only if all values are t. Return nil immediately if any value is nil
  :or return first non-nil value

See also  
define-action
  with-action-list-mapping

extended-char

Summary  The extended character type.

Package  lispworks

Signature  extended-char

Description  The type of extended characters. A synonym for extended-character, but with standard spelling.

extended-character

Summary  The extended character type.

Package  lispworks

Signature  extended-character

Description  The type of extended characters.
### extended-character-p

**Summary**
Tests if an object is an extended character.

**Package**
lispworks

**Signature**
```
extended-character-p object => bool
```

**Arguments**
`object` The object to be tested.

**Values**
`bool` 
- `t` if `object` is an extended character;
- `nil` otherwise.

**Description**
This is the predicate for extended characters.

**See also**
extended-character

### extended-char-p

**Summary**
Tests if an object is an extended character.

**Package**
lispworks

**Signature**
```
extended-char-p object => bool
```

**Arguments**
`object` The object to be tested.

**Values**
`bool` 
- `t` if `object` 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
The variable `*external-formats*` contains a list of the names of the defined external formats.

The platform-specific external format names are:

- **code-page**
  
  Uses the encoding in the Microsoft Windows code page specified by the `:id` parameter.

- **latin-portable**
  
  Intended for use when communicating with X servers, for example when passing XLFD 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: `ignore` 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: `pathname` A pathname, string, or file-stream.

Values: `bool` 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  \texttt{file-directory-p} tests whether the pathname represents a directory.

Examples  
\begin{verbatim}
CL-USER 70 > (file-directory-p "/")
T
CL-USER 71 > (file-directory-p ".login")
NIL
\end{verbatim}

\textbf{find-regexp-in-string}  \textit{Function}

Summary Matches a regular expression.

Package \texttt{lispworks}

Signature \texttt{find-regexp-in-string pattern string \&key start end from-end case-sensitive => pos, len}

Arguments \begin{itemize}
\item \textit{pattern} A string or a precompiled regular expression object.
\item \textit{string} A string.
\item \textit{start, end} Bounding index designators of \textit{string}.
\item \textit{from-end} A generalized boolean.
\item \textit{case-sensitive} A generalized boolean.
\end{itemize}

Values \begin{itemize}
\item \textit{pos} A non-negative integer or \texttt{nil}.
\item \textit{len} A non-negative integer or \texttt{nil}.
\end{itemize}

Description The function \texttt{find-regexp-in-string} searches the string \textit{string} for a match for the regular expression \textit{pattern}. The index in \textit{string} of the start of the first match is returned in \textit{pos}, and the length of the match is \textit{len}.

If \textit{from-end} is \texttt{nil} (the default value) then the search starts at index \textit{start} and ends at index \textit{end}. \textit{start} defaults to 0 and \textit{end}
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:

```lisp
(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 precompiled 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  function-lambda-list function &optional error-p => args

Arguments  function  A symbol or a function.
           error-p  A boolean.

Values  args  A list, or the symbol :none

Description  function is the function whose arguments are required
             If error-p is nil, then function-lambda-list returns :none if
             function is not defined, and does not start the debugger. The
             default value of error-p is t, meaning that an error is signalled
             if function is undefined.

Example  TEST 2 > (function-lambda-list 'editor:create-buffer-command)
          (EDITOR::P &OPTIONAL EDITOR::BUFFER-NAME)
get-inspector-values  

**Summary**
Customizes the information displayed in the Common LispWorks inspector tool.

**Package**
lispworks

**Signature**
get-inspector-values object mode

**Arguments**
object The object to be inspected.
mode Name of a mode, or nil. nil defines the default inspection format for object.

**Values**
Returns five values: names, values, getter, setter and type. names and values are the two lists displayed in columns in the inspector window. getter is ignored. setter is a function used to updated slot values. type is displayed at the foot of the inspector window.

**Description**
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 nil is the default mode, which is always present (it can be overwritten).

LispWorks includes methods for:

(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))

and so on.

**Example**
This example allows inspection of a CLOS object, displaying only direct slots form 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))))))
**get-unix-error**  
*Function*

**Summary**
Returns the text associated with a given error.

**Package**
lispworks

**Signature**
get-unix-error number => error

**Arguments**
number The *errno* value whose text is required.

**Values**
error The text associated with the error.

**Description**
The `get-unix-error` function returns the text associated with the specified value of the UNIX variable *errno*.

**Note:** this is implemented only on UNIX/Linux/Mac OS X/FreeBSD.

**See also**
errno-value

**`*grep-command*`**  
*Variable*

**Package**
lispworks

**Summary**
Determines the search utility used by Grep searches in the Search Files tool in the LispWorks IDE.

**Initial Value**
"grep" on Unix/Linux/Mac OS X/FreeBSD platforms.  
nil 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 nil, then the value of

(sys:lispworks-file "etc/grep")
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*`

---

**Variable**

**`*grep-command-format*`**

**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 /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**

lispworks

**Summary**

Arguments added to the command string of a Grep search in the Search Files tool.

**Initial Value**

"-n"

**Description**

The variable *grep-fixed-args* 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 Common LispWorks User Guide for more information about the Search Files tool.

See also

*grep-command*

*grep-command-format*

**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**

lispworks

**Initial value**

(:warn :redefine)

**Description**

A list containing one of :warn, or :silent, determining whether to notify the user, and one of :skip, or :redefine, to determine what to do about an action definition when the action already exists in the given action list.

It is used by define-action.
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 undefine-action.

See also  
undefine-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 LispWorks 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**  

`system-name`  
A symbol representing the name of the system. The system must have been defined using the `defsystem` macro.

`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 `simulate`:

- `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**  

`(hardcopy-system 'blackboard)`

`(hardcopy-system 'tms :simulate :ask :command "lpr")`
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.

---

### lisp-image-name

**Function**

**Summary**

Returns the name of the running image.

**Package**

`lispworks`

**Signature**

`lisp-image-name => name`

**Arguments**

None.

**Values**

`name` A string.

**Description**

The function `lisp-image-name` 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:

```
CL-USER 1 > (lisp-image-name)
"C:\Program Files\LispWorks\lispworks-5-1-0-x86-win32.exe"
```

On Linux:

```
CL-USER 1 > (lisp-image-name)
"/usr/bin/lispworks-5-1-0-x86-linux"
```

**See also**

*`line-arguments-list`*
<table>
<thead>
<tr>
<th><strong>variable</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><em>lispworks-directory</em></td>
<td>The variable <em>lispworks-directory</em> holds the name of the directory where various files important for the running of LispWorks are located. When LispWorks starts in a directory which contains an appropriate numbered subdirectory such as lib/5-1-0-0/, then it assumes this is the LispWorks installation directory and sets <em>lispworks-directory</em> accordingly. Additionally, LispWorks for Macintosh running on Cocoa looks for such a subdirectory in the Library folder alongside its application bundle, and if found it sets <em>lispworks-directory</em> accordingly. On non-Windows platforms, LispWorks then consults the Unix environment variable LISPWORKS_DIRECTORY. If this is set, then <em>lispworks-directory</em> is set accordingly. The lib/5-1-0-0/ subdirectory of <em>lispworks-directory</em> should include these subdirectories: config, which contains the configuration files.</td>
</tr>
</tbody>
</table>

**Summary** | The main LispWorks installation directory. |
**Package** | lispworks |
**Initial value** | The initial value is

- #P"/usr/lib/lispworks/" on Unix.
- #P"/usr/local/lib/LispWorks/" on Linux (for an installation from the tar archive) or FreeBSD.
- #P"C:\Program Files\LispWorks\" on Microsoft Windows.
- #P"/Applications/LispWorks 5.1/Library/" on Mac OS X.

Note however that the value can be set when configuring an image or on startup.
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.

---

### load-all-patches

**Function**

**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` Function

**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**

`lispworks`

**Signature**

```lisp
load-system system-name &key force simulate source-only target-directory => nil
```

**Arguments**

- `system-name`: A symbol representing the name of the system. The system must have been defined using the `defsystem` macro.
- `force`: If `t` then all the files in the system are loaded regardless. (This argument was formerly called `force-p`. The old name is currently still accepted for compatibility.)
- `simulate`: If `nil` or not present then `load-system` works silently. Otherwise a plan of the actions which `load-system` intends to carry out is printed. What happens next depends on the value of `simulate`:
  - `t` — do nothing.
  - `:ask` — you are asked, using `y-or-n-p`, 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

(lossystem 'blackboard)

(lossystem '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 foo.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
lispworks

Signature
make-unregistered-action-list &key documentation sort-time
dummy-actions default-order execution-function =>

Arguments
documentation A string.

sort-time One of :execute or :define-action.

dummy-actions A list.

default-order A list.

execution-function A function.

Description
Return an action-list not registered in the global registry of lists. The keyword arguments are as for define-action-list.

The documentation string allows you to provide documentation for the action list.

sort-time is a keyword specifying when added actions are sorted for the given list — either :execute or :define-action (see *default-action-list-sort-time*).

dummy-actions 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

'(:beginning :middle :end)

default-order specifies default ordering constraints for subsequently defined action-items where no explicit ordering constraints are specified. An example is

'(: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 define-action-list
*handle-warn-on-redefinition*

---

**make-mt-random-state**

*Function*

**Summary**
Creates an object of type `mt-random-state`.

**Package**
lispworks

**Signature**
make-mt-random-state &optional state => new-state

**Arguments**
state nil, t or an object of type `mt-random-state`. The default is nil.

**Values**
new-state A new object of type `mt-random-state`.

**Description**
The function `make-mt-random-state` creates a new object of type `mt-random-state` which is suitable for use as the value of `*mt-random-state*`.

If `state` is an object of type `mt-random-state`, then `new-state` is a copy of `state`. If `state` is nil, then `new-state` is a copy of the value of `*mt-random-state*`. If `state` is t then `new-state` is an object of type `mt-random-state` initialized using a call to `get-universal-time`.

`make-mt-random-state` is analogous to `cl:make-random-state`. 
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.

random-number is generated using the Mersenne Twister algorithm published by Makoto Matsumoto and Takuji Nishimura at

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*
**Variable**

*mt-random-state*

- **Summary**: The default random state used by `mt-random`.
- **Package**: lispworks
- **Description**: The variable *mt-random-state* contains an object of type `mt-random-state` which is the default state used by `mt-random` if a state is not supplied.

*mt-random-state* is analogous to `cl:*random-state*.

- **See also**: make-mt-random-state, mt-random, mt-random-state

---

**Type**

`mt-random-state`

- **Summary**: The type of objects containing state information used by `mt-random`.
- **Package**: lispworks
- **Description**: The Mersenne Twister pseudo-random number generator uses state data contained in a object of type `mt-random-state`.

`mt-random-state` is analogous to `cl:random-state`.

- **See also**: *mt-random-state*, mt-random, mt-random-state-p

---

**Function**

`mt-random-state-p`

- **Summary**: The predicate for objects of type `mt-random-state`. 
Package | lispworks
Signature | mt-random-state-p arg => result
Arguments | arg | An object.
Values | result | A boolean.
Description | The function \texttt{mt-random-state-p} returns \texttt{t} if \textit{arg} is an object of type \texttt{mt-random-state}, and \texttt{nil} otherwise. \texttt{mt-random-state-p} is analogous to \texttt{cl:random-state-p}.
See also | \texttt{mt-random-state}

\textbf{pathname-location} \hspace{2cm} \textit{Function}

Summary | Returns the location of a file.
Signature | \texttt{pathname-location pathname => location}
Arguments | \textit{pathname} | A pathname designator.
Values | \textit{location} | A pathname.
Description | The function \texttt{pathname-location} returns a pathname \textit{location} that represents the directory where the file \textit{pathname} resides. Each of the name, type and version components of \textit{location} are \texttt{nil}.
Example | Due to the ANSI Common Lisp definition of the \texttt{directory} function and the fact that LispWorks returns fully specified true-names, the form
\begin{verbatim}
(directory (truename "/tmp/"))
\end{verbatim}
will always signal an error or return the list \begin{verbatim} (#P "/tmp/"
\end{verbatim}. To obtain the contents of the \begin{verbatim} /tmp\end{verbatim} directory, use the form
(directory (pathname-location (truename "/tmp/")))

See also  
current-pathname  
directory

**precompile-regexp**  
*Function*

**Summary**  
Precompiles a regular expression object.

**Package**  
lispworks

**Signature**  
precompile-regexp string => pattern

**Arguments**  
string A string.

**Values**  
pattern A precompiled regular expression object.

**Description**  
The function `precompile-regexp` returns a precompiled regular expression object suitable for passing as `pattern` to `find-regexp-in-string`.

See also  
find-regexp-in-string

**print-actions**  
*Function*

**Summary**  
Generates a listing of the action items on a given action list in order.

**Package**  
lispworks

**Signature**  
print-actions name-or-list &optional stream

**Arguments**  
name-or-list An action list.
stream A stream.
Description
Generates a listing of the action items on this action-list, in order. If the action-list specified by name-or-list does not exist, then this is handled according to the value of *handle-missing-action-list*.

stream is an optional argument specifying where to print the output. The default value of stream is the value of *standard-output*.

See also
print-action-lists

print-action-lists

Function

Summary
Prints a list of all the actions lists in the global registry.

Package
lispworks

Signature
print-action-lists &optional stream

Arguments
stream A stream.

Description
Generates a listing of all the action lists in the global registry. The ordering of the action lists is random.

stream is an optional argument specifying where to print the output. The default value of stream is the value of *standard-output*.

See also
print-actions

*print-command*

Variable

Summary
A command used for some printing operations.

Package
lispworks

375
Initial Value  "print" on Windows.
              "lpr" on UNIX/Linux/Mac OS X/FreeBSD systems.

Description  This variable is used as the command sent by LispWorks to
              the shell in hardcopy-system.

See also    hardcopy-system

*print-nickname*  Variable

Summary  Controls the package prefix used when a symbol is printed.

Package  lispworks

Initial Value  nil

Description  The variable *print-nickname* controls which package pre-
              fix is used when a symbol is printed and the symbol’s pack-
              age needs to be output.

              If *print-nickname* 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 package-nicknames) is out-
              put. Otherwise, the package name is output.

*prompt*  Variable

Summary  Defines the LispWorks listener prompt.

Package  lispworks

Initial Value  "~%~A ~D~[~;~:* : ~D~] > ”

Description  The variable *prompt* 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 "~&
      (subseq *default-
      (sys:get-user-name)
      #\Space
      2)))))
  nil)
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.
The function **quit** exits LispWorks unless the user cancels the operation.

There are two stages which may allow the user the chance to cancel.

1. First the action items of the action list "**Confirm when quitting image**" are run. If any action item returns **nil**, then LispWorks does not exit.

2. Otherwise, if **confirm** is true (the default value is **nil**) then a question like
   
   "**Do you really want to exit LispWorks?**"
   
   is presented to the user. If the answer No is supplied, then LispWorks does not exit. Otherwise, the action items of the action list "**When quitting image**" are run, and then LispWorks exits, and the value **status** is returned to the Operating System as the exit value of the LispWorks process. The default value of **status** is 0.

If **ignore-errors-p** is true, then any error signalled during the running of the action list items or the confirm prompt is ignored and **quit** proceeds to exit the image. If **ignore-errors-p** is **nil** 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 **ignore-errors-p** is **nil**.

If **return** is true and LispWorks is going to exit, then **quit** returns **t**. 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 **return** is **nil** then **quit** does not return. The default value of **return** is **nil**.

**Note:** To make a Cocoa application quit cleanly from inside the **Quit** menu command you need to call **capi:destroy** on the application interface instead of calling **quit**. See
**rebinding**

*Macro*

**Summary**
Ensures unique names for all the variables in a groups of forms.

**Package**
lispworks

**Signature**
rebinding (&rest vars) &body body => form

**Arguments**
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 body wrapped in a form which creates a unique name for each of the variables (compare with gensym) 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

```lisp
(defmacro lister (x y)
  (rebinding (x y)
    '(list ,x ,y)))
```

the form (lister i j) macroexpands to

```lisp
(LET* ((#:X-77 I)
    (#:Y-78 J))
  (LIST #:X-77 #:Y-78))
```
See also \texttt{with-unique-names}

\textbf{regexp-find-symbols} \hspace{1cm} Function

\textbf{Summary} \hspace{1cm} Returns a list of symbols that match a supplied regular expression.

\textbf{Package} \hspace{1cm} \texttt{lispworks}

\textbf{Signature} \hspace{1cm} \texttt{regexp-find-symbols regexp-string &key case-sensitive packages test external-only => symbols}

\textbf{Arguments} \hspace{1cm} \texttt{regexp-string} \hspace{0.5cm} A string.
\hspace{1cm} \texttt{case-sensitive} \hspace{0.5cm} A boolean.
\hspace{1cm} \texttt{packages} \hspace{0.5cm} A list of package designators, a single package designator, or the keyword \texttt{:all}.
\hspace{1cm} \texttt{test} \hspace{0.5cm} A function of one argument returning a boolean result.
\hspace{1cm} \texttt{external-only} \hspace{0.5cm} A generalized boolean.

\textbf{Values} \hspace{1cm} \texttt{symbols} \hspace{0.5cm} A list of symbols.

\textbf{Description} \hspace{1cm} The function \texttt{regexp-find-symbols} returns a list of symbols that match the regular expression in \texttt{regexp-string}.

\texttt{case-sensitive} determines whether the match is case sensitive. The default value of \texttt{case-sensitive} is \texttt{nil}.

\texttt{packages} specifies in which packages to search. The default value of \texttt{packages} is \texttt{:all}, meaning search in all packages.

\texttt{test}, if supplied, must be a function of one argument, which returns \texttt{t} if the argument should be returned, and \texttt{nil} otherwise. The function \texttt{test} is applied to each symbol that matches \texttt{regexp-string}, and if it returns \texttt{nil} the symbol is not included
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.

**name**

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**

remove-advice returns nil.

**Description**

remove-advice 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 defadvice. 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).

hcl:delete-advice is a macro, identical in effect to remove-advice, except that you do not need to quote the arguments.

**Notes**

remove-advice is an extension to Common Lisp.

**See also**

defadvice
delete-advice

**removelf**

*Macro*

**Summary**

Removes an item from a sequence.

**Package**
lispworks
Signature  \texttt{removef place item &key test test-not start end key => result}

Arguments  \begin{itemize}
\item \textit{place} \hspace{1cm} A place.
\item \textit{item} \hspace{1cm} An object.
\item \textit{test} \hspace{1cm} A test function.
\item \textit{test-not} \hspace{1cm} A test function.
\item \textit{start} \hspace{1cm} An integer.
\item \textit{end} \hspace{1cm} An integer or \texttt{nil}.
\item \textit{key} \hspace{1cm} A key function.
\end{itemize}

Values  \begin{itemize}
\item \textit{result} \hspace{1cm} A sequence.
\end{itemize}

Description  The modifying macro \texttt{removef} removes an item from a sequence using \texttt{remove}. See \texttt{remove} for more details.

See also  \texttt{append}

\begin{itemize}
\item \textbf{\texttt{*require-verbose*}} \hspace{1cm} \textit{Variable}
\end{itemize}

Summary  Controls the output of \texttt{require}.

Package  \texttt{lispworks}

Initial value  \texttt{t}

Description  The variable \texttt{*require-verbose*} is a generalized boolean controlling whether \texttt{require} prints the names of the files which are being loaded.
round-to-single-precision  

Function

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
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.

The default size on Windows and Linux is 64 bits as specified by the IEEE standard.

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 (coerce float 'double-float)

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
CL-USER 197 > pi
3.141592653589793D0

CL-USER 198 > round-to-single-precision pi
3.1415927410125732D0
sbchar

Function

Summary
The accessor for simple base strings.

Package
lispworks

Signature
sbchar string index => value

Arguments
string A simple-base-string.
index An index.

Values
value The character in string at index.

Description
This is the accessor for simple base strings. setf is allowed.

See also
simple-base-string

set-default-character-element-type

Function

Summary
Configures the value of lw:*default-character-element-type*.

Package
lispworks

Signature
set-default-character-element-type type => type-defaults

Arguments
type A character type. This can take any of the values base-char; lw:simple-char and character

Values
type-defaults The new value of lw:*default-character-element-type*.

Description
The function set-default-character-element-type sets the value of lw:*default-character-element-type*, 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

- `string`
- `open`
- `*default-character-element-type*`
- `with-output-to-string`

---

**simple-base-string-p**

*Function*

**Summary** Tests if an object is a simple base string.

**Package** lispworks

**Signature** `simple-base-string-p object => bool`

**Arguments**

- `object` The object to be tested.

**Values**

- `bool` `t` if `object` is a simple base string; `nil` otherwise.

**Description** This is the predicate for simple base strings.

**See also** `simple-base-string`
simple-char

Type
Summary The simple character type.
Package lispworks
Signature simple-char
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 lispworks
Signature simple-char-p object => bool
Arguments object The object to be tested.
Values bool t if object is a simple character; nil otherwise.
Description The predicate for simple characters.
See also simple-char

simple-text-string

Type
Summary The simple text string type.
Package lispworks
Signature simple-text-string length
Arguments length The length of the string (or *, meaning any).

Description This is the simple version of text-string, that is, the string itself is simple. Equivalent to:

\[(\text{simple-vector lw:simple-char length})\]

See also text-string

**simple-text-string-p**

Function

Summary Tests if an object is a simple text string.

Package lispworks

Signature simple-text-string-p object => bool

Arguments object The object to be tested.

Values bool t if object is a simple text string; nil otherwise.

Description This is the predicate for simple text strings.

See also simple-text-string

**start-tty-listener**

Function

Summary Starts a listener in the startup shell.

Package lispworks

Signature start-tty-listener force => process

Arguments force A generalized boolean.
Values

process A listener process, or nil.

Description

The function `start-tty-listener` returns a process that runs a listener read-eval-print loop connected to `*terminal-io*`. If `force` is `nil`, then `start-tty-listener` checks if the default listener process is alive or if there is a live process with name "TTY Listener". If such a process exists, `start-tty-listener` simply returns `nil` and does not start a new process. If no such process exists, or if `force` was `t`, then `start-tty-listener` starts a new listener process named "TTY Listener", and returns it.

If a REPL with I/O through `*terminal-io*` (such as a REPL started by `start-tty-listener`) is in the debugger, then by default it blocks multiprocessing. This behavior is controlled by the value of `*terminal-debugger-block-multiprocessing*`.

See also `*terminal-debugger-block-multiprocessing*`

---

**stchar**

*Function*

Summary The accessor for simple text strings.

Package `lispworks`

Signature `stchar string index => value`

Arguments

`string` A `simple-text-string`.
`index` An index.

Values `value` The character in `string` at `index`.

Description This is the accessor for simple text strings. `setf` is allowed.

See also `simple-text-string`
The string-append Function

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
The string-append function takes any number of string designators and constructs a single string from them.

A string designator is a string, a symbol or a character object.

Each of the elements of the strings argument are first coerced into a string using the string function if they are not already a string.

string is a string of the "widest" type amongst strings. That is, the constructed string is of the same type as the argument with the largest element type.

Example
(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
text-string

Summary The text string type.

Package lispworks

Signature `text-string length`

Arguments `length` The length of the string (or *, meaning any).

Description The type of strings that can hold any simple character, that is, `(vector lw:simple-char length)`. 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 16-bit-string.

See also 8-bit-string
16-bit-string

text-string-p

Summary Tests if an object is a text string.

Package lispworks

Signature `text-string-p object => bool`

Arguments `object` The object to be tested.

Values `bool t` if `object` is a text string; `nil` otherwise.

Description This is the predicate for text strings.

See also text-string
true

Function

Summary
Ignores its arguments and returns t.

Package
lispworks

Signature
true &rest ignore => t

Arguments
ignore
All arguments are ignored.

Values
t

Description
The function true ignores all its arguments and returns t. It is useful as a functional argument.

See also
do-nothing
false

undefine-action

Macro

Summary
Removes an action from a specified list.

Package
lispworks

Signature
undefine-action name-or-list action-name =>

Arguments
name-or-list
A list or action list object.
action-name
A general lisp object.

Description
The undefine-action macro removes the specified action from the specified list. If the action specified by action-name does not exist, then this is handled according to the value of *handle-missing-action-in-action-list*.

name-or-list is evaluated to give either a list UID (to be looked up in the global registry of lists) or an action list object. action-
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 lispworks

Signature undefine-action-list uid =>

Arguments uid A lisp object.

Values None.

Description The undefine-action-list flushes the specified list (and all its action-items). If the action-list specified by uid does not exist, then handling is controlled by the value of the *handle-missing-action-list* variable.

See also define-action-list

user-preference  

Function

Summary Gets or sets a persistent value in the user’s registry.

Package lispworks

Signature user-preference path value-name $key product => value, valuep
The LISPWORKS Package

Signature

(setf user-preference) value path value-name &key product => value

Arguments

path A string or a list of strings.
value-name A string.
product A keyword.

Values

value A Lisp object.
valuep A boolean.

Description

The function user-preference reads the value of the registry entry value-name under path under the registry path defined for product by (setf product-registry-path). If the registry entry was found a second value t is returned. If the registry entry was not found, then value is nil.

The function (setf user-preference) sets the value of that registry entry to value.

If path is a list of strings, then it is interpreted like the directory component of a pathname. If path 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.

Note: when value is a string, user-preference stores a print-escaped string in the registry and reads it back with read-from-string. Therefore it may not work with string values stored by other software.

Note: while product can in principle be any Lisp object, values of product are compared by eq, so you should use keywords.

Note: The CAPI provides a way to store window geometry - see the entry for capi:top-level-interface-save-geometry-p in the LispWorks CAPI Reference Manual.

Example

This example is on Microsoft Windows. Note the use of backslashes as directory separators in the path 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 | lispworks
---|---
Signature | **when-let** (var form) &body body => result
Arguments | var | A variable whose value is used in the evaluation of body.
form | A form, which must evaluate to non-nil.
body | A body of code to be evaluated conditionally on the result of form.
Values | result | The result of evaluating body using the value var.
Description | This macro executes the body of code if the form evaluates to a non-nil result. Within the body, the variable var is bound to the result of the form.
Example | The form
(when-let (position (search string1 string2))
  (print position))
macroexpands to
(let ((position (search string1 string2)))
  (when position
    (print position)))
See also | when-let*

**when-let***  
*Macro*

Summary | Executes a body of code if a series of forms evaluates to non-nil, propagating the results of the forms through the body of code.
Package | lispworks
Signature  \texttt{when-let* \ bindings \ \&body \ \ body => \ result}

\[\text{bindings} ::= ((\text{var} \ \text{form})^*)\]

Arguments  \begin{itemize}
  \item \texttt{var} A variable whose value is used in the evaluation of \texttt{body}.
  \item \texttt{form} A form, which must evaluate to non-\texttt{nil}.
  \item \texttt{body} A body of code to be evaluated conditionally on the result of \texttt{form}.
\end{itemize}

Values  \texttt{result} The result of evaluating \texttt{body} using the value \texttt{var}.

Description  The macro \texttt{when-let*} expands into nested \texttt{when-let} forms.

The bindings are evaluated in turn as long as each returns non-\texttt{nil}. If the last binding evaluates to non-\texttt{nil}, \texttt{body} is executed. Within the code \texttt{body}, each variable \texttt{var} is bound to the result of the corresponding form \texttt{form}.

Example  \begin{verbatim}
(defmacro divisible (n &rest divisors)
  `(when-let* ,,(loop for div in divisors
     collect (list (gensym)
       (zerop (mod n div)))))
  t))
\end{verbatim}

See also  \texttt{when-let}

\begin{center}
\textbf{whitespace-char-p}
\end{center}

Function  Tests whether a character represents white space.

Summary  Tests whether a character represents white space.

Package  \texttt{lispworks}

Signature  \texttt{whitespace-char-p \ char => \ bool}

Arguments  \begin{itemize}
  \item \texttt{char} A character.
\end{itemize}
Values

| **bool** t | if char represents white space; **nil** otherwise |

Description

This predicate recognizes [whitespace], as described in the standard:

“Space and non-graphic characters that only moved the print position.”

If **sys:*extended-spaces* is t, U+3000 Ideographic Space is also considered whitespace.

See also ***extended-spaces***

**with-action-item-error-handling**

Macro

Summary

Executes a body of code across action lists and items, signalling errors and then continuing to the next action item.

Package

lispworks

Signature

**with-action-item-error-handling** action-list-var action-item-var ignore-errors-p &body body

Arguments

| **action-list-var** | A variable. |
| **action-item-var** | A variable. |
| **ignore-errors-p** | A boolean. |
| **body** | A body of Lisp code. |

Description

The **with-action-item-error-handling** macro executes the **body** with **action-list-var** and **action-item-var** are bound to the action list and item respectively. If **ignore-errors-p** is set to **t** 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 exe-
cuting 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.
data-var  A Lisp symbol.
post-process  A keyword.
body  A body of Lisp code.

Description  The with-action-list-mapping macro maps over an action-list's action-items. During execution, the symbols specified for item-var and data-var are bound to the executing action-item and its data respectively. See execute-actions for more on post-processing.

If this function is invoked with the keyword argument :post-process :collect, a list the values returned by each action-item's setf operation are returned.

Examples

(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)))))

See also  execute-actions

with-unique-names  
Macro

Summary  Returns a body of code with each specified name bound to a similar name.

Package  lispworks

Signature  with-unique-names (&rest names) &body body => result
Arguments

*names*

The names to be rebound in *body*.

*body*

The body of code within which *names* are rebound.

Values

*result*

The result of evaluating *body*.

Description

Returns the body with each *name* bound to a symbol of a similar name (compare *gensym*).

Example

After defining

```lisp
(defun 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

```lisp
(let* ((#:X-88 (x-function))
       (#:Y-89 (y-function))
       (LIST i j #:X-88 #:Y-89))
```

See also

*rebinding*
10 The LISPWORKS 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*.

**change-process-priority**

*Function*

**Summary**  Changes the priority of a process.

**Package**  mp

**Signature**  change-process-priority process new-priority => new-priority

**Arguments**  
- process  A process.
- new-priority  A fixnum.

**Description**  Changes the priority of process to be new-priority.

**See also**  process-priority
**create-simple-process**

*Function*

**Summary**

Creates and returns a simple process, which is a process with no stack of its own.

**Signature**

```lisp
create-simple-process name function wait-function &key
function-arguments wait-function-arguments priority => process
```

**Package**

`mp`

**Arguments**

- `name`: A string or symbol
- `function`: A function
- `wait-function`: A function
- `function-arguments`: A list of arguments for `function`
- `wait-function-arguments`: A list of arguments for `wait-function`
- `priority`: A fixnum

**Values**

- `process`: A simple process

**Description**

The `create-simple-process` function creates and returns a simple process, which is a process that has no stack of its own.

The `name` argument is a string or symbol that names the process. The `function` argument is a function to be run in the process, and the `wait-function` argument is a wait function that determines when the process function is run. The value of `function-arguments` is a list of arguments to which the process function is applied. The value of `wait-function-arguments` is a list of arguments to which the wait function is applied. The `priority` argument is a fixnum that specifies a priority for the process. The default priority is the value of `*default-simple-process-priority*`, and is usually 0.
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*. 
The following example creates a simple process that prints the value of *a* to the background output when the value is other than nil. The process function then sets *a* to nil. From a listener, the value of *a* can be set to trigger the process to run once and then sleep again.

```lisp
(defvar *a* 'i)

(defun a ()
  (let ((a *a*))
    (setq *a* nil)
    (format mp:*background-standard-output*
            "*a* is ~a~" a)))

(defun b () *a*)

(setq r (mp:create-simple-process 'test-proc 'a 'b))

<MP::SIMPLE-PROCESS Name TEST-PROC Priority 0 State NIL>

See also process-run-function

**current-process**

*Summary*

Contains the object that is the current process.

*Package*  

mp

*Description*

This special variable contains the object that is the current process.

See also get-current-process
**debug-other-process**  
*Function*

**Summary**  
Debug a thread other than the current process.

**Package**  
mp

**Signature**  
`debug-other-process process`

**Arguments**  
`process`  
A process or a string.

**Description**  
The function `mp:debug-other-process` causes the debugger to be entered in the given process. If `process` is a string, the process is found as if by `mp:find-process-from-name`. The list of process names can be found via `mp:ps`.

**See also**  
`find-process-from-name`

`ps`

---

**{*default-process-priority*}**  
*Variable*

**Summary**  
The default priority for processes.

**Package**  
mp

**Description**  
The {*default-process-priority*} variable contains the default priority for processes.

**See also**  
`process-run-function`

`create-simple-process`

{*default-simple-process-priority*}

---

**{*default-simple-process-priority*}**  
*Variable*

**Summary**  
The default priority for simple processes.
The *default-simple-process-priority* variable contains the default priority for simple processes.

See also
create-simple-process
*default-process-priority*

**ensure-process-cleanup**

**Function**

Summary
Run forms when a given process terminates.

Package
mp

Signature
\texttt{ensure-process-cleanup cleanup-form \&optional process =>}

Arguments
\begin{itemize}
  \item \texttt{cleanup-form} Form to run when \texttt{process} terminates.
  \item \texttt{process} The process to watch for termination. By default, this is the value of \texttt{mp:*current-process*}.
\end{itemize}

Values
None.

Description
Ensures that the \texttt{cleanup-form} 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 \texttt{car} is applied to the process and the \texttt{cdr} of the list.

When adding cleanup forms, this function uses \texttt{equal} to ensure that the form is only added once.

Example
A process calls \texttt{add-process-dependent} each time a dependent object is added to a process. When the process terminates, \texttt{inform-dependent-of-dead-process} 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
The function `get-current-process` returns the actual process in which it is called. In this respect it differs from `*current-process*`, which can be bound to another process. In particular, when a process A calls the `wait-function` of process B, in the `wait-function` `get-current-process` returns the process A, but `*current-process*` is bound to process B.

`result` is `nil` if multiprocessing is off.

See also `*current-process*`

---

### get-process

**Function**

**Summary**

Returns a process corresponding to a supplied designator.

**Package**

`mp`

**Signature**

`get-process process-designator => process`

**Arguments**

`process-designator`

- A `mp:process`, a string, a stack-group, a function, a symbol or a fixnum.

**Values**

`process`

- A `mp:process`.

**Description**

The function `get-process` returns a process according to the supplied `process-designator`, which is interpreted as follows:

- `mp:process` Return it.
- A string Find the first process (highest priority) with matching name. Process names are compared by `string=`.
A stack-group Return the process of the stack-group.

A function Return the first process that has process-designator as its function (that is, the third argument of process-run-function).

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 process-designator is its unique id. The unique id of the current process can be found by (sys:current-thread-unique-id).

result is nil if multiprocessing is off.

See also find-process-from-name

initialize-multiprocessing Function

Summary Initializes multiprocessing before use.

Package mp

Signature initialize-multiprocessing &rest main-process-args => nil

Arguments main-process-args

A set of arguments for process-run-function.

Values Returns nil.

Description The function initialize-multiprocessing 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* Special Variable**

**Summary**
A list of the processes the system initializes on startup.

**Package**
mp

**Description**
The variable *initial-processes* specifies the processes which the system initializes on startup.

Each element of the *initial-processes* list is a set of arguments for process-run-function.
Example

To create a listener process as well as your own processes, evaluate this form before saving your image:

```lisp
(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.
The MP Package

Example

```lisp
CL-USER 71 > (pprint (mp:list-all-processes))

(#<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**

**Function**

**Summary**

Returns the name of a lock.

**Package**

`mp`

**Signature**

`lock-name lock => name`

**Arguments**

`lock` A lock object

**Values**

`name` A string

**Description**

The `lock-name` function takes a lock object as its argument and returns the name of the lock object.

**Example**

```lisp
(let ((lock (mp:make-lock :name "my lock")))
  (mp:lock-name lock))
=> "my lock"
```
See also  
make-lock  
with-lock  
process-lock  
process-unlock  
lock-owner

lock-owner  

Function  

Summary  Returns the owner of a lock.

Package  mp

Signature  lock-owner lock => result

Arguments  lock  A lock object

Values  result  A process, t or :unknown

Description  The lock-owner function returns the process that currently owns the lock, or nil.

If lock is locked then result is normally the process that locked it. If lock was locked while multiprocessing was not running then result is t. Also, if lock was locked by an unknown process (for example, the process is killed whilst holding the lock) then result is :unknown.

result is nil if lock is not locked.
Example

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">

See also
make-lock
with-lock
process-lock
process-unlock
lock-name

mailbox-empty-p

Function

Summary Tests whether a mailbox is empty.

Package mp

Signature mailbox-empty-p mailbox => bool

Arguments mailbox A mailbox

Values bool A generalized boolean

Description The mailbox-empty-p function returns t if the given mailbox is empty and nil otherwise.

See also mailbox-send
mailbox-peek
mailbox-read
make-mailbox
**mailbox-peek**  

*Function*

**Summary**  
Peeks at the first object in a mailbox.

**Package**  
mp

**Signature**  
`mailbox-peek mailbox => result`

**Arguments**  
`mailbox` A mailbox.

**Values**  
`result` Any object or nil.

**Description**  
The `mailbox-peek` function returns the first object in the mailbox without removing it. If the mailbox is empty, nil is returned.

**See also**  
`mailbox-empty-p`  
`mailbox-send`  
`mailbox-read`  
`make-mailbox`

---

**mailbox-read**  

*Function*

**Summary**  
Reads the next object in a mailbox.

**Package**  
mp

**Signature**  
`mailbox-read mailbox &optional wait-reason timeout => object`

**Arguments**  
`mailbox` A mailbox.

`wait-reason` A string or nil.

`timeout` A non-negative number or nil.

**Values**  
`object` Any object.
The mailbox-read function returns the next object from the mailbox mailbox, or nil.

If mailbox is empty and timeout is nil, then mailbox-read blocks until an object is placed in mailbox. If mailbox is empty and timeout is a number, then mailbox-read blocks until an object is placed in mailbox or timeout seconds have passed. If the timeout occurs, then mailbox-read returns nil.

The wait-reason argument (or the string "Waiting for message" if wait-reason is nil) and the timeout argument are both passed to process-wait-with-timeout.

The default value of wait-reason is nil and the default value of timeout is nil.

See also mailbox-empty-p mailbox-peek mailbox-send mailbox-wait-for-event make-mailbox process-wait-with-timeout

**mailbox-reader-process**

Function

Returns the reader process of the mailbox.

**Summary**

**Package** mp

**Signature** mailbox-reader-process mailbox => process

**Arguments**

mailbox A mailbox

**Values**

process A process

**Description**

The mailbox-reader-process function returns the reader process of mailbox.
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.
process-other-messages-p
A generalized boolean.

no-hang-p
A generalized boolean.

stop-at-user-operation-p
A generalized boolean.

Values

result
An event or nil.

Description
The function mailbox-wait-for-event waits for an event in a mailbox in a "windowing friendly" way. It reads an event from the mailbox mailbox. If there is no event in the mailbox, it waits for an event (unless no-hang-p is true).

The value event is any object that was put in the mailbox, or nil if the mailbox is empty, possibly after waiting.

mailbox-wait-for-event 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 process-other-messages-p is true it processes Windows messages while it is waiting. The default value of process-other-messages-p is t.

wait-function is the wait function to be used, which is called with the mailbox mailbox as its argument. If wait-function is not supplied, a function that returns t when the mailbox is not empty is used is used internally.

wait-reason is used as the wait reason if it needs to wait. The default value of wait-reason is "Waiting for an event".

process-other-messages-p controls processing of other messages. On Microsoft Windows this means Windows messages. On other platforms it has no effect.

no-hang-p controls whether mailbox-wait-for-event should really wait. If no-hang-p is true and there is no event, it returns immediately except on Microsoft Windows, where it may
first process all Windows messages (depending on the value of \textit{process-other-messages-p}). The default value of \textit{no-hang-p} is \texttt{nil}.

\textit{stop-at-user-operation-p} on Microsoft Windows causes \texttt{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 \textit{stop-at-user-operation-p} is \texttt{nil}.

If \texttt{mailbox-wait-for-event} is called when not Lisp is not multiprocessing, it returns immediately. The return value is an event or \texttt{nil}.

See also \texttt{mailbox-read} \texttt{mailbox-send} \texttt{make-mailbox} \texttt{process-wait-for-event}

\textbf{*main-process*} \hspace{2cm} \textit{Variable}

\textbf{Summary} \hspace{2cm} The process associated with the main thread.

\textbf{Package} \hspace{2cm} \texttt{mp}

\textbf{Description} \hspace{2cm} 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 \texttt{nil}.

\textbf{make-lock} \hspace{2cm} \textit{Function}

\textbf{Summary} \hspace{2cm} Makes a lock.

\textbf{Package} \hspace{2cm} \texttt{mp}
The MP Package

Signature

\texttt{make-lock \&key name important-p safep =\> lock}

Arguments

name \quad A string.

important-p \quad A generalized boolean.

safep

Values

lock \quad The lock object.

Description

\texttt{make-lock} returns a lock object. See the "Multiprocessing" chapter of the \textit{LispWorks User Guide} for a general description of locks.

\textit{name} names the lock and can be queried with \texttt{mp:lock-name}. The default value of \textit{name} is "Anon".

\textit{important-p} controls whether the lock is automatically freed when the holder process finishes. When \textit{important-p} is true, the lock is pushed onto the list \texttt{mp:*important-locks*}. Locks in this list are automatically freed when the holder process finishes. \textit{important-p} should be \texttt{nil} 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 \textit{important-p} is \texttt{nil}.

\textit{safep} controls whether the lock is safe. A safe lock gives an error if \texttt{process-unlock} 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 \textit{safep} is \texttt{t}. 
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
size An integer

Values
mailbox 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
Works with named timers created by the function make-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*))

See also
make-timer
schedule-timer
schedule-timer-milliseconds
make-timer

Function

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-relative
schedule-timer-relative-milliseconds
map-all-processes

Function

Summary
Calls the function for all processes.

Package
mp

Signature
map-all-processes function =>

Arguments
function A function taking one argument

Values
None.

Description
The function map-all-processes calls function for every process, including simple processes.

function is passed each process as its single argument.

See also
map-processes

map-all-processes-backtrace

Function

Summary
Produces a backtrace for every known process.

Package
mp

Signature
map-all-processes-backtrace &optional function

Arguments
function A function taking one argument
The map-all-processes-backtrace function calls function, which defaults to print, for every known process and each line of its backtrace.

See also  map-process-backtrace

**map-process-backtrace**  
*Function*

**Summary**  Produces a backtrace for a process

**Package**  mp

**Signature**  map-process-backtrace process function

**Arguments**  
- process  A process
- function  A function taking one argument

**Values**  None.

**Description**  The map-process-backtrace function collects a backtrace for the process specified by process, and the function function is called on each line of the backtrace in turn.
Example

```lisp
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  

Summary  Add a file descriptor to the set of interesting input file descriptors.

Package  mp

Signature  notice-fd  fd

Arguments  fd  A UNIX file descriptor

Values  None.

Description  The notice-fd function adds the given fd to the set of fds that cause LispWorks to wake up when they contain input. This function is not implemented on Microsoft Windows.

See also  unnotice-fd

process-alive-p  

Summary  Determines if a process is alive.

Package  mp

Signature  process-alive-p  process  =>  bool

Arguments  process  A process

Values  bool  A boolean

Description  The process-alive-p function returns t if process is alive, that is, if mp:process-kill has not been called on the process.

Example  (mp:process-alive-p  mp:*current-process*)
=> 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.
Returns a list of the reasons why a Lisp process has stopped. A process is inactive if it has any arrest reasons. Use of (setf mp:process-arrest-reasons) is deprecated. You should use process-stop instead. If you set the arrest reasons of the current process, this causes the current process to stop immediately, before returning from mp:process-arrest-reasons (like process-stop).

The immediate stopping behavior of (setf mp:process-arrest-reasons) is different from LispWorks 5.0 and previous versions.

See also process-run-reasons process-stop

**process-break**

**Function**

Summary Breaks a Lisp process and enters the debugger.

Package mp

Signature `process-break process =>`

Arguments `process` A process.

Values None.

Description The function `process-break` forces the process `process` to break and enter the debugger.

**process-continue**

**Function**

Summary Wakes up a process.
Package  
```
mp
```

Signature  
```
process-continue process => nil
```

Arguments  
```
process
```
A `mp:process` object.

Description  
The function `process-continue` is wakes up the process `process`, regardless of whether it is sleeping, stopped or waiting. `process-continue` returns `nil`.

### process-idle-time

**Function**

Summary  
Returns the time for which a process has been idle.

Package  
```
mp
```

Signature  
```
process-idle-time process => time
```

Arguments  
```
process
```
A process.

Values  
```
time
```
A non-negative integer.

Description  
The function `process-idle-time` returns the length of time in internal time units that `process` has been idle. If the process is running (for example the current process) then the return value is 0.

See also  
`process-run-time`

### *process-initial-bindings*

**Special Variable**

Summary  
Specifies the variables initially bound in a new process.

Package  
```
mp
```

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:

```lisp
(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:

```lisp
(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.~%")  
               (format stream "Binding 2 is unbound.~")  
               *standard-output*)  
               (sleep 1))))  
     (check-binding-2)  
     (let ((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
mp

Signature
process-interrupt process function &rest arguments =>

Arguments
process A process.
function A function to apply on resuming process.
arguments Arguments to supply to function.

Values
None.

Description
Causes the Lisp process process to apply function to arguments 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: `mp`  
Signature: `process-kill process`  
Arguments: `process` A process.  
Values: None.  
Description: The function `process-kill` kills the specified Lisp process.  
See also: `ensure-process-cleanup`  

**process-lock**  
*Function*  
Summary: Claims the lock for the current process.  
Package: `mp`  
Signature: `process-lock lock &optional whostate timeout`  
Arguments:  
- `lock` A lock object (see `make-lock`).  
- `whostate` The status of the current Lisp process, before `process-lock` returns, that is, the status while the current process is waiting to time-out. This can be seen in the Process Browser.  
- `timeout` A timeout interval, in seconds. If this is not given, `process-lock` waits until the lock can be set by the current Lisp process. A process can set a lock more than once.  
Values: `result` A boolean.
11 The MP Package

Description  
process-lock attempts to lock lock and returns t if successful, or nil if timed out. If lock is already locked and the owner of the lock is the value of *current-process*, then lock remains locked and an internal count is incremented. The Lisp process sleeps until the lock is claimed or the timeout period expires.

result is t if lock was successfully locked, and nil otherwise.

Example  
(process-lock *my-lock* "waiting to lock" 10)

See also  
create-simple-process
make-lock
process-unlock
schedule-timer
with-lock

process-mailbox

Function

Summary  
Accesses the mailbox associated with a process.

Package  
mp

Signature  
process-mailbox process => result

(setf process-mailbox) result process => result

Arguments  
process  
A process.

Values  
result  
A mailbox object, or nil.

Description  
process-mailbox is an accessor function which returns or sets the mailbox associated with process.

Example  
(setf (mp:process-mailbox mp:*current-process*)
(mp:make-mailbox))
**process-name**  
*Function*

Summary  Returns the name of a specified process.

Package  mp

Signature  \texttt{process-name} \texttt{process => name}

Arguments  \texttt{process}  A process.

Values  \texttt{name}  The name of the process specified by \texttt{process}.

Description  Returns the name of the specified Lisp process.

**process-p**  
*Function*

Summary  A predicate to identify non-simple processes

Package  mp

Signature  \texttt{process-p} \texttt{object => bool}

Arguments  \texttt{object}  Any object

Values  \texttt{bool}  A generalized boolean.

Description  The \texttt{process-p} function returns \texttt{t} if \texttt{object} is a non-simple process, and \texttt{nil} otherwise.

**process-plist**  
*Function*

Summary  Returns the plist associated with a process.

Package  mp
Signature  
\texttt{process-plist} \texttt{process} \Rightarrow \texttt{plist}

Arguments  
\texttt{process} \hspace{1cm} \text{A process}

Values  
\texttt{plist} \hspace{1cm} \text{A plist}

Description  
The \texttt{process-plist} function returns the plist associated with \texttt{process}.

Example  
\begin{verbatim}
(setf (getf (mp:process-plist mp:*current-process*)
    'foo) 'foo-value)
\Rightarrow \texttt{FOO-VALUE}
\end{verbatim}

\begin{verbatim}
(getf (mp:process-plist mp:*current-process*) 'foo)
\Rightarrow \texttt{FOO-VALUE}
\end{verbatim}

\textbf{process-priority} \hspace{1cm} \textit{Function}

Summary  
Returns the numerical priority of the Lisp process.

Package  
\texttt{mp}

Signature  
\texttt{process-priority} \texttt{process} \Rightarrow \texttt{priority}

Arguments  
\texttt{process} \hspace{1cm} \text{A process.}

Values  
\texttt{priority} \hspace{1cm} \text{A fixnum, the priority of \texttt{process}.}

Description  
Returns the numerical priority of the Lisp process. This can be modified by calling \texttt{mp:change-process-priority}.

Example  
\begin{verbatim}
CL-USER 17 > (mp:process-priority mp:*current-process*)
600000
\end{verbatim}

See also  
\texttt{change-process-priority}
**process-reset**

*Function*

**Summary**
Resets a process by discarding its current state.

**Package**
`mp`

**Signature**
`process-reset process =>`

**Arguments**
process A process.

**Values**
None.

**Description**
`process-reset` interrupts the execution of process and “throws away” its current state. Upon resuming execution, the process calls its function with its initial argument and priority.

**process-run-function**

*Function*

**Summary**
Create a new process, passing it a function to run.

**Package**
`mp`

**Signature**
`process-run-function name keywords function &rest arguments => process`

**Arguments**
name A name for the new process.
keywords Keywords specifying properties of the new process.
function A function to apply.
arguments Arguments to pass to function.

**Values**
process The newly created process.
Description

This function creates a new Lisp process with name `name`. Other properties of `process` may be specified in keyword/value pairs in `keywords`:

- **: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 `process`.

The new process is preset to apply `function` to `arguments` and runs in parallel, while `process-run-function` returns immediately.

Example

```lisp
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*)))
#<MP:PROCESS Name "My process" Priority 850000 State "Running">

0
1
2
42
CL-USER 255 >
```

See also

- `create-simple-process`
- `*default-process-priority*`
process-run-reasons

Function

Summary Returns the reasons that a specified process is running.

Package mp

Signature process-run-reasons process => reasons
(setf process-run-reasons) process reasons => reasons

Arguments process A process.

Values reasons A list of run reasons.

Description The function process-run-reasons returns a list of reasons for the specified Lisp process running. These can be changed using setf.

A process is only active if it has at least one run reason and no arrest reasons.

See also process-arrest-reasons
process-run-function
process-whostate

process-run-time

Function

Summary Returns the current run time for a process.

Package mp

Signature process-run-time process => time

Arguments process A process.

Values time A positive integer or nil.
Description

The function `process-run-time` returns the current run time for `process` in internal time units. If the value cannot be determined (currently this is only on FreeBSD), then the return value is `nil`.

**Note:** The value returned by `get-internal-run-time` is similar, but on some operating systems it is the total time for all Lisp processes in the image.

See also

`process-idle-time`

**process-send**

*Function*

**Summary**

Sends an object to the mailbox of a given process.

**Package**

`mp`

**Signature**

`process-send process object &key change-priority =>`

**Arguments**

- `process` A process
- `object` An object
- `change-priority` A fixnum, `nil`, `t`, or `:default`

**Values**

None.

**Description**

The `process-send` function queues `object` in the mailbox of the given process.

`object` should be a list of the form `(function . arguments)`. This causes `function` to be applied to `arguments` in the process `process`. (The results of `function` are discarded.)

If `change-priority`, which has a default value of `:default`, is non-`nil`, it controls how the priority of that process is calculated as follows:
• **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
- `process-arrest-reasons`
- `process-stopped-p`
- `process-unstop`

### 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 unstop the process).

See also
- `process-stop`
- `process-unstop`

### process-unlock Function

**Summary**
Relinquishes a lock held by the current process.
Package

mp

Signature

process-unlock  lock  &optional  errorp  =>  result

Arguments

lock  The lock to be relinquished.

errorp  When this is t, an error is signalled if *current-process* is not the owner of the lock. The default is t.

Values

result  A boolean.

Description

Attempts to release a lock. If the lock is owned by *current-process*, process-unlock decrements an internal count. If this lock count is then zero, the lock is released. Note that process-unlock relates only on Lisp processes.

result is t if the lock was released, and nil otherwise.

See also

create-simple-process
make-lock
process-lock
schedule-timer
with-lock

process-unstop

Function

Summary

Unstops a process.

Package

mp

Signature

process-unstop  process  =>  result

Arguments

process  A mp:process object.

Values

result  A boolean.
The function `process-unstop` unstops the process `process` if it is stopped.

`process` must be a full process (that is, not one created by `create-simple-process`).

If `process` was stopped (by `process-stop`), it is unstopped and resumes execution.

`result` is `t` if `process` was stopped, and `nil` otherwise.

There is a discussion of a typical use of `process-unstop` in the section "Stopping and unstopping processes" in the *Lisp-Works User Guide*.

See also `process-stop` `process-stopped-p`
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**

mp

**Signature**

process-wait-for-event &key wait-reason wait-function process-other-messages-p no-hang-p stop-at-user-operation-p => event

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wait-reason</td>
<td>A string or nil.</td>
</tr>
<tr>
<td>wait-function</td>
<td>A function designator.</td>
</tr>
<tr>
<td>process-other-messages-p</td>
<td>A generalized boolean.</td>
</tr>
<tr>
<td>no-hang-p</td>
<td>A generalized boolean.</td>
</tr>
<tr>
<td>stop-at-user-operation-p</td>
<td>A generalized boolean.</td>
</tr>
</tbody>
</table>

**Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>result</td>
<td>An event or nil.</td>
</tr>
</tbody>
</table>

**Description**

The function process-wait-for-event calls mailbox-wait-for-event on the mailbox of the current process, after ensuring that the current process has a mailbox.

The arguments and value are interpreted as for mailbox-wait-for-event.
See also  

**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  
`process-wait-with-timeout`  `wait-reason`  `timeout`  `&optional`  `wait-function`  `&rest`  `wait-arguments`  `=>`  `bool`

Arguments  
`wait-reason`  A string describing the reason that the process is waiting.
`timeout`  A timeout, in seconds.
`wait-function`  A function to test.
`wait-arguments`  The arguments to apply to `wait-function`.

Values  
`bool`  A boolean.

Description  
This function uses `process-wait` to suspend the current Lisp process until the predicate `wait-function` applied to `wait-arguments` returns `t`, or until `timeout` seconds have passed.

`bool` is `nil` if the timeout occurred before `wait-function` returned true. `bool` is true otherwise.

See also  
`process-wait`

---

`process-whostate`  

Function

Summary  
Returns the state of a process.

Signature  
`process-whostate`  `process`  `=>`  `result`

Package  
`mp`

Arguments  
`process`  A process.

Values  
`reason`  A string.

Description  
The function `process-whostate` returns a string describing the state of the process.

Depending on the state of `process`, `reason` can be:
• "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

**ps**

**Function**

**Summary**
Prints the processes in the system

**Package**
mp

**Signature**
ps =>

**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 ps.)
schedule-timer

**Summary**
Schedules a timer to expire at a given time after the start of the program.

**Signature**
```
schedule-timer timer absolute-expiration-time &optional repeat-time => timer
```

**Package**
`mp`

**Arguments**
- `timer` A timer
- `absolute-expiration-time` A non-negative real
- `repeat-time` A non-negative real

**Values**
- `timer` A timer

**Description**
The `schedule-timer` function schedules a timer to expire at a given time after the start of the program. The `timer` argument is a timer, returned by `make-timer` or `make-named-timer`. The `absolute-expiration-time` argument is a non-negative real number of seconds since the start of the program at which the timer is to expire. If `repeat-time` 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.

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 `absolute-expiration-time` argument. If that argument is `nil`, the timer is not rescheduled, but the repeat interval is set to the interval specified by the `repeat-time` argument.

The function `schedule-timer-relative` schedules a timer to expire at a time relative to the call to that function.
Example

The following example schedules a timer to expire 15 minutes after the start of the program and every 5 minutes thereafter.

```lisp
(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

**schedule-timer-milliseconds**

Function

Summary

Schedules a timer to expire after a given amount of time.

Signature

```
schedule-timer-milliseconds timer absolute-expiration-time &optional repeat-time => timer
```

Package

`mp`

Arguments

- `timer` A timer
- `absolute-expiration-time` A non-negative real
- `repeat-time` A non-negative real

Values

`timer` A timer
The `schedule-timer-milliseconds` function schedules a timer to expire at a given time after the start of the program. The `timer` argument is a timer returned by `make-timer` or `make-named-timer`. The `absolute-expiration-time` argument is a non-negative real number of milliseconds since the start of the program at which the timer is to expire. If `repeat-time` 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.

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 `absolute-expiration-time` argument. If that argument is `nil`, the timer is not rescheduled, but the repeat interval is set to the interval specified by the `repeat-time` argument.

The function `schedule-timer-relative-milliseconds` schedules a timer to expire at a time relative to the call to that function.

The following example schedules a timer to expire 15 minutes after the start of the program and every 5 minutes thereafter.

```lisp
(setq timer
    (mp:make-timer 'print 10 *standard-output*))
#<Time Event : PRINT>
(mp:schedule-timer-milliseconds timer 900000 300000)
#<Time Event : PRINT>
```

See also `make-named-timer` `make-timer` `schedule-timer` `schedule-timer-relative` `schedule-timer-relative-milliseconds` `timer-expired-p` `timer-name` `unschedule-timer`
Schedule a timer to expire at a given time after this function is called.

Signature

\[
schedule-timer-relative \quad \text{timer} \quad \text{relative-expiration-time} \\
&optional \quad \text{repeat-time} \Rightarrow \text{timer}
\]

Package

mp

Arguments

\begin{itemize}
  \item \textit{timer} \quad A timer
  \item \textit{relative-expiration-time} \quad A non-negative real
  \item \textit{repeat-time} \quad A non-negative real
\end{itemize}

Values

\begin{itemize}
  \item \textit{timer} \quad A timer
\end{itemize}

Description

The \texttt{schedule-timer-relative} function schedules a timer to expire at a given time after the call to the function. The \textit{timer} argument is a timer returned by \texttt{make-timer} or \texttt{make-named-timer}. The \textit{relative-expiration-time} argument is a non-negative real number of seconds after the call to the function at which the timer is to expire. If \textit{repeat-time} 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.

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 \textit{relative-expiration-time} argument. If that argument is \texttt{nil}, the timer is not rescheduled, but the repeat interval is set to the interval specified by the \textit{repeat-time} argument.

The function \texttt{schedule-timer} schedules a timer to expire at a time relative to the start of the program.
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
The `schedule-timer-relative-milliseconds` function schedules a timer to expire at a given time after the call to the function. The `timer` argument is a timer returned by `make-timer` or `make-named-timer`. The `relative-expiration-time` argument is a non-negative real number of milliseconds after the call to the function at which the timer is to expire. If `repeat-time` 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.

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 `relative-expiration-time` argument. If that argument is `nil`, the timer is not rescheduled, but the repeat interval is set to the interval specified by the `repeat-time` argument.

The function `schedule-timer-milliseconds` schedules a timer to expire at a time relative to the start of the program.

Example

The following example schedules a timer to expire 5 seconds after the call to `schedule-timer-relative-milliseconds` and every 5 seconds thereafter.

```lisp
(setq timer
  (mp:make-timer 'print 10 *standard-output*))

#<Time Event : PRINT>

(mp:schedule-timer-relative-milliseconds timer 5000 5000)

#<Time Event : PRINT>
```

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**  
`simple-process-p object => bool`

**Arguments**  
`object`  
An object

**Values**  
`bool`  
A generalized boolean

**Description**  
The `simple-process-p` function returns `t` if `object` is a simple process and `nil` otherwise.

**See also**  
`create-simple-process`

---

**symeval-in-process**  
*Function*

**Summary**  
Reads the value of symbol which is dynamically bound in a given process.

**Package**  
mp

**Signature**  
`symeval-in-process symbol process => value, flag`

`(setf symeval-in-process) value symbol process => value`

**Arguments**  
`symbol`  
A symbol

`process`  
A process

**Values**  
`value`  
A Lisp object

`flag`  
One of `t`, `nil` or the keyword `:unbound`
The function `symeval-in-process` reads the value of the symbol `symbol` in the process `process` if it is bound dynamically. The global value of `symbol` is never returned.

If `symbol` is not bound in `process`, then `value` and `flag` are both `nil`. If `symbol` is bound in `process` but `makunbound` has been called within the dynamic scope of the binding, `value` is `nil` and `flag` is `:unbound`. Otherwise, `value` is the value of `symbol` and `flag` is `t`.

In addition, the form

```lisp
(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 `setq` form returns `value` as specified by Common Lisp.

### timer-expired-p

**Function**

**Summary**

Returns `t` if a given timer has expired or is about to expire.

**Signature**

`timer-expired-p timer &optional delta => bool`

**Package**

`mp`

**Arguments**

- `timer` A timer
- `delta` A non-negative real

**Values**

- `bool` A boolean

**Description**

The `timer-expired-p` function returns `t` if the specified timer is not scheduled to expire or is scheduled to expire within the number of seconds specified by the `delta` argument after the call to `timer-expired-p`. Otherwise, the function returns `nil`. 
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

```lisp
(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
The `timer-name` function returns the name of the specified timer. The `timer` argument is a timer returned by `make-timer` or `make-named-timer`. If the timer has no name, `timer-name` returns `nil`.

The name of a timer created by either `make-timer` or `make-named-timer` can be set by means of the following syntax:

```lisp
(setf (mp:timer-name timer) name)
```

**Example**

```lisp
(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
```

**See also**

- `make-named-timer`
- `make-timer`
- `schedule-timer`
- `schedule-timer-milliseconds`
- `schedule-timer-relative`
- `timer-expired-p`
- `unschedule-timer`

**unnotice-fd**

**Function**

**Summary**

Removes a file descriptor from the set of interesting input file descriptors.

**Package**

`mp`
Signature: \texttt{unnotice-fd \textit{fd}}

Arguments: \textit{fd} \quad A file descriptor

Values: None.

Description: The \texttt{unnotice-fd} function removes \textit{fd} from the set of fds that cause LispWorks to wake up when they contain input.

This function is not implemented on Microsoft Windows.

See also: \texttt{notice-fd}

\textbf{unschedule-timer} \hspace{1cm} \textit{Function}

Summary: Unschedules a scheduled timer

Signature: \texttt{unschedule-timer \textit{timer} \Rightarrow \textit{result}}

Package: \texttt{mp}

Arguments: \textit{timer} \quad A timer

Values: \textit{result} \quad A timer or \texttt{nil}

Description: If the specified timer has been scheduled to expire at a time after the call to \texttt{unschedule-timer}, this function unschedules the timer and returns the timer. Otherwise, the function returns \texttt{nil}.

The argument is a timer, returned by \texttt{make-timer} or \texttt{make-named-timer}.

Example:
\begin{verbatim}
(setq timer
  (mp:make-timer 'print 10 *standard-output*))

<Time Event : PRINT>

(mp:schedule-timer-relative timer 60)
\end{verbatim}
11 The MP Package

### wait-processing-events

**Function**

**Summary**
Waits processing events.

**Signature**
```lisp
wait-processing-events timeout &key wait-reason wait-function
  wait-args => result
```

**Package**
`mp`

**Arguments**
- `timeout` A number.
- `wait-reason` A string.
- `wait-function` A function designator.
- `wait-args` A list.

**Values**
- `result` `t` or `nil`

**Description**
The function `wait-processing-events` does not return until one of two conditions is met:

- `timeout` seconds have passed. In this case, `result` 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
- `process-send`
- `process-whostate`

**with-lock**

*Macro*

**Summary**
Executes a body of code while holding a lock.

**Package**
`mp`

**Signature**
`with-lock (lock &rest lock-args) &body body => result`

**Arguments**

- `lock` The lock.
- `lock-args` These are the optional arguments used by `process-lock: whostate` (the status of the process while the lock is locked, as seen in the Process Browser) and `timeout` (a timeout period, in seconds).
The forms to execute.

Values

result The result of executing body.

Description

with-lock executes body while holding the lock, and unlocks the lock when body exits. This is the recommended way of using locks. The value of body is returned normally. body is not executed if the lock could not be claimed, in which case, with-lock returns nil.

See also

create-simple-process
make-lock
process-lock
process-unlock
schedule-timer

---

**without-interrupts**

*Macro*

Summary

Causes any interrupts that occur during the execution of a body of code to be queued.

Package

mp

Signature

without-interrupts &rest body => result

Arguments

body The forms to execute while interrupts are queued.

Values

result The result of executing body.

Description

While body is executing, all interrupts (for example, preemption, keyboard break etc.) are queued. They are executed when body 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.

```lisp
(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`
## The MP Package

<table>
<thead>
<tr>
<th>Signature</th>
<th>yield</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arguments</td>
<td>None.</td>
</tr>
<tr>
<td>Values</td>
<td>None.</td>
</tr>
</tbody>
</table>

**Description**

Normally code compiled at safety 0 cannot be preempted because the necessary checks are omitted. This can be overcome by calling `yield` at regular intervals. Usually there is no need to call this if you use functions from the `common-lisp` 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, `yield` can be useful. Note that `process-allow-scheduling` also allows preemption, but also checks the wait functions of other processes.

**See also**

`process-allow-scheduling`
This chapter describes symbols available in the PARSERGEN package, the LispWorks parser generator. You should use this chapter in conjunction with the relevant chapter in the LispWorks User Guide.

**defparser**

**Summary**

Creates a parsing function of the given name for the grammar defined.

**Package**

PARSERGEN

**Signature**

```
defparser name (rule)* => parsing-function
rule ::= normal-rule | error-rule
normal-rule ::= ((non-terminal (grammar-symbol)* (form)*)
error-rule ::= ((non-terminal :error) (form)*)
```

**Arguments**

- `name` 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**

- `parsing-function` 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> (lexer &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.
This chapter describes the symbols available in the SERIAL-PORT package.

The Serial Port functionality is loaded into LispWorks by evaluating

```
(requires "serial-port")
```

Note: this chapter applies only to LispWorks for Windows, and not the UNIX, Linux or Mac OS X platforms.

**open-serial-port**

*Function*

**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 drt rts read-interval-timeout
read-total-base-timeout read-total-byte-timeout write-total-base-timeout
write-total-byte-timeout => serial-port)
```

**Arguments**

`name` A string naming a serial port.
The SERIAL-PORT Package

See in the Description below for details of the remaining arguments.

Values

- **serial-port**: A serial-port object.

Description

The function `open-serial-port` attempts to open the serial port `name` and return a serial-port object. `name` is passed directly to `Createfile()`. For ports COMn where n > 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](http://support.microsoft.com/kb/115831).

If any of `baud-rate`, `data-bits`, `stop-bits` and `parity` are supplied then the corresponding serial port settings are changed. The values of `baud-rate` and `data-bits` should each be an appropriate integer. The value of `stop-bits` should be 1, 1.5 or 2. The value of `parity` should be one of the keywords `:even`, `:mark`, `:none`, `:odd` or `:space`.

The arguments `cts-flow-p` and `dsr-flow-p` control whether write operations respond to CTS and DSR flow control. A non-nil value means that the corresponding flow control is used.

The arguments `dtr` and `rts` control whether read operations generate DTR or RTS flow control. If the value is `:handshake` then the corresponding flow control signal is generated automatically. If the value is `nil` or `t` then the initial state of the flow control signal is set and automatic flow control is not used. See `set-serial-port-state` for manual flow control.

The argument `read-interval-timeout` can be used to control the maximum time to wait between each input character. The value `:none` means that reading will not wait for characters at all, only returning whatever is already in the input buffer.

The arguments `read-total-base-timeout` and `read-total-byte-timeout` can be used to control the maximum time to wait for a sequence of characters. The arguments `write-total-base-timeout` and `write-total-byte-timeout` 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:

$$\text{base_timeout} + \text{nchars} \times \text{byte_timeout}$$

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.
### The SERIAL-PORT Package

This chapter applies only to LispWorks for Windows

<table>
<thead>
<tr>
<th>Package</th>
<th>serial-port</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signature</td>
<td><code>get-serial-port-state serial-port keys =&gt; state</code></td>
</tr>
</tbody>
</table>
| Arguments     | serial-port A serial-port object.  
               | keys A list of keywords.  
               | state A list. |
| Values        | state A list. |
| Description   | The function `get-serial-port-state` queries various aspects of the state of the serial port associated with `serial-port`.  
               | The argument `keys` should be a list of one or more of the keywords `:dsr` and `:cts`. These cause `get-serial-port-state` to check the DSR and CTS lines respectively.  
               | The result `state` is a list giving the state of each line in the same order as they appear in the argument `keys`. |

#### serial-port

**Class**

**Summary** The class of objects representing serial ports.

**Package** serial-port

**Description** The class `serial-port` is the class of objects representing serial ports. These are constructed by `open-serial-port` - do not create them directly.

**See also** `open-serial-port`

#### read-serial-port-char

**Function**

**Summary** Reads a character from a serial port.

**Package** serial-port
This chapter applies only to LispWorks for Windows

**read-serial-port-char**

*Signature*

```
read-serial-port-char serial-port &optional timeout-error-p
timeout-char => char
```

*Arguments*

- `serial-port`: A `serial-port` object.
- `timeout-error-p`: A boolean.
- `timeout-char`: A character.

*Values*

- `char`: A character.

*Description*

The function `read-serial-port-char` reads and returns a character from the serial port associated with `serial-port`. 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 `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 `timeout-char` is returned. The default value of `timeout-error-p` is `t`.

*See also*

`read-serial-port-string`

**read-serial-port-string**

*Function*

*Summary*

Reads a string from a serial port.

*Package*

`serial-port`

*Signature*

```
read-serial-port-string string serial-port &optional timeout-error-p &key start end => nread
```

*Arguments*

- `string`: A string.
- `serial-port`: A `serial-port` object.
- `timeout-error-p`: A boolean.
- `start`, `end`: Bounding index designators for `string`. 
Values  

\( nread \)  

An integer.

Description  
The function \texttt{read-serial-port-string} reads characters from the serial port associated with \texttt{serial-port} and places them in \texttt{string}, bounded by \texttt{start} and \texttt{end}.

The default values of \texttt{start} and \texttt{end} are 0 and \texttt{nil} (interpreted as the length of \texttt{string}) respectively. The number of characters requested is the difference between \texttt{end} and \texttt{start}.

If the number of characters actually read, \( nread \), is less than the number requested, then if \texttt{timeout-error-p} is non-\texttt{nil} an error of type \texttt{serial-port-timeout} is signalled.

If \( nread \) is the number of characters requested, or if \texttt{timeout-error-p} is \texttt{nil}, \( nread \) is returned.

The default value of \texttt{timeout-error-p} is \texttt{t}.

See also  
\texttt{read-serial-port-char}

\texttt{serial-port-input-available-p}  

Function  

Summary  
Checks whether a character is available on a serial port.

Package  
\texttt{serial-port}

Signature  
\texttt{serial-port-input-available-p serial-port \( \Rightarrow \) result}

Arguments  
\texttt{serial-port}  
A \texttt{serial-port} object.

Values  
\texttt{result}  
A boolean.

Description  
The function \texttt{serial-port-input-available-p} checks the serial port associated with \texttt{serial-port} to see if a character is available. \texttt{result} is \texttt{t} if input is available, and \texttt{nil} otherwise.
This chapter applies only to LispWorks for Windows

**set-serial-port-state**

*Function*

**Summary**
Changes various aspects of the state of a serial port.

**Package**
serial-port

**Signature**

\[
\text{set-serial-port-state serial-port &key dtr rts break}
\]

**Arguments**

- `serial-port` A `serial-port` object.
- `dtr` A boolean.
- `rts` A boolean.
- `break` A boolean.

**Description**
The function `set-serial-port-state` changes various aspects of the state of the serial port associated with `serial-port`.

The argument `dtr`, if supplied, controls the DTR line. A true value means set and `nil` means clear. If `dtr` is not supplied, the state is unchanged.

The argument `rts` controls the RTS line in the same way.

The argument `break` controls the break state of the data line in the same way.

**wait-serial-port-state**

*Function*

**Summary**
Waits for some aspect of the state of a serial port to change.

**Package**
serial-port

**Signature**

\[
\text{wait-serial-port-state serial-port keys &key timeout => result}
\]

**Arguments**

- `serial-port` A `serial-port` object.
- `keys` A list of keywords.
- `timeout` A number.
The function `wait-serial-port-state` waits for some state in the serial port associated with `serial-port` to change.

The argument `keys` should be a list of one or more of the keywords `:cts`, `:dsr`, `:err`, `:ring`, `:rlsd` and `:break`.

`result` is a list giving the keys for which the state has changed.

If `timeout` is non-`nil` then the function will return `nil` after that many seconds even if the state has not changed.

---

**write-serial-port-char**

Function

**Summary**

Writes a character to a serial port.

**Package**

`serial-port`

**Signature**

`write-serial-port-char char serial-port &optional timeout-error-p => char`

**Arguments**

- `char` A character.
- `serial-port` A `serial-port` object.
- `timeout-error-p` A boolean.

**Values**

- `char` A character.

**Description**

The function `write-serial-port-char` writes the character `char` to the serial port associated with `serial-port`, and returns `char`.

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 `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-string

Function

Summary

Writes a string to a serial port.

Package

`serial-port`

Signature

```
write-serial-port-string string serial-port &optional timeout-error-p &key start end => nwritten
```

Arguments

- `string`: A string.
- `serial-port`: A `serial-port` object.
- `timeout-error-p`: A boolean.
- `start, end`: Bounding index designators for `string`.

Values

- `result`: The string `string` or `nil`.

Description

The function `write-serial-port-string` writes characters from the subsequence of `string` bounded by `start` and `end` to the serial port associated with `serial-port`.

The default values of `start` and `end` are 0 and `nil` (interpreted as the length of `string`) respectively.

If the characters are successfully written then `string` 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`
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.

```
add-sql-stream
```

Function

Summary
Adds a stream to the broadcast list for SQL commands or results traffic.

Package
`sql`

Signature
```
add-sql-stream stream &key type database => added-stream
```

Arguments
- `stream`: A stream, or `t`.
- `type`: A keyword.
The SQL Package

This chapter applies to the Enterprise Edition only

---

**database**  A database.

**Values**

*added-stream*  The argument stream.

**Description**

The `add-sql-stream` function adds the stream `stream` to the list of streams which receive SQL commands traffic or results traffic.

To add `*standard-output*` to the list, pass `stream t`.

The argument `type` is one of `:commands`, `:results` or `:both`, and determines whether a stream for commands traffic, results traffic, or both is added.

The argument `type` has a default value of `:commands`. The `database` is the value of `*default-database*` by default.

See also

* `*default-database*`
* `delete-sql-stream`
* `list-sql-streams`
* `sql-recording-p`
* `sql-stream`
* `start-sql-recording`
* `stop-sql-recording`

---

**attribute-type**

**Function**

*Summary*  Returns the type of an attribute.

**Package**  sql

**Signature**  

```
attribute-type attribute table &key database owner => datatype
```

**Arguments**

* `table`  A table.
* `attribute`  An attribute from `table`.
* `database`  A database.
This chapter applies to the Enterprise Edition only

owner    nil, :all or a string.

Values  datatype  A keyword or list denoting a vendor-specific type.

Description  The function attribute-type returns the type of the attribute specified by attribute in the table given by table. The database, in which table is found, has a default value of *default-database*.

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.

datatype demotes a vendor-specific type. Examples in a MS Access database are :integer, :longchar and :datetime. When datatype is a list, the second element is the length of the type, for example (:varchar 255).

Example  To print the type of every attribute in the database, do

  (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)))))

See also  *default-database*

  list-attribute-types

  list-attributes
cache-table-queries

Summary
Controls the caching of attribute type information.

Package
sql

Signature
cache-table-queries table &key database action

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table</td>
<td>A string naming a table, :default or t.</td>
</tr>
<tr>
<td>database</td>
<td>A database.</td>
</tr>
<tr>
<td>action</td>
<td>t, nil or :flush.</td>
</tr>
</tbody>
</table>

Description
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.

If table is a string, it is the name of the table for which caching is to be altered. If table is t, then the action applies to all tables. If table is :default, then the default caching action is set for those tables which do not have an explicit setting.

database specifies the database connection, its default value is the value of *default-database*.

action 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 table is :default, the setting applies to all tables which do not have an explicit setup.

The value :flush means remove any existing cache for table in database, but continue to cache.

cache-table-queries should be called with action :flush when the attribute specifications in table have changed.

See also
*cache-table-queries-default*
*default-database*
This chapter applies to the Enterprise Edition only

**cache-table-queries-default**  
Variable

<table>
<thead>
<tr>
<th>Package</th>
<th>sql</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial Value</td>
<td>nil</td>
</tr>
<tr>
<td>Description</td>
<td>The variable <em>cache-table-queries-default</em> provides the default attribute type caching behaviour. It allowed values are as described for the action argument of cache-table-queries.</td>
</tr>
</tbody>
</table>

See also  
cache-table-queries

commit  
Function

<table>
<thead>
<tr>
<th>Summary</th>
<th>Commits changes made to a database.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package</td>
<td>sql</td>
</tr>
<tr>
<td>Signature</td>
<td>commit &amp;key database =&gt; nil</td>
</tr>
<tr>
<td>Arguments</td>
<td>database A database.</td>
</tr>
<tr>
<td>Values</td>
<td>nil</td>
</tr>
<tr>
<td>Description</td>
<td>The commit function commits changes made to the database specified by database, which is <em>default-database</em> by default.</td>
</tr>
</tbody>
</table>

Example | This example changes records in a database, and uses commit 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

**connect**

*Function*

**Summary**
Opens a connection to a database.

**Package**
sql

**Signature**
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

**Arguments**
- *connection-spec*  The connection specifications.
- *if-exists*  A keyword.
- *database-type*  A database type.
- *interface*  A displayed CAPI element, or nil.
- *name*  A Lisp object.
- *encoding*  A keyword naming an encoding.
- *signal-rollback-errors*  nil, the keyword :default, or a function designator.
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 'g' 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 nil), 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 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 nil (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:
If `encoding` is `nil` or `:default` then the encoding is chosen according to the default character set of the connection (if available) and if that fails the encoding `:utf-8` is used. The other recognised values of `encoding` are `:unicode`, `:utf-8`, `:ascii`, `:latin-1`, `:euc` and `:sjis`. `:unicode` uses `:utf-8` internally.

The only recognised values of `encoding` are `nil` and `:unicode`.

`encoding` is ignored.

The only recognised values of `encoding` are `nil` and `:unicode`.

`encoding` is ignored.

`encoding` is ignored.

`encoding` 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-type` 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:

- `nil` Ignore the error.
- `:default` If `default-table-type` is `:support-transactions`, "innodb" or "bdb", then rollback errors are signalled. Otherwise rollback errors are not signalled.

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:

```sql
"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:

```sql
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.
Compatibity Note

LispWorks 4.4 (and previous versions) use connection-spec passed to connect as the database name. connect checks if a connection with this name already exists (according to the value of if-exists). find-database can be used to find a database using this name.

LispWorks 5.0 (and later versions) does not use connection-spec as the name. Instead, by default it generates a name from the connection-spec. The name is intended to be unique (by including a counter). Thus normally connect will not find an existing connection even if it is called again with identical value of connection-spec.

Example

The following example connects LispWorks to the info database.

```
(connect "info")
```

The next example connects to the ODBC database personnel using the username "admin" and the password "secret".

```
(connect "personnel/admin/secret" :database-type :odbc)
```

The next example opens a connection to MySQL which treats quotes as in ANSI but does not set other ANSI features:

```
(sql:connect "me/mypassword/mydb"
             :sql-mode "ANSI_QUOTES")
```

See also

*default-database*
*default-database-type*
connected-databases
*connect-if-exists*
database-name
disconnect
find-database
reconnect
status
This chapter applies to the Enterprise Edition only

**connect-if-exists**

*Variable*

**Summary**
The default value for the *if-exists* keyword of the `connect` function.

**Package**
sql

**Initial Value**
: error

**Description**
The variable *connect-if-exists* is the default value for the *if-exists* keyword of the `connect` function. It can take the following values:

- **:new**
  Instructs `connect` to make a new connection even if connections to the same database already exist.

- **:warn-new**
  Instructs `connect` to make a new connection but warn about existing connections.

- **:error**
  Instructs `connect` to make a new connection but signal an error for existing connections.

- **:warn-old**
  Instructs `connect` to select an old connection if one exists (and warns) or make a new one.

- **:old**
  Instructs `connect` to select an old connection if one exists or make a new one.

**See also**
connect

**connected-databases**

*Function*

**Summary**
Returns a list of connected databases.

**Package**
sql

**Signature**
connected-databases => database-list
Arguments
None.

Values
\texttt{database-list} \quad A list of connected databases.

Description
The function \texttt{connected-databases} returns a list of the databases LispWorks is connected to.

See also
\texttt{connect} \quad \texttt{disconnect} \quad \texttt{status} \quad \texttt{find-database} \quad \texttt{database-name}

\textbf{create-index} \quad \textit{Function}

Summary
Creates an index for a table.

Package
\texttt{sql}

Signature
\texttt{create-index name &key on unique attributes database =>}

Arguments
\texttt{name} \quad The name of the index.
\texttt{on} \quad The name of a table.
\texttt{unique} \quad A boolean.
\texttt{attributes} \quad A list of attributes.
\texttt{database} \quad A database.

Values
None.

Description
The function \texttt{create-index} creates an index called \texttt{name} on the table specified by \texttt{on}. The attributes of the table to index are given by \texttt{attributes}. Setting \texttt{unique} to \texttt{t} includes \texttt{UNIQUE} 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

```lisp
(create-index [manager]
  :on [emp] :unique t :attributes '([ename] [sal]))
```

See also

*default-database*
drop-index
create-table

**create-table**

*Function*

**Summary**

Creates a table.

**Package**

sql

**Signature**

`create-table name description &key database type extra-options`

**Arguments**

- `name` The name of the table.
- `description` The table properties.
- `database` A database.
- `type` A string or the keyword :support-transactions, or nil.
- `extra-options` A string or nil.

**Values**

None.

**Description**

The function `create-table` creates a table called `name` and defines its columns and other properties with `description`. The argument `description` is a list containing lists of attribute-name and type information pairs.

The default value of `database` is *default-database*. 
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:

```sql
create table MyTable (column-specs) TYPE = type
```

except that if type is :support-transactions then create-table 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:

```sql
(create-table [manager]
  '(((id] (char 10) not-null)
   ([salary] (number 8 2))))
```

is equivalent to the following SQL:

```sql
CREATE TABLE MANAGER
  (ID CHAR(10) NOT NULL, SALARY NUMBER(8,2))
```

See also  connect

*default-database*

drop-table

create-view

Summary  Creates a view using a specified query.
This chapter applies to the Enterprise Edition only

Package sql

Signature create-view name &key as column-list with-check-option database =>

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The view to be created.</td>
</tr>
<tr>
<td>as</td>
<td>An SQL query statement.</td>
</tr>
<tr>
<td>column-list</td>
<td>A list.</td>
</tr>
<tr>
<td>with-check-option</td>
<td>A boolean.</td>
</tr>
<tr>
<td>database</td>
<td>A database.</td>
</tr>
</tbody>
</table>

Values None.

Description The create-view function creates a view called name using the as query and the optional column-list and with-check-option. The column-list argument is a list of columns to add to the view. The with-check-option adds WITH CHECK OPTION to the resulting SQL.

The default value of with-check-option is nil. The default value of database is *default-database*.

Example This example creates the view manager with the records in the employee table whose department is 50.


See also create-index
create-table
*default-database*
drop-view
**create-view-from-class**

*Function*

**Summary**
Creates a view in a database based on a class that defines the view.

**Package**
sql

**Signature**
create-view-from-class class &key database =>

**Arguments**
class A class.
database A database.

**Values**
None.

**Description**
The function create-view-from-class creates a view in database based on class which defines the view. The argument database has a default value of *default-database*.

**See also**
*default-database*
drop-view-from-class
create-view

**database-name**

*Function*

**Summary**
Returns the name of a database.

**Package**
sql

**Signature**
database-name database => connection

**Arguments**
database A database.

**Values**
connection A string.

**Description**
The function database-name returns the name of the database specified by database.


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 sql

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 sql

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 \texttt{initialize-database-type}

\textbf{*default-update-objects-max-len*} \textit{Variable}

\textbf{Summary} \hspace{1cm} The default maximum number of objects supplying data for a query when updating remote joins.

\textbf{Package} \hspace{1cm} \texttt{sql}

\textbf{Initial Value} \hspace{1cm} \texttt{nil}

\textbf{Description} \hspace{1cm} The variable \texttt{*default-update-objects-max-len*} provides the default value of the \texttt{max-len} argument in the function \texttt{update-objects-joins}.

See also \texttt{update-objects-joins}

\textbf{def-view-class} \textit{Macro}

\textbf{Summary} \hspace{1cm} Extends the syntax of \texttt{defclass} to allow specified slots to be mapped onto the attributes of database views.

\textbf{Package} \hspace{1cm} \texttt{sql}

\textbf{Signature} \hspace{1cm} \texttt{def-view-class name superclasses slots &rest class-options => class}

\textbf{Arguments} \hspace{1cm} \begin{itemize}
  \item \texttt{name} \hspace{1cm} A class name.
  \item \texttt{superclasses} \hspace{1cm} The superclasses of the class to be created.
  \item \texttt{slots} \hspace{1cm} The slot definitions of the new class.
  \item \texttt{class-options} \hspace{1cm} The class options of the new class.
\end{itemize}

\textbf{Values} \hspace{1cm} \texttt{class} \hspace{1cm} The defined class.
This chapter applies to the Enterprise Edition only

Slot Options

The slot options for `def-view-class` are `:db-kind` and `:db-info`. In addition the slot option `:type` is treated specially for View Classes.

`:db-kind` may be one of `:base`, `:key`, `:join`, or `:virtual`. The default is `:base`. Each value is described below:

`:base` This indicates that this slot corresponds to an ordinary attribute of the database view. You can name the database attribute by using the keyword `:column`. By default, the database attribute is named by the slot.

`:key` This indicates that this slot corresponds to part of the unique key for this view. A `:key` slot is also a `:base` slot. All View Classes must have `:key` fields that uniquely distinguish the instances, to maintain object identity.

To specify a key which spans multiple slots, each of the slots should have `:db-kind :key`. The underlying requirement is that tuples of the form (key1 ... keyN) are unique. The `:db-kind :key` slots do not need to be keys in the table.

`:join` This indicates that this slot corresponds to a join. A slot of this type will contain View Class objects.

`:virtual` 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.
This chapter applies to the Enterprise Edition only.

**: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)
Class Options  

**def-view-class** recognizes the following class options in addition to the standard class options defined for **defclass**:

\[
(:base-table \text{table-name})
\]

The slots of the class \text{name} will be read from the table \text{table-name}. If you do not specify the \text{:base-table} option, then \text{table-name} defaults to the name of the class.

Description  

The macro **def-view-class** creates a class called \text{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 \text{superclasses} is \text{nil} then **standard-db-object** automatically becomes the superclass of the newly-defined View Class. If \text{superclasses} is \text{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.
(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
      :db-info (: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  
\textbf{delete-instance-records} \hspace{0.5em} \textit{instance} \Rightarrow

Arguments  
\textit{instance} \hspace{1.5em} \text{An instance of a View Class.}

Values  
None.

Description  
The \textbf{delete-instance-records} function deletes the records represented by \textit{instance} from the database associated with it. If \textit{instance} has no associated database, \textbf{delete-instance-records} signals an error.

See also  
update-records
update-records-from-instance

\textbf{delete-records} \hspace{1.5em} \textit{Function}

Summary  
Deletes rows from a database table.

Package  
\textit{sql}

Signature  
\textbf{delete-records} \&key \hspace{0.5em} \textit{from where database} \Rightarrow

Arguments  
\textit{from} \hspace{1.5em} \text{A database table.}
\textit{where} \hspace{1.5em} \text{An SQL conditional statement.}
\textit{database} \hspace{1.5em} \text{A database.}

Values  
None.

Description  
The \textbf{delete-records} function deletes rows from a table specified by \textit{from} in which the \textit{where} condition is true. The argument \textit{database} specifies a database from which the records are to be removed, and defaults to \texttt{*default-database*}. 
This chapter applies to the Enterprise Edition only

See also  
*default-database*  
insert-records  
update-records

dele**te-sql-stream**

**Summary**  
Deletes a stream from the broadcast list for SQL commands or results traffic.

**Package**  
sql

**Signature**  
delete-sql-stream stream &key type database => deleted-stream

**Arguments**  
stream  
A stream or t.

type  
A keyword.

database  
A database.

**Values**  
deleted-stream  
The argument stream.

**Description**  
The function delete-sql-stream deletes the stream stream from the list of streams which receive SQL commands or results traffic.

To remove *standard-output* from the list, pass stream t.

The keyword type is :commands, :results or :both. It determines whether a stream for SQL commands traffic, results traffic, or both is deleted.

The default value of type is :commands. The default value for database is the value of *default-database*.
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**

```
disconnect &key database error => success
```

**Arguments**

- `database`: A database.
- `error`: A boolean.

**Values**

- `success`: A boolean.

**Description**

The function `disconnect` closes a connection to a database specified by `database`. If successful, `success` is `t` and if only one other connection exists, `*default-database*` is reset.

The default value for `database` is `*default-database*`. If `database` is a database object, then it is used directly. Otherwise, the list of connected databases is searched to find one with `database` as its connection specifications (see `connect`). If no such database is found, then if `error` and `database` are both non-nil an error is signaled, otherwise `disconnect` returns `nil`.

**Example**

```
(disconnect :database "test")
```

**See also**

- `connect`
- `connected-databases`
- `database-name`
- `*default-database*`
- `find-database`
- `reconnect`
- `status`

---

**do-query**

**Macro**

**Summary**

Repeatedly binds a set of variables to the results of a query, and executes a body of code using the bound variables.

**Package**

`sql`
do-query ((&rest args) query &key database not-inside-transaction get-all) &body body =>

Arguments
- **args**: A set of variables.
- **query**: A database query.
- **database**: A database.
- **not-inside-transaction**: A generalized boolean.
- **get-all**: A generalized boolean.
- **body**: A Lisp code body.

Values
None.

Description
The macro `do-query` repeatedly executes `body` within a binding of `args` on the attributes of each record resulting from `query`. `do-query` returns no values.

The default value of `database` is `*default-database*.`

`not-inside-transaction` and `get-all` may be useful when fetching many records through a connection with `database-type :mysql`. Both of these arguments have default value `nil`. See the section “Special considerations for iteration functions and macros” in the LispWorks User Guide for details.

Example
The following code repeatedly binds the result of selecting an entry in `ename` from the table `emp` to the variable `name`, and then prints `name` using the Lisp function `print`.

```lisp
(do-query ((name) [select [ename] :from [emp]])
  (print name))
```

See also
- `loop`
- `map-query`
- `query`
- `select`
- `simple-do-query`
This chapter applies to the Enterprise Edition only

**drop-index**

**Function**

**Summary**
Deletes an index from a database.

**Package**
sql

**Signature**
\texttt{drop-index index &key database =>}

**Arguments**
- \texttt{index} The name of an index.
- \texttt{database} A database.

**Values**
None.

**Description**
The function \texttt{drop-index} deletes \texttt{index} from \texttt{database}.
The default value of \texttt{database} is \texttt{*default-database*}.

**See also**
create-index
drop-table

**drop-table**

**Function**

**Summary**
 Deletes a table from a database.

**Package**
sql

**Signature**
\texttt{drop-table table &key database =>}

**Arguments**
- \texttt{table} The name of a table.
- \texttt{database} A database.

**Values**
None.

**Description**
The function \texttt{drop-table} deletes \texttt{table} from a database.
The default value of \texttt{database} is \texttt{*default-database*}. 
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**

```
drop-view-from-class class &key database =>
```

**Arguments**

- `class` A class.
- `database` A database.

**Values**

None.

**Description**

The function `drop-view-from-class` deletes a view or base table from `database` based on `class` which defines that view. The argument `database` has a default value of `*default-database*`.

**See also**

- `create-view-from-class`
- `*default-database*` `drop-view`

---

**enable-sql-reader-syntax**

**Function**

**Summary**

Turns on square bracket SQL syntax.

**Package**

`sql`

**Signature**

```
enable-sql-reader-syntax =>
```

**Arguments**

None.

**Values**

None.

**Description**

The function `enable-sql-reader-syntax` turns on square bracket syntax and sets the state so that `restore-sql-reader-syntax-state` will make the syntax enabled if it is subsequently disabled.

**See also**

- `disable-sql-reader-syntax`
- `locally-disable-sql-reader-syntax`
locally-enable-sql-reader-syntax
restore-sql-reader-syntax-state

**execute-command**

**Function**

Summary
Executes an SQL expression.

Package
sql

Signature
execute-command sql-exp &key database =>

Arguments

<table>
<thead>
<tr>
<th>sql-exp</th>
<th>Any SQL statement other than a query.</th>
</tr>
</thead>
<tbody>
<tr>
<td>database</td>
<td>A database.</td>
</tr>
</tbody>
</table>

Values
None.

Description
The function `execute-command` executes the SQL command specified by `sql-exp` for the database specified by `database`, which has a default value of `*default-database*`. The argument `sql-exp` may be any SQL statement other than a query. To run a stored procedure, pass an appropriate string. The call to the procedure needs to be wrapped in a PL/SQL `BEGIN` `END` pair, for example:

```sql
(sql:execute-command
 "BEGIN my_procedure(1, 'foo'); END;")
```

See also

*default-database*

query

**find-database**

**Function**

Summary
Returns a database, given a database or database name.

Package
sql
This chapter applies to the Enterprise Edition only

**Signature**

`find-database database &optional errorp => database, count`

**Arguments**

- `database`: A string or a database.

**Values**

- `database`: A database.
- `count`: An integer.

**Description**

The function `find-database`, given a string `database`, searches amongst the connected databases for one matching the name `database`. If there is exactly one such database, it is returned and the second return value `count` is 1. If more than one databases match and `errorp` is `nil`, then the most recently connected of the matching databases is returned and `count` is the number of matches. If no matching database is found and `errorp` is `nil`, then `nil` is returned. If none, or more than one, matching databases are found and `errorp` is true, then an error is signalled.

If the argument `database` is a database, it is simply returned.

**See also**

- `connect`
- `connected-databases`
- `database-name`
- `disconnect`
- `status`

### initialize-database-type

**Function**

**Summary**

Initializes a database type.

**Package**

`sql`

**Signature**

`initialize-database-type &key database-type => type`
The function \texttt{initialize-database-type} initializes a database type by loading code and appropriate database libraries according to the value of \texttt{database-type}. If \texttt{*default-database-type*} is not initialized, this function initializes it. It adds \texttt{database-type} to the list of initialized types. The initialized database type is returned.

**Example**

The following example shows how to use \texttt{initialize-database-type} to initialize the \texttt{:odbc} database type.

\begin{verbatim}
(require "odbc")
(in-package sql)
(setf *default-database-type* :odbc)
(initialize-database-type)
(print *initialized-database-types*)
\end{verbatim}

The ODBC database type is now initialized, and connections can be made to ODBC databases.

**See also**

\texttt{database-name}

\texttt{*initialized-database-types*}

\texttt{*default-database-type*}

**\*initialized-database-types***

**Variable**

**Summary**

A list of initialized database types.

**Package**

\texttt{sql}

**Initial Value**

\texttt{nil}

**Description**

The variable \texttt{*initialized-database-types*} contains a list of database types that have been initialized by calls to \texttt{initialize-database-type}. 
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

```lisp
(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

```lisp
(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`.

```lisp
(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.
Values

None.

Description

The function `instance-refreshed` is called inside `select` when its `refresh` argument is true and the instance `instance` has just been updated.

The supplied method on `standard-db-object` does nothing. If your application needs to take action when a View Class instance has been updated by

```
(select ... :refresh t)
```
then add an `instance-refresh` method specializing on your subclass of `standard-db-object`.

See also

`def-view-class`

`select`

---

**list-attribute-types**

<table>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Returns type information for a table’s attributes.</td>
</tr>
</tbody>
</table>

**Summary**

Returns type information for a table’s attributes.

**Package**

`sql`

**Signature**

`list-attribute-types table &key database owner => result`

**Arguments**

- `table` A table.
- `database` A database.
- `owner` `nil`, `:all` or a string.

**Values**

- `result` A list.

**Description**

The function `list-attribute-types` returns type information for the attributes in the table given by `table`. `database` has a default value of `*default-database*`. 
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

owner

nil, :all or a string.

Values

result

A list of attributes.

Description

The function `list-attributes` returns a list of attributes from table in database, which has a default value of `*default-database*`.

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.

See also

`attribute-type`

`list-attribute-types`

`list-tables`

**list-classes**

*Function*

Summary

Returns a list of View Classes connected to a given database.

Package

`sql`

Signature

`list-classes &key database root-class test => result-list`

Arguments

`database`

A database.

`root-class`

A class.

`test`

A test function.

Values

`result-list`

A list of class objects.

Description

The function `list-classes` collects all the classes below `root-class` (which defaults to `standard-db-object`) that are connected to the given database specified by `database`, 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*`.

### list-sql-streams

**Function**

- **Summary**: Returns the broadcast list of streams recording SQL commands or results traffic.
- **Package**: `sql`
- **Signature**: `list-sql-streams &key type database => streams`
- **Arguments**
  - `type`: A keyword.
  - `database`: A database.
- **Values**: `streams` A list.
- **Description**: The function `list-sql-streams` returns the broadcast list of streams recording SQL commands or results traffic.

Each element of `streams` is a stream or the symbol `t`, denoting `*standard-output*`.

The keyword `type` is one of `:commands` or `:results`, and determines whether to return a list of streams for SQL commands or results traffic.

The default value of `type` is `:commands`. The default value for `database` is the value of `*default-database*`.

**See also**

- `add-sql-stream`
- `delete-sql-stream`
- `sql-recording-p`
- `sql-stream`
This chapter applies to the Enterprise Edition only

```lisp
(start-sql-recording)
(stop-sql-recording)
```

### list-tables

**Function**

**Summary**

Returns a list of the table names in a database.

**Package**

sql

**Signature**

`list-tables &key database owner => table-list`

**Arguments**

- `database` A database.
- `owner` `nil`, `:all` or a string.

**Values**

- `table-list` A list of table names.

**Description**

The function `list-tables` returns the list of table names in `database`, which has a default value of `*default-database*`. If `owner` is `nil`, only user-owned tables are considered. This is the default.

If `owner` is `:all`, all tables are considered.

If `owner` is a string, this denotes a username and only tables owned by `owner` are considered.

**See also**

- `create-table`
- `drop-table`
- `list-attributes`
- `table-exists-p`

### lob-stream

**Class**

**Summary**

The LOB stream class.
Superclasses  

buffered-stream

Initargs

:lob-locator

A LOB locator.

:direction  

One of :input or :output.

:free-lob-locator-on-close

A generalized boolean.

Accessors

lob-stream-lob-locator

Description

The lob-stream class implements LOB streams in the Oracle LOB interface.

A lob-stream for input can be returned from select or query by specifying :input-stream as the type to return for the LOB column.

A lob-stream for output can be returned from select or query by specifying :output-stream as the type to return for the LOB column.

A lob-stream can be attached to an existing LOB locator by creating the stream explicitly.

direction specifies whether the stream is for input or output. The default value of direction is :input.

By default, if the stream is closed the LOB locator is freed, unless free-lob-locator-on-close is passed as nil. The default value of free-lob-locator-on-close is t.

Example

This creates an input stream connected to the LOB locator lob-locator:

(make-instance 'lob-stream :lob-locator lob-locator)

See also

query

select
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**  
sql

**Signature**  
locally-disable-sql-reader-syntax =>

**Arguments**  
None.

**Values**  
None.

**Description**  
The function `locally-disable-sql-reader-syntax` turns off square bracket syntax and does not change syntax state. This ensures that `restore-sql-reader-syntax-state` restores the current enable/disable state.

**Example**  
The intended use of `locally-disable-sql-reader-syntax` is in a file:

```
#. (locally-disable-sql-reader-syntax)
<Lisp code not using [...] syntax>
#. (restore-sql-reader-syntax-state)
```

**See also**  
disable-sql-reader-syntax  
enable-sql-reader-syntax  
locally-enable-sql-reader-syntax  
restore-sql-reader-syntax-state

**locally-enable-sql-reader-syntax**  
*Function*

**Summary**  
Turns on square bracket syntax and does not change syntax state.

**Package**  
sql
The function `locally-enable-sql-reader-syntax` turns on square bracket syntax and does not change the syntax state. This ensures that `restore-sql-reader-syntax-state` restores the current enable/disable state.

The intended use of `locally-enable-sql-reader-syntax` is in a file:

```
#. (locally-enable-sql-reader-syntax)
<code using [...] syntax>
#. (restore-sql-reader-syntax-state)
```

See also

`disable-sql-reader-syntax`
`enable-sql-reader-syntax`
`locally-disable-sql-reader-syntax`
`restore-sql-reader-syntax-state`

### Loop

Macro

Extends `loop` to provide a clause for iterating over query results.

```
loop {for|as} var [type-spec] being {the|each} {records|record} {in|of} query-expression [not-inside-transaction not-inside-transaction] [get-all get-all] => result
```

Arguments

- `var` A variable.
- `query-expression` An SQL query statement.
This chapter applies to the Enterprise Edition only

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>not-inside-transaction</td>
<td>A generalised boolean.</td>
</tr>
<tr>
<td>get-all</td>
<td>A generalised boolean.</td>
</tr>
</tbody>
</table>

**Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>result</td>
<td>A <code>loop</code> return value.</td>
</tr>
</tbody>
</table>

**Description**

The Common Lisp `loop` 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 `loop` 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 `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.

The additional loop keywords `not-inside-transaction` and `get-all` may be useful when fetching many records through a connection with `database-type :mysql`. See the section "Special considerations for iteration functions and macros" in the *LispWorks User Guide* for details.

**Example**

This extended `loop` example performs the following on each record returned as a result of a query: bind `name` 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 "~&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

get-all A generalized boolean.

<table>
<thead>
<tr>
<th>Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>result A sequence of type <code>output-type-spec</code> containing the results of the map function.</td>
<td></td>
</tr>
</tbody>
</table>

The function `map-query` returns the result of mapping `function` across the results of `query-exp`. The `output-type-spec` argument specifies the type of the result sequence as per the Common Lisp `map` function.

The default value of `database` is `*default-database*`. `not-inside-transaction` and `get-all` may be useful when fetching many records through a connection with `database-type`: `mysql`. Both of these arguments have default value `nil`. See the section “Special considerations for iteration functions and macros” in the LispWorks User Guide for details.

Example

This example binds `name` to each name in the employee table and prints it.

```lisp
(map-query
 nil
 #'(lambda (name) (print name))
 [select [ename] :from [emp] :flatp t])
```

See also
do-query
loop
print-query
query
select
simple-do-query

`*mysql-library-directories*` Variable

<table>
<thead>
<tr>
<th>Package</th>
<th>Initial Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>sql</td>
<td>&quot;C:\Program Files\MySQL\MySQL*\bin&quot;</td>
</tr>
</tbody>
</table>
The variable *mysql-library-directories* helps LispWorks for Windows to locate the MySQL library for use with database-type :mysql.

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:

"libmysql.dll"

On other platforms with pthreads:

"-lmysqlclient_r"

On other platforms without pthreads:

"-lmysqlclient"

Description The variable *mysql-library-path* helps the system to locate the MySQL library for use with database-type :mysql. 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**

```lisp
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.
**ora-lob-assign**

*Summary* Assigns a LOB to another LOB locator.

*Package* sql

*Signature* `ora-lob-assign src-lob-locator &key dest-lob-locator errorp => lob-locator`

*Arguments*
- `src-lob-locator` A LOB locator.
- `dest-lob-locator` A LOB locator.
- `errorp` A generalized boolean.

*Values* `lob-locator` A LOB locator.

*Description* The function `ora-lob-assign` assigns the underlying LOB for `src-lob-locator` to another LOB locator. If `dest-lob-locator` is `nil` then a new LOB locator is created and returned. Otherwise `dest-lob-locator` should be an existing LOB locator which is modified and returned. The default value of `dest-lob-locator` is `nil`.

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 OCILobAssign.

**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-char-set-form**

*Summary* Returns the character set form of a LOB.

*Package* sql
This chapter applies to the Enterprise Edition only

**Signature**

ora-lob-char-set-form  lob-locator &key errorp => charset

**Arguments**

*lob-locator*  
A LOB locator.

*errorp*  
A generalized boolean.

**Values**

*charset*  
A non-negative integer.

**Description**

The function **ora-lob-char-set-form** returns the char set form of the LOB underlying *lob-locator*.

*charset* 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.

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 OCILOBCharSetForm.

**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-char-set-id**

*Function*

**Summary**

Returns the database character set identifier of a LOB.

**Package**

sql

**Signature**

ora-lob-char-set-id  lob-locator &key errorp => db-charset-id

**Arguments**

*lob-locator*  
A LOB locator.

*errorp*  
A generalized boolean.

**Values**

*db-charset-id*  
A non-negative number.
The function ora-lob-char-set-id returns the database character set identifier of the LOB underlying lob-locator.

db-charset-id is 0 for a binary LOB.

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 OCILobCharSetID.

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-close

Closes an opened LOB.

Package sql

Signature ora-lob-close lob-locator &key errorp

Arguments lob-locator A LOB locator.

errorp A generalized boolean.

Description The function ora-lob-close closes a LOB which has been opened by ora-lob-open.

For more information see ora-lob-open.

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 OCILobClose.

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-copy**

*Function*

**Summary**
Copies part of an internal LOB.

**Package**
sql

**Signature**

```sql
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
This chapter applies to the Enterprise Edition only

described in the section "Inserting empty LOBs" in the *LispWorks User Guide*.

The default value of \textit{db} is the value of *\texttt{default-database}*.

\textbf{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.

\textbf{ora-lob-create-temporary} \hspace{1cm} \textit{Function}

\textbf{Summary} \hspace{0.5cm} Creates a temporary LOB.

\textbf{Package} \hspace{0.5cm} \texttt{sql}

\textbf{Signature} \hspace{0.5cm} \texttt{ora-lob-create-temporary db-or-lob-locator \&key errorp cache session-duration clob-p => lob-locator}

\textbf{Arguments} \hspace{0.5cm} \texttt{db-or-lob-locator}

A database or a LOB locator.

\texttt{errorp} \hspace{0.5cm} A generalized boolean.

\texttt{cache} \hspace{0.5cm} A generalized boolean.

\texttt{session-duration} \hspace{0.5cm} A generalized boolean.

\texttt{clob-p} \hspace{0.5cm} A generalized boolean.

\textbf{Values} \hspace{0.5cm} \texttt{lob-locator} \hspace{0.5cm} A LOB locator.

\textbf{Description} \hspace{0.5cm} The function \texttt{ora-lob-create-temporary} creates a temporary LOB.

\texttt{db-or-lob-locator} 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.

If an error occurs and \texttt{errorp} is true, an error is signaled. If \texttt{errorp} is false, the function returns an object of type \texttt{sql-database-error}. The default value of \texttt{errorp} is \texttt{nil}. 

537
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
ora-lob-free-temporary
ora-lob-is-temporary

### ora-lob-disable-buffering

**Function**

**Summary**
Disables the buffering of the Oracle client.

**Package**
sql

**Signature**
ora-lob-disable-buffering lob-locator &key errorp

**Arguments**
- **lob-locator**: A LOB locator.
- **errorp**: A generalized boolean.

**Description**
The function ora-lob-disable-buffering disables the buffering of the Oracle client. This function does not flush the buffers.

This function 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 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 CFILE `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.
ora-lob-enable-buffering

Summary
Enables the buffering of the Oracle client.

Package
sql

Signature
ora-lob-enable-buffering lob-locator &key errorp

Arguments
lob-locator A LOB locator.
errorp A generalized boolean.

Description
The function ora-lob-enable-buffering enables the buffering of the Oracle client. This function does not flush the buffers.

This function 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 OCILobEnableBuffering.

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-disable-buffering
ora-lob-flush-buffer

ora-lob-env-handle

Summary
Returns a foreign pointer to the environment handle of a LOB.

Package
sql

Signature
ora-lob-env-handle lob-locator => pointer
This chapter applies to the Enterprise Edition only

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lob-locator</td>
<td>A LOB locator.</td>
</tr>
</tbody>
</table>

**Values**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pointer</td>
<td>A foreign pointer of type sql:p-oci-env.</td>
</tr>
</tbody>
</table>

**Description**

The function `ora-lob-env-handle` returns a foreign pointer to the environment handle of the LOB underlying `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.

---

**ora-lob-erase**

**Function**

**Summary**

Erases part of an internal LOB.

**Package**

sql

**Signature**

`ora-lob-erase lob-locator offset amount &key errorp => erased`

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lob-locator</td>
<td>A LOB locator.</td>
</tr>
<tr>
<td>offset</td>
<td>A non-negative integer.</td>
</tr>
<tr>
<td>amount</td>
<td>A non-negative integer.</td>
</tr>
<tr>
<td>errorp</td>
<td>A generalized boolean.</td>
</tr>
</tbody>
</table>

**Values**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>erased</td>
<td>A non-negative integer.</td>
</tr>
</tbody>
</table>

**Description**

The function `ora-lob-erase` erases part of the LOB pointed to by `lob-locator`. That is, it fills part of the LOB with 0 for BLOBs or spaces for CLOBs.

The operation starts from offset `offset` into the LOB and erases `amount` of data in the LOB, or to the end of the LOB. Note that the offset starts from 1, and that `offset` and `amount` are in characters for CLOBs and bytes for BLOBs.
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.

**ora-lob-file-close**

**Function**

**Summary**
Closes a file LOB.

**Package**
sql

**Signature**
ora-lob-file-close file-lob-locator &key errorp

**Arguments**
file-lob-locator A file LOB locator.
errorp A generalized boolean.

**Description**
The function ora-lob-file-close closes the file that file-lob-locator is associated with.

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 OCILobFileClose.

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.
This chapter applies to the Enterprise Edition only

See also  ora-lob-file-open

**ora-lob-file-close-all**

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**

Summary  The predicate for whether a LOB file exists.

Package  sql
ora-lob-file-exists

**Signature**
signature: `ora-lob-file-exists lob-locator &key errorp => result`

**Arguments**
- `lob-locator`: A LOB locator.
- `errorp`: A generalized boolean.

**Values**
- `result`: A boolean.

**Description**
The function `ora-lob-file-exists` returns `t` if the file associated with the LOB exists. This function is applicable only to file LOBs (CFILE or BFILE).

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 OCILobFileExists.

**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-get-name

**Function**

**Summary**
Returns the directory and name for the file associated with a file LOB.

**Package**
`sql`

**Signature**
signature: `ora-lob-file-get-name lob-locator &key errorp => dir, filename`

**Arguments**
- `lob-locator`: A LOB locator.
- `errorp`: A generalized boolean.

**Values**
- `dir`: A string of length no greater than 30.
- `filename`: A string of length no greater than 255.
This chapter applies to the Enterprise Edition only

Description
The function `ora-lob-file-get-name` returns as multiple values the directory alias `dir` and the filename `filename` associated with the LOB denoted by `lob-locator`. The function is applicable only to file LOBs (CFILE or BFILE).

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 OCILobFileGetName.

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-is-open

**Function**

**Summary**
The predicate for whether a LOB file is open.

**Package**
`sql`

**Signature**
`ora-lob-file-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-file-is-open` returns `t` 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 `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 OCILobFileIsOpen.
**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**  
sql

**Signature**  
ora-lob-file-open file-lob-locator &key errorp

**Arguments**  
file-lob-locator  
A file LOB locator.

errorp  
A generalized boolean.

**Description**  
The function ora-lob-file-open opens the file that file-lob-locator is associated with. 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 OCILobFileOpen.

**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-set-name  
**Function**

**Summary**  
Sets the name of a file LOB.

**Package**  
sql
This chapter applies to the Enterprise Edition only

**Signature**

```lisp
ora-lob-file-set-name file-lob-locator dir-alias name &key errorp
```

**Arguments**

- `file-lob-locator` A file LOB locator.
- `dir-alias` A string or `nil`.
- `name` A string or `nil`.
- `errorp` A generalized boolean.

**Description**

The function `ora-lob-file-set-name` sets the directory alias and the name of the file LOB pointed to by `file-lob-locator`.

If `dir-alias` is a string it should be of length no greater than 30. If it is `nil` then the directory alias of the file LOB is not changed.

If `name` is a string it should be of length no greater than 255. If it is `nil` then the name of the file LOB is not changed.

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 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.

---

**ora-lob-flush-buffer**

**Function**

**Summary**

Flushes the buffer of the Oracle client.

**Package**

`sql`

**Signature**

```lisp
ora-lob-flush-buffer lob-locator &key free-buffer errorp
```

**Arguments**

- `lob-locator` A LOB locator.
free-buffer  A generalized boolean.
errorp      A generalized boolean.

Description The function ora-lob-flush-buffer flushes the buffer that is used by the Oracle client.
If free-buffer is true, it also frees the buffer. The default value of free-buffer is nil.
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 OCILobFlushBuffer.
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-free  Function

Summary  Frees a LOB locator.

Package  sql

Signature  ora-lob-free  lob-locator

Arguments  lob-locator  A LOB locator.

Description The function ora-lob-free frees the LOB locator lob-locator.
If lob-locator was retrieved inside an iteration macro or function (that is, one of map-query, do-query, simple-do-query and loop), it is freed before the next record is fetched, or when terminating the iteration for the last record.
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 rollback 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.

### ora-lob-free-temporary

**Function**

**Summary**
Frees a temporary LOB locator.

**Package**
`sql`

**Signature**

```
ora-lob-free-temporary temp-lob-locator &key errorp
```

**Arguments**

- `temp-lob-locator`  A temporary LOB locator.
- `errorp`  A generalized boolean.

**Description**

The function `ora-lob-free-temporary` frees a temporary LOB locator.

`temp-lob-locator` should be a temporary LOB locator as created by `ora-lob-create-temporary`.

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: 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.
This chapter applies to the Enterprise Edition 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 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.
**ora-lob-internal-lob-p**

**Function**

**Summary**
The predicate for internal LOBs.

**Package**
sql

**Signature**
ora-lob-internal-lob-p lob-locator => result

**Arguments**
lob-locator
A LOB locator.

**Values**
result
A boolean.

**Description**
The function *ora-lob-internal-lob-p* returns *t* if *lob-locator* is internal (BLOB, CLOB, or NCLOB). Otherwise it returns *nil*.

*Note:* this function is available only when the "oracle" module is loaded. See the section on the Oracle LOB interface in *LispWorks User Guide* for more information.

---

**ora-lob-is-equal**

**Function**

**Summary**
The comparison function for LOB locators.

**Package**
sql

**Signature**
ora-lob-is-equal lob-locator1 lob-locator2 => result

**Arguments**
lob-locator1
A LOB locator.
lob-locator2
A LOB locator.

**Values**
result
A boolean.

**Description**
The function *ora-lob-is-equal* returns *t* if *lob-locator1* and *lob-locator2* 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.
The SQL Package

This chapter applies to the Enterprise Edition only

Package  sql

Signature  ora-lob-is-temporary  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-temporary returns t if the LOB underlying lob-locator is temporary, that is, it was created by ora-lob-create-temporary.

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 OCILobIsTemporary.

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-load-from-file  Function

Summary  Loads data from a file LOB into a LOB.

Package  sql

Signature  ora-lob-load-from-file  dest-lob-locator src-lob-file amount &key src-offset dest-offset errorp

Arguments  dest-lob-locator  An internal LOB locator.
src-lob-file  A file LOB locator.
amount  A non-negative integer.
This chapter applies to the Enterprise Edition only

\[ \text{src-offset} \quad \text{A non-negative integer.} \]

\[ \text{dest-offset} \quad \text{A non-negative integer.} \]

\[ \text{errorp} \quad \text{A generalized boolean.} \]

**Description**

The function `ora-lob-load-from-file` loads the data from the `src-lob-file` into the destination LOB pointed to by `dest-lob-locator`.

The source LOB must be a BFILE and the destination must be an internal LOB.

The details of the operation are determined by `amount`, `src-offset` and `dest-offset`. `amount` and `dest-offset` are in characters for CLOB/NCLOB and are in bytes for BLOB. `src-offset` is in bytes. The offsets start from 1. The default value of `dest-offset` is 1 and the default value of `src-offset` is 1.

No conversion is performed by `ora-lob-load-from-file`, so if the destination is a CLOB/NCLOB, the source must already be in the right format.

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 OCILobReadFromFile. The Oracle documentation is ambiguous on whether it is mandatory to open the source LOB before calling this function.

**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-copy`

**ora-lob-lob-locator**

**Function**

**Summary**

Returns a foreign pointer to the underlying LOB locator.
Package sql

Signature `ora-lob-lob-locator lob-locator => pointer`

Arguments
- `lob-locator`: A LOB locator.

Values
- `pointer`: A foreign pointer.

Description
The function `ora-lob-lob-locator` returns a foreign pointer to the OCI LOB locator underlying `lob-locator`. `pointer` is of type `sql:p-oci-lob-locator` or `sql:p-oci-file`.

*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-locator-is-init**

Function

Summary The predicate for whether a LOB is initialized.

Package sql

Signature `ora-lob-locator-is-init lob-locator &key errorp => result`

Arguments
- `lob-locator`: A LOB locator.
- `errorp`: A generalized boolean.

Values
- `result`: A boolean.

Description
The function `ora-lob-locator-is-init` returns `t` if the LOB locator `lob-locator` is initialized.

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`. 
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**

**Function**

**Summary**

Opens a LOB.

**Package**

sql

**Signature**

ora-lob-open lob-locator &key errorp

**Arguments**

lob-locator A LOB locator.

errorp A generalized boolean.

**Description**

The function ora-lob-open opens the LOB pointed to by lob-locator, which can be an internal LOB or a file LOB.

Opening the LOB creates a transaction, so any updates associated with modifying the LOB are delayed until the ora-lob-close call. This saves round-trips and avoids extra work on the server side. However it is not mandatory to use ora-lob-open.

Calls to ora-lob-open must be strictly paired to calls to ora-lob-close, and the latter must be called before a call to commit. It is also an error to call ora-lob-open on a server LOB object that is already open, even if it has been opened via a different 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 OCILobOpen.
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

ora-lob-read-buffer

Function

Summary

Reads from a LOB into a buffer.

Package

sql

Signature

ora-lob-read-buffer lob-locator offset amount buffer &key
buffer-offset csid => amount-read, eof-or-error-p

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lob-locator</td>
<td>A LOB locator.</td>
</tr>
<tr>
<td>offset</td>
<td>A non-negative integer or nil.</td>
</tr>
<tr>
<td>amount</td>
<td>A non-negative integer.</td>
</tr>
<tr>
<td>buffer</td>
<td>A string, or a vector of element type</td>
</tr>
<tr>
<td></td>
<td>(unsigned-byte 8).</td>
</tr>
<tr>
<td>buffer-offset</td>
<td>A non-negative integer.</td>
</tr>
<tr>
<td>csid</td>
<td>A Character Set ID</td>
</tr>
</tbody>
</table>

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>amount-read</td>
<td>A non-negative integer.</td>
</tr>
<tr>
<td>eof-or-error-p</td>
<td>A boolean or an error object.</td>
</tr>
</tbody>
</table>

Description

The function ora-lob-read-buffer reads into buffer from the LOB pointed to by lob-locator.

offset specifies the offset to start reading from. It starts with 1, and specifies characters for CLOB/NCLOB/CFILE and bytes for BLOB/BFILE. If offset is nil then the offset after the end of the previous read operation is used (write operations are
This chapter applies to the Enterprise Edition only

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.
The SQL Package

This chapter applies to the Enterprise Edition only

(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 lob-locator file-name &key
offset file-offset if-exists

Arguments
lob-locator A LOB locator.
file-name A pathname designator.
offset A non-negative integer, or nil.
file-offset A non-negative integer, or nil.
if-exists A keyword or nil.

Description
The function ora-lob-read-into-plain-file writes the contents of a LOB into a file.

file-name specifies the file to write, which should be a standard file. The file is always opened in a binary mode, so if the
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.

\textit{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 \texttt{nil} then the operation starts from the end of the previous read operation. The default value of \textit{offset} is \texttt{nil}.

\textit{file-offset} specifies the offset into the file to start the operation from. If \textit{file-offset} is \texttt{nil} then it starts writing at the start of the file. The default value of \textit{file-offset} is \texttt{nil}.

\textit{if-exists} is passed to \texttt{open} when opening the file, with the standard Common Lisp meaning. The default value of \textit{if-exists} is \texttt{:error}.

\textbf{Note:} this function is available only when the "oracle" module is loaded. See the section on the Oracle LOB interface in the \textit{LispWorks User Guide} for more information.

See also \texttt{ora-lob-write-from-plain-file}

\textbf{ora-lob-read-foreign-buffer} \hspace{1cm} \textit{Function}

\textbf{Summary} \hspace{1cm} Reads from a LOB into a foreign buffer.

\textbf{Package} \hspace{1cm} \texttt{sql}

\textbf{Signature} \hspace{1cm} \texttt{ora-lob-read-foreign-buffer lob-locator offset amount foreign-buffer buffer-length &key buffer-offset csid => amount-read, eof-or-error-p}

\textbf{Arguments} \hspace{1cm} 

\begin{itemize}
  \item \texttt{lob-locator} \hspace{1cm} A LOB locator.
  \item \texttt{offset} \hspace{1cm} A non-negative integer or \texttt{nil}.
  \item \texttt{amount} \hspace{1cm} A non-negative integer.
  \item \texttt{foreign-buffer} \hspace{1cm} A FLI pointer.
  \item \texttt{buffer-length} \hspace{1cm} A non-negative integer.
\end{itemize}
### ora-lob-svc-ctx-handle

**Function**

**Summary**

Returns a foreign pointer to the context handle of a LOB.

**Package**

`sql`

**Signature**

`ora-lob-svc-ctx-handle lob-locator => pointer`

**Arguments**

- `lob-locator` A LOB locator.

**Values**

- `pointer` A foreign pointer of type `sql:p-oci-svc-ctx`.

---

**Values**

- `buffer-offset` A non-negative integer.
- `csid` A Character Set ID

**Description**

The function `ora-lob-read-foreign-buffer` reads from the LOB pointed to by `lob-locator` into the foreign buffer `foreign-buffer`.

This is just like `ora-lob-read-buffer` except that it reads from the LOB locator into a foreign buffer.

`foreign-buffer` is a FLI pointer to a buffer, which must be of size at least `buffer-length`.

**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.

See also

- `ora-lob-get-buffer`
- `ora-lob-read-buffer`
The function `ora-lob-svc-ctx-handle` returns a foreign pointer to the context handle of the LOB underlying `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.

---

**ora-lob-trim**

**Summary**
Trims an internal LOB.

**Package**
`sql`

**Signature**
`ora-lob-trim lob-locator new-size &key errorp`

**Arguments**
- `lob-locator`: A LOB locator.
- `new-size`: A non-negative integer.
- `errorp`: A generalized boolean.

**Description**
The function `ora-lob-trim` trims the LOB pointed to by `lob-locator` to a new size `new-size`, which must be smaller than its current size.

Note that `new-size` is in characters for CLOBs and bytes for BLOBs.

`ora-lob-trim` 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 OCIlobTrim.

**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.
Function

**ora-lob-write-buffer**

**Summary**
Writes a buffer to a LOB.

**Package**
sql

**Signature**
ora-lob-write-buffer lob-locator offset amount buffer &key
buffer-offset csid => amount-written, eof-or-error-p

**Arguments**
- **lob-locator**: A LOB locator.
- **offset**: A non-negative integer or nil.
- **amount**: A non-negative integer.
- **buffer**: A string, or a vector of element type (unsigned-byte 8).
- **buffer-offset**: A non-negative integer.
- **csid**: A Character Set ID

**Values**
- **amount-written**: A non-negative integer.
- **eof-or-error-p**: A boolean or an error object.

**Description**
The function **ora-lob-write-buffer** writes to the LOB pointed to by **lob-locator** from **buffer**.

**offset** specifies the offset to start writing to. It starts with 1, and specifies characters for CLOB/NCLOB/CFILE and bytes for BLOB/BFILE. If offset is nil 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.

**amount** is the amount to write, 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 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

- *ora-lob-element-type*
- *ora-lob-write-foreign-buffer*

---

**ora-lob-write-from-plain-file**

**Function**

**Summary**

Writes the contents of a file into a LOB.

**Package**

*sql*

**Signature**

*ora-lob-write-from-plain-file* *lob-locator* *file-name* &key

*offset* *file-offset* *if-does-not-exist*
Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lob-locator</td>
<td>A LOB locator.</td>
</tr>
<tr>
<td>file-name</td>
<td>A pathname designator.</td>
</tr>
<tr>
<td>offset</td>
<td>A non-negative integer, or nil.</td>
</tr>
<tr>
<td>file-offset</td>
<td>A non-negative integer, or nil.</td>
</tr>
<tr>
<td>if-does-not-exist</td>
<td>A keyword or nil.</td>
</tr>
</tbody>
</table>

Description

The function `ora-lob-write-from-plain-file` writes the contents of a file into a LOB.

`file-name` 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.

`offset` is the offset into the LOB from where to start writing. It starts from 1, counts characters in a CLOB, and if it is `nil` then the operation starts from the end of the previous write 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 reading at the start of the file. The default value of `file-offset` is `nil`.

`if-does-not-exist` is passed to `open` when opening the file, with the standard Common Lisp meaning. The default value of `if-does-not-exist` 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-read-into-plain-file`

**Function**

**ora-lob-write-foreign-buffer**

Summary

Writes a foreign buffer to a LOB.
This chapter applies to the Enterprise Edition only

Package  sql

Signature  
ora-lob-write-foreign-buffer  lob-locator  offset  amount  
foreign-buffer  buffer-length  &key  buffer-offset  csid  =>  amount-written,  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.
buffer-offset  A non-negative integer.
csid  A Character Set ID

Values  
amount-written  A non-negative integer.
eof-or-error-p  A boolean or an error object.

Description  
The function ora-lob-write-foreign-buffer writes to the LOB pointed to by lob-locator from buffer. This is just like ora-lob-write-buffer except that it writes the LOB locator from a foreign buffer. foreign-buffer is a FLI pointer to a buffer, which must be of size at least buffer-length.

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-get-buffer
ora-lob-write-buffer
Function

**print-query**

Summary Prints a tabulated version of records resulting from a query.

Package `sql`

Signature `print-query query-exp &key titles formats sizes stream database =>`

Arguments
- `query-exp` An SQL query expression.
- `titles` A list of strings.
- `formats` A list of strings.
- `sizes` A list.
- `stream` An output stream.
- `database` A database.

Values None.

Description The `print-query` function takes a symbolic SQL query expression and formatting information and prints onto `stream` a table containing the results of the query.

A list of strings to use as column headings is given by `titles`, which has a default value of `nil`.

The `formats` argument is a list of format strings used to print each attribute, and has a default value of `t`, which means that `~A` or `~VA` are used if `sizes` are provided or computed.

The field sizes are given by `sizes`. It has a default value of `t`, which specifies that minimum sizes are computed.

The output stream is given by `stream`, which has a default value of `t`. This specifies that `*standard-output*` is used.

Examples The following call prints out two even columns of names and salaries:
This chapter applies to the Enterprise Edition only

```lisp
(print-query (select [surname] [income] :from [person])
  :titles '("NAME" "SALARY")
)
```

See also

- map-query
- print-query
- select

**Function**

**query**

**Summary**

Queries a database and returns a list of values.

**Package**

sql

**Signature**

```lisp
query sql-exp &key database result-types flatp => result-list, field-names
```

**Arguments**

- `sql-exp` An SQL query statement to be performed.
- `database` A database.
- `result-types` A list of symbols.
- `flatp` A boolean.

**Values**

- `result-list` A list of values.
- `field-names` A list of strings.

**Description**

The function `query` is the basic SQL query function. It queries the database specified by `database` with an SQL query statement given by `sql-exp`.

The argument `database` defaults to `*default-database*`. `result-types` is a list of symbols such as `:string` and `:integer`, one for each field in the query, which are used to specify the types to return.

`flatp` is used as in `select`.
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:

```lisp
(sql:query "select * from some_table"
 :result-types '(nil :string))
```

```lisp
(sql:query [select [*]
 :from [some_table]
 :result-types '(nil :string)])
```

See also

do-query
eexecute-command
lob-stream
loop
map-query
select
simple-do-query

reconnect

Function

Summary
Reconnects a database to its underlying RDBMS.

Package
sql

Signature
reconnect &key database error force => success

Arguments

database
The database to be reconnected.

eroerror
A boolean.

force
A boolean.

Values
success
A boolean.
This chapter applies to the Enterprise Edition only

Description

The reconnect function reconnects database to its underlying RDBMS. If successful, success is t and the variable *default-database* is set to the newly reconnected database.

The default value for database is *default-database*. If database is a database object, then it is used directly. Otherwise, the list of connected databases is searched to find one with database as its connection specifications (see connect). If no such database is found, then if error and database are both non-nil an error is signaled, otherwise reconnect returns nil.

force controls whether an error should be signaled if the existing database connection cannot be closed. When non-nil (this is the default value) the connection is closed without error checking. When force is nil, an error is signaled if the database connection has been lost.

Note: force non-nil 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).

See also

connect
connected-databases
*default-database*

---

**restore-sql-reader-syntax-state**

**Summary**

Sets the enable/disable square bracket syntax state to reflect the last call to either disable-sql-reader-syntax or enable-sql-reader-syntax.

**Package**

sql

**Signature**

restore-sql-reader-syntax-state

**Arguments**

None.
The function \texttt{restore-sql-reader-syntax-state} sets the enable/disable state of the square bracket syntax to reflect the last call to either \texttt{enable-sql-reader-syntax} or \texttt{disable-sql-reader-syntax}. The default state of the square bracket syntax is disabled.

\textbf{See also}\quad \texttt{disable-sql-reader-syntax} \hfill \texttt{enable-sql-reader-syntax} \\
\hfill \texttt{locally-disable-sql-reader-syntax} \hfill \texttt{locally-enable-sql-reader-syntax}

\begin{description}
\item[rollback] \textbf{Function}
\item[Summary] Rolls back changes made to a database since the last commit.
\item[Package] \textit{sql}
\item[Signature] \texttt{rollback \&key database => nil}
\item[Arguments] \texttt{database} \quad A database.
\item[Values] \texttt{nil}
\item[Description] The function \texttt{rollback} rolls back changes made in \texttt{database} since the last commit, that is, changes made since the last commit are not recorded. The argument \texttt{database} defaults to \texttt{*default-database*}.
\item[See also] \texttt{commit} \hfill \texttt{with-transaction}
\end{description}
This chapter applies to the Enterprise Edition only

**select**  

**Function**  

**Summary**  

Selects data from a database given a number of specified constraints.

**Package**  

sql

**Signature**  

```
select &rest selections &key all set-operation distinct from result-types flatp where group-by having database order-by refresh for-update => result-list
```

**Arguments**

- **selections** A set of database identifiers or strings.
- **all** A boolean.
- **set-operation** An SQL operation.
- **distinct** A boolean.
- **from** An SQL table.
- **result-types** A list of symbols.
- **flatp** A boolean.
- **where** An SQL condition.
- **group-by** An SQL condition.
- **having** An SQL condition.
- **database** A database.
- **order-by** An SQL condition.
- **refresh** A boolean.
- **for-update** t, :nowait, a string or a list.

**Values**  

- **result-list** A list of selections.

**Description**  

The function `select` selects data from `database`, which has a default value of `*default-database*`, 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 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
This chapter applies to the Enterprise Edition only

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:

```lisp
(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:

```lisp
(select [surname] :from [person] :flatp t)
```
In this example data in the attribute `largenum`, which is of a vendor-specific large numeric type, is returned to Lisp as strings:

```lisp
=> ("Brown" "Jones" "Smith")
```

In this example the second column of `some_table` is a date that we want to return as a string:

```lisp
(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:

```lisp
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")
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:

```lisp
(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**

`sql`

**Signature**

```
simple-do-query (values-list query &key names-list database
                      not-inside-transaction get-all) &body body =>
```

**Arguments**

- `values-list` A variable.
- `query` A database query.
- `names-list` A variable, or nil.
- `database` A database.
- `not-inside-transaction` A generalized boolean.
- `get-all` A generalized boolean.
- `body` A Lisp code body.
Values
None.

Description
The macro `simple-do-query` repeatedly executes `body` within a binding of `values-list` to the attributes of each record resulting from `query`.

If a variable `names-list` is supplied, then it is bound to a list of the column names for the query during the execution of `body`. The default value of `names-list` is `nil`.

`simple-do-query` returns no values.

The default value of `database` is `*:default-database*`.

`not-inside-transaction` and `get-all` may be useful when fetching many records through a connection with `database-type :mysql`. Both of these arguments have default value `nil`. See the section “Special considerations for iteration functions and macros” in the `LispWorks User Guide` for details.

Example
```
(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

See also
`do-query`
`loop`
`map-query`
`query`
`select`
This chapter applies to the Enterprise Edition only

**sql**

- **Function**

  **Summary**
  Generates SQL from a set of expressions.

  **Package**
  sql

  **Signature**
  sql &rest args => sql-expression

  **Arguments**
  args  
  A set of expressions.

  **Values**
  sql-expression  
  An SQL expression.

  **Description**
  The function sql generates SQL from a set of expressions given by args. Each argument to sql is translated into SQL and then the args are concatenated with a single space between each pair. The rules for translation into SQL, based on the type of each individual argument \( x \), are as follows:

  
  string => (format nil "'~A'" x)

  nil => "NULL"

  symbol => (symbol-name x)

  number => (princ-to-string x)

  list => (format nil "(~{~A~^,~})" (mapcar #'sql x))

  vector => (format nil "(~{~A~^,~})" (map 'list #'sql x))

  sql-expression => x

  Any other symbol => error

**See also**

  sql-expression
  sql-operation
  sql-operator

---

**sql-connection-error**

- **Condition**

  **Package**
  sql
Superclasses  sql-database-error
Subclasses  sql-fatal-error
            sql-timeout-error
Description  The condition class sql-connection-error is used to signal an error with the connection to the database.

sql-database-data-error  Condition
Package  sql
Superclasses  sql-database-error
Description  The condition class sql-database-data-error 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  sql
Superclasses  simple-error
Subclasses  sql-connection-error
            sql-database-data-error
            sql-temporary-error
Accessors  sql-error-error-id
            sql-error-secondary-error-id
            sql-error-database-message
Description  The condition class sql-database-error is used to signal errors in the database interface that Common SQL uses.
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

sql-expression Function

Summary Generates an SQL expression from the given keywords.

Package sql

Signature sql-expression &key string table alias attribute type => sql-result

Arguments

string A string.
table A table in a database.
alias A table alias.
attribute An attribute.
type A type.
The function `sql-expression` generates an SQL expression from the given keywords.

Valid combinations of the arguments are:

- `string`
- `table`
- `table` and `alias`
- `table` and `attribute`
- `table`, `attribute`, and `type`
- `table` or `alias`, and `attribute`
- `table` or `alias`, and `attribute` and `type`
- `attribute` and `type`

See also `sql`, `sql-operation`, `sql-operator`
sql-fatal-error

Condition

Package sql
Superclasses sql-connection-error
Description The condition class sql-fatal-error is used to signal errors that mean the connection can no longer be used.

*sql-libraries*

Variable

Package sql
Initial Value nil
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 ORACLE_HOME) 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 sql
Initial Value nil
Description The variable *sql-loading-verbose* controls the verbosity of messages while loading the database libraries.

Note: applicable only on Unix.
**sql-operation**

### Function

**Summary**
Generates an SQL statement from an operator and arguments.

**Package**
sql

**Signature**

```
sql-operation op &rest args => sql-result
sql-operation sql-function name &rest args => sql-result
sql-operation sql-operator inop1 left &rest rights => sql-result
sql-operation sql-boolean-operator inop2 left &rest rights => sql-result
```

**Arguments**

- **op**
  An operator.
- **args**
  A set of arguments for op.
- **name**
  An arbitrary function.
- **args**
  A set of arguments for name.
- **inop1**
  An infix operator with non-boolean result.
- **inop2**
  An infix operator that returns a boolean.
- **left**
  Argument to be placed on the left of an infix operator.
- **rights**
  Arguments to be placed on the right of an infix operator.

**Values**

- **sql-result**
  An SQL expression.

**Description**
The function `sql-operation` takes an operator and its arguments, and returns an SQL expression.

`(sql-operation op args)`
is shorthand for

`(apply (sql-operator op) args).`
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.

```lisp
(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
#<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`:

```lisp
(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
sql

Signature
sql-operator symbol => sql-symbol

Arguments
symbol
A symbol naming an SQL operator.

Values
sql-symbol
A symbol.

Description
The function sql-operator takes an operator as an argument and returns the Lisp symbol for the operator.

See also
sql
sql-expression
sql-operation

sql-recording-p

Function
Summary
A predicate for determining if SQL commands or results traffic is being recorded.

Package
sql

Signature
sql-recording-p &key type database => recording-p

Arguments
type
One of :commands or :results.
This chapter applies to the Enterprise Edition only

\[ database \] A database.

Values \[ recording-p \] A boolean.

Description The function sql-recording-p returns t if type is :commands and SQL commands traffic is being recorded, or if type is :results and SQL results traffic is being recorded. Otherwise it returns nil.

The default value of type is :commands. The default value of database is the value of *default-database*.

See also add-sql-stream
delete-sql-stream
list-sql-streams
sql-stream
start-sql-recording
stop-sql-recording

\textbf{sql-stream} Function

Summary Returns the broadcast stream used for recording SQL commands or results traffic

Package sql

Signature \[ sql-stream &key type database => stream \]

Arguments \[ type \] One of :commands or :results.
\[ database \] A database.

Values \[ stream \] A broadcast stream.

Description The function sql-stream 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

**sql-timeout-error**

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.
### database

A database.

**Values**

None.

**Description**

The function `start-sql-recording` 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 `*standard-output*`. You can modify the broadcast list of recording streams using `add-sql-stream` and `delete-sql-stream`.

`type` is one of `:commands`, `:results` or `:both`. It determines whether SQL commands traffic, results traffic or both is recorded.

The default value of `type` is `:commands`. The default value for `database` is the value of `*default-database*`.

**See also**

`add-sql-stream`

`delete-sql-stream`

`list-sql-streams`

`sql-stream`

`sql-recording-p`

`stop-sql-recording`

---

### status

**Summary**

Returns status information for the connected databases and initialized database types.

**Package**

`sql`

**Signature**

`status &optional full =>`

**Arguments**

`full` A boolean.
This chapter applies to the Enterprise Edition only

Values
None.

Description
The function \texttt{status} prints status information to the standard output, for the connected databases and initialized database types.

If \texttt{full} is \texttt{t}, detailed status information is printed. The default value of \texttt{full} is \texttt{nil}.

See also
\texttt{connect}, \texttt{connected-databases}, \texttt{database-name}, \texttt{disconnect}, \texttt{find-database}

\textbf{stop-sql-recording} \quad \textit{Function}

Summary
Stops recording SQL commands or results traffic.

Package
\texttt{sql}

Signature
\texttt{stop-sql-recording \&key type database =>}

Arguments
\begin{itemize}
\item \texttt{type} \quad \texttt{A keyword.}
\item \texttt{database} \quad \texttt{A database.}
\end{itemize}

Values
None.

Description
The function \texttt{stop-sql-recording} stops recording SQL commands or results traffic.

\texttt{type} is one of \texttt{:commands}, \texttt{:results} or \texttt{:both}. It determines whether the recording of SQL commands traffic, results traffic or both is stopped.

The default value of \texttt{type} is \texttt{:commands}. The default value for \texttt{database} is \texttt{*default-database*}. 

593
See also  
add-sql-stream  
delete-sql-stream  
list-sql-streams  
sql-recording-p  
sql-stream  
start-sql-recording

**table-exists-p**

**Function**

**Summary**  
A predicate for the existence of a table.

**Package**  
sql

**Signature**  
`table-exists-p table &key database owner => result`

**Arguments**  
`table`  
A potential table name.

`database`  
A database.

`owner`  
nil, :all or a string.

**Values**  
`result`  
a boolean.

**Description**  
The function `table-exists-p` determines whether there is a table named `table` in database `database`.

If `owner` is `nil`, only user-owned tables are considered. This is the default.

If `owner` is `:all`, all tables are considered.

If `owner` is a string, this denotes a username and only tables owned by `owner` are considered.

The default value of `database` is `*default-database*`.

See also  
list-tables
update-instance-from-records  

**Generic Function**

**Summary**
Updates a View Class instance.

**Package**
sql

**Signature**
update-instance-from-records  

instance &key database => instance

**Arguments**

- **instance**
  An instance of a View Class.

- **database**
  A database.

**Values**

- **instance**
  The updated View Class instance.

**Description**
The generic function update-instance-from-records updates the values in the slots of the View Class instance instance using the data in the database database.

database defaults to the database that instance is associated with, or the value of *default-database*. If instance is associated with a database, then database must be that same database.

The argument slot is the CLOS slot name; the corresponding column names are derived from the View Class definition.

The update is not recursive on joins. Join slots (that is, slots with :db-kind :join) are updated, but the joined objects are not updated.

**See also**

- def-view-class
- update-slot-from-record

update-objects-joins  

**Function**

**Summary**
Updates the remote join slots.
Signature

\texttt{update-objects-joins objects &key slots force-p class-name max-len}

Arguments

- \texttt{objects} A list of database objects.
- \texttt{slots} A list of slot names, or \texttt{t}.
- \texttt{force-p} A boolean.
- \texttt{class-name} The class of the objects, or \texttt{nil}.
- \texttt{max-len} A non-negative integer, or \texttt{nil}.

Description

The function \texttt{update-objects-joins} updates the remote join slots, that is those slots defined without \texttt{:retrieval :immediate}.

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.

\texttt{objects} is a list of database objects. If \texttt{class-name} is non-\texttt{nil}, then all the database objects are of this class. If \texttt{class-name} is \texttt{nil}, then all the database objects are of the class of the first database object in the list \texttt{objects}.

If \texttt{objects} is \texttt{nil}, then \texttt{update-objects-joins} does nothing.

\texttt{class-name} specifies a class containing all the database objects in the list \texttt{objects}. If \texttt{class-name} is \texttt{nil} (the default) then the class of the first database object is used.

\texttt{slots} provides a list of the names of slots to update. Each of these slots should be a remote join slot (as defined above).

\texttt{slots} can also be \texttt{t}, meaning update all the remote join slots.

The default value of \texttt{slots} is \texttt{t}.

\texttt{force-p} controls whether to force the update of all values in the objects. If \texttt{force-p} is \texttt{nil}, then slots which are already are not updated. The default value of \texttt{force-p} is \texttt{t}. 
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

**update-records**

<table>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
</table>

**Summary** Changes the values of fields in a table.

**Package** sql

**Signature** `update-records table &key attributes values av-pairs where database =>`

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>table</code></td>
<td>A database table.</td>
</tr>
<tr>
<td><code>attributes</code></td>
<td>A set of columns.</td>
</tr>
<tr>
<td><code>values</code></td>
<td>A set of values.</td>
</tr>
<tr>
<td><code>av-pairs</code></td>
<td>An association list alternative to <code>attributes</code> and <code>values</code>.</td>
</tr>
<tr>
<td><code>where</code></td>
<td>A condition.</td>
</tr>
<tr>
<td><code>database</code></td>
<td>A database.</td>
</tr>
</tbody>
</table>

**Values** None.
Description

The function `update-records` changes the values of existing fields in `table` with columns specified by `attributes` and `values` (or `av-pairs`) where the `where` condition is true.

See also

- `delete-instance-records`
- `delete-records`
- `insert-records`
- `update-records-from-instance`

**update-records-from-instance**

**Generic Function**

**Summary**

Updates a set of specified records in a database.

**Package**

`sql`

**Signature**

`update-records-from-instance instance &key database =>`

**Arguments**

- `instance` An instance of a View Class.
- `database` A database.

**Values**

None.

**Description**

The generic function `update-records-from-instance` updates the records in `database` represented by `instance`. If the instance is already associated with a database, that database is used, and `database` is ignored. If `instance` is not yet associated with a database, a record is created for `instance` in the appropriate table of `database` and the instance becomes associated with that database.

`update-records-from-instance` only updates the records from the base slots of `instance` - it doesn’t look at the join slots.

See also

- `def-view-class`
- `delete-instance-records`
- `update-records`
update-record-from-slot  

**Generic Function**

**Summary** Updates an individual data item from a slot.

**Package** `sql`

**Signature**

```
update-record-from-slot instance slot &key database
```

**Arguments**

- `instance` An instance of a View Class.
- `slot` A slot.
- `database` A database.

**Values** None.

**Description**

The generic function `update-record-from-slot` updates an individual data item in the column represented by `slot`. The `database` is only used if `instance` is not yet associated with any database, in which case a record is created in `database`. Only `slot` is initialized in this case; other columns in the underlying database receive default values. The argument `slot` is the CLOS slot name; the corresponding column names are derived from the View Class definition.

**See also**

- `def-view-class`
- `update-records-from-instance`

update-slot-from-record  

**Generic Function**

**Summary** Updates a slot in a View Class instance.

**Package** `sql`

**Signature**

```
update-slot-from-record instance slot => instance
```

**Arguments**

- `instance` An instance of a View Class.
slot A slot name.

Values

instance The updated View Class instance.

Description

The generic function \texttt{update-slot-from-record} updates the value in the slot \texttt{slot} of the View Class instance \texttt{instance} using the records in the database.

\texttt{instance} must be associated with a database.

The argument \texttt{slot} is the CLOS slot name; the corresponding column names are derived from the View Class definition.

The update is not recursive on joins. Join slots (that is, slots with \texttt{:db-kind :join}) are updated, but the joined objects are not updated.

See also

\texttt{def-view-class}
\texttt{update-instance-from-records}

\textbf{with-transaction} \hfill \textit{Macro}

Summary

Performs a body of code within a transaction for a database.

Package

\texttt{sql}

Signature

\texttt{with-transaction &key database &body body => results}

Arguments

\texttt{database} A database.

\texttt{body} A set of Lisp expressions.

Values

\texttt{results} The values returned by \texttt{body}.

Description

The macro \texttt{with-transaction} executes \texttt{body} within a transaction for \texttt{database} (which defaults to \texttt{*default-database*}). The transaction is committed if the body finishes successfully
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 employees 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
The SQL Package

This chapter applies to the Enterprise Edition only
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**
stream

**Superclasses**
fundamental-stream

**Subclasses**
lob-stream
string-stream
socket-stream

**Initargs**

:direction One of :input, :output or :io. This argument is required.

:element-type One of base-char, simple-char or character.
The class `buffered-stream` 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 `stream-read-buffer`, `stream-write-buffer` and `stream-element-type` for each subclass of `buffered-stream`. 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 `stream-listen`, `stream-check-eof-no-hang` and `close` as well.

The initargs are handled by the method `(method initialize-instance :after (buffered-stream))` as follows:

Input and/or output buffers are created based on the value `direction`. There is no default value, and you must supply a value.

`element-type` determines the `stream-element-type` of the stream. The default is `base-char`. For binary streams, use `base-char`.

All the methods in the User Defined Streams protocol are defined for `buffered-stream` as follows:

- The methods on `stream-read-char`, `stream-read-line`, `stream-read-sequence`, `stream-unread-char`, `stream-read-char-no-hang`, `stream-clear-input` handle input from the buffer. They each call `stream-fill-buffer` to fill the empty buffer as required.

- The methods on `stream-write-char`, `stream-write-string`, `stream-write-sequence`, `stream-clear-output`, `stream-finish-output`, `stream-force-output` and `stream-line-column` handle output to the buffer. They each call `stream-flush-buffer` to make the buffer empty as required.

- There are `:around` methods on `stream-listen` and `close` 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  

**Summary**  
A stream class for binary output.

**Package**  
stream

**Superclasses**  
fundamental-binary-stream  
fundamental-output-stream

**Description**  
The class `fundamental-binary-output-stream` provides a class for generating customized binary output stream classes. A method for `stream-write-byte` should be provided.

**See also**  
fundamental-binary-stream  
fundamental-output-stream  
stream-write-byte

fundamental-binary-stream  

**Summary**  
A class for binary streams.

**Package**  
stream

**Superclasses**  
fundamental-stream

**Subclasses**  
fundamental-binary-input-stream  
fundamental-binary-output-stream

**Description**  
The class `fundamental-binary-stream` is the superclass of the binary input and output stream classes. A method for `stream-element-type` should be provided for instantiable subclasses of this class.

**See also**  
fundamental-binary-input-stream  
fundamental-binary-output-stream  
fundamental-stream  
stream-element-type
fundamental-character-input-stream  

Class

Summary A class that should be included in stream classes for character input.

Package stream

Superclasses 
- fundamental-character-stream
- fundamental-input-stream

Subclasses None.

Description The class fundamental-character-input-stream 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 
- fundamental-character-stream
- fundamental-input-stream
- stream-clear-input
- stream-listen
- stream-peek-char
- stream-read-char
- stream-read-char-no-hang
- stream-read-line
- stream-read-sequence
- stream-unread-char

fundamental-character-output-stream  

Class

Summary A class that should be included in stream classes for character output.
Package  
stream

Superclasses  
fundamental-character-stream  
fundamental-output-stream

Subclasses  
None.

Description  
The class fundamental-character-output-stream 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  
fundamental-character-stream  
fundamental-input-stream  
stream-clear-output  
stream-finish-output  
stream-force-output  
stream-start-line-p  
stream-terpri  
stream-line-column  
stream-write-char  
stream-write-sequence  
stream-write-string

fundamental-character-stream  
Class

Summary  
A class whose inclusion provides a method for stream-element-type that returns character.

Package  
stream

Superclasses  
fundamental-stream
Subclasses

fundamental-character-input-stream
fundamental-character-output-stream

Description

The class `fundamental-character-stream` is a superclass for character streams. Its inclusion provides a method for the generic function `stream-element-type` that returns the symbol `character`.

See also

fundamental-character-input-stream
fundamental-character-output-stream
fundamental-stream
stream-element-type

---

**fundamental-input-stream**

*Class*

Summary

A class whose inclusion causes `input-stream-p` to return `t`.

Package

stream

Superclasses

fundamental-stream

Subclasses

fundamental-binary-input-stream
fundamental-character-input-stream

Description

The `fundamental-input-stream` class is a superclass to the binary and character input classes. Its inclusion causes the generic function `input-stream-p` to return `t`.

See also

fundamental-binary-input-stream
fundamental-character-input-stream
fundamental-stream
input-stream-p
### fundamental-output-stream

**Class**

**Summary**
A class whose inclusion causes `output-stream-p` to return `t`.

**Package**
stream

**Superclasses**
fundamental-stream

**Subclasses**
- fundamental-binary-output-stream
- fundamental-character-output-stream

**Description**
The `fundamental-output-stream` class is a superclass to the binary and character output classes. Its inclusion causes the generic function `output-stream-p` to return `t`.

**See also**
- fundamental-binary-output-stream
- fundamental-character-output-stream
- fundamental-stream
- input-stream-p

### fundamental-stream

**Class**

**Summary**
A class whose inclusion causes `streamp` to return `t`.

**Package**
stream

**Superclasses**
standard-object
stream

**Subclasses**
- fundamental-binary-stream
- fundamental-character-stream
- fundamental-input-stream
- fundamental-output-stream

**Description**
The class `fundamental-stream` is a superclass to the fundamental input, output, character and binary streams. Its inclusion causes `streamp` to return `t`.

---

610
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 stream column => result

Arguments

stream A stream.

column An integer.

Values

result A boolean.

Description

The generic function stream-advance-to-column writes enough blank spaces to stream to ensure that the next character is written at column. 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

close
fundamental-binary-stream
fundamental-character-stream
fundamental-input-stream
fundamental-output-stream
open-stream-p
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 stream => result

Arguments
stream An input stream.

Values
result nil or :eof.

Description
The generic function stream-check-eof-no-hang determines if the data source of the stream is at end of file, without hanging.

stream should be an instance of a subclass of buffered-stream.

result is :eof if stream is at end of file and nil otherwise.

There is a built-in method specialized on buffered-stream which returns :eof in all cases.

See also
buffered-stream

stream-clear-input

Generic Function

Summary
Implements clear-input.

Package
stream

Signature
stream-clear-input stream => nil

Arguments
stream 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.
The generic function `stream-file-position` implements `file-position`. `stream-file-position` is called when `file-position` is called with one argument.

`(setf stream:stream-file-position)` is called when `file-position` is called with two arguments.

The return value is returned by `file-position`. For the setf function, this is a slight anomaly because setf functions normally return the new value. However in this case it should return the `success-p` value mandated by the ANSI Common Lisp standard.

The default methods specialized on `stream` return `nil`.

---

**stream-fill-buffer**

*Generic Function*

**Summary**
Fills the stream buffer.

**Package**
`stream`

**Signature**
`stream-fill-buffer stream => result`

**Arguments**
`stream` An input stream.

**Values**
`result` A generalized boolean.

**Description**
The generic function `stream-fill-buffer` 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.
stream-flush-buffer

Generic Function

Summary Flushes a stream's buffer.

Package stream

Signature stream-flush-buffer stream => result

Arguments stream An output stream.

Values result A generalized boolean.

Description The generic function stream-flush-buffer is called by the writing functions to flush a stream buffer to the underlying data sink.

stream should be an instance of a subclass of buffered-stream.

Before returning, stream-flush-buffer must set the output index of stream so that more characters can be written to the buffer. If desired, the output buffer and limit can be set too.

There is a built-in method specialized on buffered-stream which usually suffices. It calls stream-write-buffer with the currently active part of the stream's output buffer and sets the output index to 0.

result is true if the buffer was flushed.

See also buffered-stream
stream-write-buffer
**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.
The STREAM Package

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 => column

Arguments

stream A stream.

description

The generic function stream-line-column returns the column number where the next character will be written from stream, or nil if this is not meaningful for the stream. This function is used in the implementation of print and the format -t 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 nil.

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  

\[ \text{stream-listen} \text{ stream } \Rightarrow \text{ result} \]

Arguments  
\text{stream} \quad \text{A stream.}

Values  
\text{result} \quad \text{A generalized boolean.}

Description  

The generic function \text{stream-listen} is called to determine if there is data immediately available on the stream \text{stream}, without hanging.

\text{result} should be true if there is input, and \text{nil} otherwise (including at end of file).

This method must be implemented for subclasses of \text{buffered-stream} that handle input.

There is a built-in primary method specialized on \text{buffered-stream} which returns \text{nil}. There is a built-in :around method specialized on \text{buffered-stream} 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 \text{buffered-stream} need only check the underlying data source.

The built-in method on \text{fundamental-input-stream} uses \text{stream-read-char-no-hang} and \text{stream-unread-char}. Most streams should define their own method as this is usually trivial and more efficient than the method provided.

See also  
\text{buffered-stream}  
\text{stream-read-char-no-hang}  
\text{stream-unread-char}

\text{stream-output-width}  

\text{Generic Function}

Summary  

Used by the pretty printer to determine the output width when \text{*print-right-margin*} is \text{nil}.

Package  
\text{stream}
The generic function stream-output-width is used by the pretty printer to determine the output width when *print-right-margin* is *nil*. It returns result, the integer width of stream in units of ems, or *nil* if the width is not known. The default method provided by fundamental-stream returns *nil*.

See also fundamental-stream

### stream-peek-char

**Generic Function**

**Summary**

A generic function used by peek-char that returns a character on a given stream without removing it from the stream buffer.

**Package**

stream

**Signature**

stream-peek-char stream => result

**Arguments**

stream A stream.

**Values**

result A character or :EOF symbol.

**Description**

The generic function stream-peek-char is used to implement peek-char, and corresponds to a peek-type of *nil*. The default method reads a character from the stream without removing it from the stream buffer, by using stream-read-char and stream-unread-char.
stream-read-buffer

Summary
Reads data into the stream buffer.

Package
stream

Signature
stream-read-buffer stream buffer start end => result

Arguments
stream An input stream.
buffer A stream buffer.
start, end Bounding indexes for a subsequence of buffer.

Values
result A non-negative integer.

Description
The generic function stream-read-buffer is called by stream-fill-buffer to place characters into the region of the buffer buffer bounded by start and end.

stream should be an instance of a subclass of buffered-stream.

stream-read-buffer should block until some data is available. result 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.

See also
buffered-stream
stream-fill-buffer
The STREAM Package

**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 stream => result`

**Arguments**  
`stream`  
An input stream.

**Values**  
`result`  
An integer or `:eof`.

**Description**  
The generic function `stream-read-byte` is used by `read-byte`, and returns either an integer read from the binary stream specified by `stream`, or the keyword `:eof`.

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`.

A method should be defined for a subclass of `fundamental-binary-input-stream`.

**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 stream => character`
Arguments

stream  An input stream.

Values

character  A character or the :EOF symbol.

Description

The generic function stream-read-char reads one item from stream. 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 fundamental-character-input-stream must define a method for this function.

See also

fundamental-character-input-stream
stream-unread-char

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  

stream

Signature

stream-read-char-no-hang stream => result

Arguments

stream  An input stream.

Values

result  Either a character, an :EOF symbol, or nil.

Description

The generic function stream-read-char-no-hang implements read-char-no-hang. 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 fundamental-character-input-stream simply calls stream-read-char which is sufficient for file streams, but interactive streams should define their own method.

See also

fundamental-character-input-stream
stream-read-char
**stream-read-line**  
*Generic Function*

**Summary**  
Returns a string read from a stream.

**Package**  
stream

**Signature**  
stream-read-line stream => result terminated

**Arguments**  
stream  
An input stream.

**Values**  
result  
A string or :eof.

terminated  
A boolean.

**Description**  
The generic function stream-read-line reads a line of characters from stream and returns this line as a string. If the string is terminated by an end-of-file instead of a newline then terminated is t.

The default method uses repeated calls to stream-read-char, and uses stream-element-type to determine the element-type of its result.

**See also**  
fundamental-character-input-stream  
stream-element-type  
stream-read-char

---

**stream-read-sequence**  
*Generic Function*

**Summary**  
Reads a number of items from a stream into a sequence.

**Package**  
stream

**Signature**  
stream-read-sequence stream sequence start end => index

**Arguments**  
stream  
A stream.

sequence  
A sequence.
The generic function `stream-read-sequence` reads from `stream` into `sequence`. Elements from the `start` of `sequence` are replaced by elements from `stream` until `end` in `sequence` or the end-of-file in `stream` is reached. The index of the first element in `sequence` that is not replaced is returned.

A default method is provided by `fundamental-character-input-stream` which makes repeated calls to `stream-read-char` and uses `(setf elt)` to insert characters into `sequence`. A default method is provided by `fundamental-binary-input-stream` that makes repeated calls to `stream-read-byte` and also uses `(setf elt)` to insert bytes into `sequence`. 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`
Description

The generic function `stream-read-timeout` reads the current `read-timeout` of an instance of `comm:socket-stream`.

`(setf stream-read-timeout)` sets the read-timeout of an instance of `comm:socket-stream`.

See also

`socket-stream`
`open-tcp-stream`

---

**stream-start-line-p**

*Generic Function*

Summary

A generic function that returns `t` if the stream is positioned at the beginning of a line.

Package

`stream`

Signature

`stream-start-line-p stream => result`

Arguments

`stream` A stream.

Values

`result` A boolean.

Description

The generic function `stream-start-line-p` returns `t` if `stream` is positioned at the beginning of a line, and `nil` otherwise. It is permissible to define a method that always returns `nil`.

Note that although a value of `0` from `stream-line-column` also indicates the beginning of a line, there are cases where `stream-start-line-p` can be meaningfully implemented and `stream-line-column` 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.

The default method for `stream-start-line-p` on class `fundamental-character-output-stream` uses `stream-line-column`. Therefore, if this is defined to return `nil`, a method
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.
The generic function stream-unread-char undoes the last call to stream-read-char, as in unread-char. Every subclass of fundamental-character-input-stream must define a method for this function.

See also fundamental-character-input-stream

stream-write-buffer

Summary
Writes a part of stream's buffer.

Package
stream

Signature
stream-write-buffer stream buffer start end

Arguments
stream An output stream.
buffer A stream buffer.
start, end Bounding indexes for a subsequence of buffer.

Description
The generic function stream-write-buffer is called by stream-flush-buffer to write the region of the buffer bounded by start and end to the stream's underlying data sink.

stream should be an instance of a subclass of buffered-stream.

This method must be implemented for subclasses of buffered-stream that handle output.

See also buffered-stream stream-flush-buffer
stream-write-byte  

**Generic Function**

Summary  
A generic function used by **write-byte** to write an integer to a binary stream.

Package  
*stream*

Signature  
`stream-write-byte stream integer => result`

Arguments  
- `stream` A stream.
- `integer` An integer.

Values  
- `result` An integer.

Description  
The generic function *stream-write-byte* is used by *write-byte*, and writes the integer `integer` to the binary stream specified by `stream`.

A method must be implemented for all binary subclasses of *buffered-stream* that handle output. A typical implementation will convert the integer to a character using *code-char* and call *stream-write-char*.

A method should be defined for all subclasses of *fundamental-binary-output-stream*.

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  
*stream*
The generic function **stream-write-char** writes character to stream. Every subclass of **fundamental-character-output-stream** must have a method defined for this function.

**See also** **fundamental-character-output-stream**

### **stream-write-sequence**

**Generic Function**

**Summary**

Writes a subsequence of a sequence to a stream.

**Package**

**stream**

**Signature**

stream-write-sequence stream sequence start end => result

**Arguments**

- stream A stream.
- sequence A sequence.
- start An integer.
- end An integer.

**Values**

result A sequence.

**Description**

The generic function **stream-write-sequence** is used by **write-sequence** to write a subsequence of sequence delimited by start and end to stream.

A default method is provided by **fundamental-character-output-stream** that tests each element of sequence in turn, and then uses **stream-write-char** 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 `write-string` to write a string to a character output stream.

**Package**

`stream`

**Signature**

`stream-write-string stream string &optional start end => result`

**Arguments**

- `stream` A stream.
- `string` A string.
- `start` An integer.
- `end` An integer.

**Values**

- `result` A string.

**Description**

The generic function `stream-write-string` is used by `write-string` to write `string` to `stream`. The string can, optionally, be delimited by `start` and `end`.

The default method provided by `fundamental-character-output-stream` uses repeated calls to `stream-write-char`.

See also

`fundamental-character-output-stream`

`stream-write-char`
with-stream-input-buffer

Macro

Summary
Allows access to the input buffer.

Package
stream

Signature
with-stream-input-buffer (buffer index limit) stream &body body => result

Arguments
buffer, index, limit

Variables.

stream An input stream.
body Code.

Values
result The value returned by body.

Description
The macro with-stream-input-buffer allows access to the state of the input buffer for the given buffered stream.

stream should be an instance of a subclass of buffered-stream.

Within the code body, the variables buffer, index and limit are bound to the buffer of stream, its current index and the limit of the buffer. Setting buffer, index or limit will change the values in the stream stream 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.

The buffer is always of type simple-string. The stream-element-type of stream 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 index is greater than or equal to length.
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))
    ;; 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

result The value returned by body.

Description

The macro \texttt{with-stream-output-buffer} allows access to the state of the output buffer for the given buffered stream. 

\textit{stream} should be an instance of a subclass of \texttt{buffered-stream}.

Within the code \textit{body}, the variable names \textit{buffer}, \textit{index} and \textit{limit} are bound to the buffer of \textit{stream}, its current index and the limit of the buffer. Setting \textit{buffer}, \textit{index} or \textit{limit} will change the values in the stream \textit{stream} 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.

The buffers are always of type \texttt{simple-string}. The \texttt{stream-element-type} of \textit{stream} depends on how the stream was constructed.

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 \textit{index} is greater than or equal to \textit{length}.

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
The STREAM Package
This chapter describes symbols available in the `SYSTEM` package.

**apply-with-allocation-in-gen-num**  
*Function*

**Summary**  
Allows control over which generation objects are allocated in, in 64-bit LispWorks.

**Package**  
system

**Signature**  
`apply-with-allocation-in-gen-num what gen-num func &rest args => results`

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gen-num</td>
<td>An integer in the inclusive range [0,7], or nil.</td>
</tr>
<tr>
<td>func</td>
<td>A function designator.</td>
</tr>
<tr>
<td>args</td>
<td>The arguments passed to func.</td>
</tr>
</tbody>
</table>
Values

results  The values returned from the call to \textit{func} with \textit{args}.

Description

The function \texttt{apply-with-allocation-in-gen-num} applies the function \texttt{func} to \texttt{args} such that objects of allocation type \texttt{what} are allocated in generation \texttt{gen-num}, in 64-bit LispWorks. See also the keyword \texttt{:allocation} to \texttt{make-array}, 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 \texttt{*symbol-alloc-gen-num*}.

\textbf{Note:} In 32-bit LispWorks the argument \texttt{what} is ignored and the effect is like that of the macro \texttt{allocation-in-gen-num}.

See also

\begin{itemize}
  \item \texttt{allocation-in-gen-num}
  \item \texttt{make-array}
  \item \texttt{*symbol-alloc-gen-num*}
\end{itemize}

\textbf{augmented-string}

\textit{Type}

Summary  The augmented string type.

Package  \texttt{system}

Signature  \texttt{augmented-string length}

Arguments  \texttt{length}  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 object => bool

Arguments  
  object  The object to be tested.

Values  
  bool  
    t if object 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 command &key current-directory wait shell-type => status

Arguments  
  command  A string, a list of strings, a simple-vector of strings, or nil.

  current-directory  A string. Implemented only on Microsoft Windows.

  wait  A boolean.

  shell-type  A string or nil.
### Values

| status | The exit status of the invoked shell or process. |

### Description

`call-system` 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 `*terminal-io*` in LispWorks.)

If `command` 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 `shell-type` as described below.

If `command` 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 argv[0] in C, and so on). If `command` 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 argv[0] in C, and so on). If `command` is `nil`, then the shell is run.

On Microsoft Windows `current-directory` is the `lpCurrentDirectory` argument passed to `CreateProcess`. If this is not supplied, the `pathname-location` of the `current-pathname` is passed.

If `wait` is true, `call-system` does not return until the process has exited. The default for `wait` is `t`.

On Unix/Linux/Mac OS X/FreeBSD, if `shell-type` is a string it specifies the shell. If `shell-type` is `nil` (the default) then the Bourne shell, `/bin/sh`, is used. The C shell may be obtained by passing `"/bin/csh"`.

On Microsoft Windows if `shell-type` is `nil` then `cmd.exe` is used on Windows Vista, Windows XP and Windows 2000 and `command.com` on Windows 98 and Windows ME.

`call-system` 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.
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.
### call-system-showing-output

*call-system-showing-output* is an extension to *call-system* 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. *call-system-showing-output* therefore allows the user to, for example, invoke a shell command and redirect the output to the current Listener window.

The argument `command` is interpreted as by `call-system`. `prefix` is a prefix to be printed at the start of any output line. The default value is `"; "`. `show-cmd` specifies whether or not the `cmd` invoked will be printed as well as the output for that command. If `t` then `cmd` will be printed. The default value for `show-cmd` is `t`. `output-stream` specifies where the output will be sent to. The default value is `*standard-output*`. If `wait` is true, *call-system-showing-output* does not return until the process has exited. If `nil`, *call-system-showing-output* returns immediately and no output is shown. The default for `wait` is `t`.

**Values**

- **status**: The exit status of the invoked shell or process.

**Description**

- `prefix`: A string.
- `show-cmd`: A boolean.
- `output-stream`: A symbol.
- `wait`: A boolean.
- `shell-type`: A string. Supported only on Unix/Linux/Mac OS X.
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

CL-USER 2 > (sys:call-system-showing-output "pwd" :prefix "&&&" :show-cmd nil)

&&/amd/xanfs1-cam/u/ldisk/sp/lispsrc/v42/builds

**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**  
*Function*

**Summary**  
A generalized reference for alist elements.

**Package**  
`system`

**Signature**  
`cdr-assoc item alist &key test test-not key => result`

```lisp
(setf cdr-assoc) value item alist => value
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>item</code></td>
<td>An object.</td>
</tr>
<tr>
<td><code>alist</code></td>
<td>An association list.</td>
</tr>
<tr>
<td><code>test</code></td>
<td>A function designator.</td>
</tr>
<tr>
<td><code>test-not</code></td>
<td>A function designator.</td>
</tr>
<tr>
<td><code>key</code></td>
<td>A function designator.</td>
</tr>
<tr>
<td><code>value</code></td>
<td>An object.</td>
</tr>
</tbody>
</table>

**Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>result</code></td>
<td>An object (from <code>alist</code>) or <code>nil</code>.</td>
</tr>
</tbody>
</table>

**Description**

The functions `cdr-assoc` and `(setf cdr-assoc)` provide a generalized reference for elements in an association list. The arguments are all as specified for the Common Lisp function `assoc`. `cdr-assoc` and `(setf cdr-assoc)` read and write the `cdr` of an element in a manner consistent with the Common Lisp notion of places.

`cdr-assoc` returns the `cdr` of the first cons in the alist `alist` that satisfies the test, or `nil` if no element of `alist` matches.

`(setf cdr-assoc)` modifies the first cons in `alist` that satisfies the test, setting its `cdr` to `value`. If no element of `alist` matches, then `(setf cdr-assoc)` constructs a new cons `(cons item value)` and inserts it in the head of `alist`. 
Example

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))

*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 key-file 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.
errorp  A boolean.

Values  gspec  A Gesture Spec object

Description  The function coerce-to-gesture-spec returns a Gesture Spec object gspec which can be used to represent the key-stroke indicated by object.

If object is a Lisp character, then gspec’s data is one of the known Gesture Spec keywords, or its char-code, and gspec’s modifiers contains its char-bits attribute mapped onto the values gesture-spec-control-bit etc.

If object is a keyword, then it must be one of the known Gesture Spec keywords and becomes gspec’s data. gspec’s modifiers is 0.

If object is a string, then coerce-to-gesture-spec 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 name-char or the name of one of the known Gesture Spec keywords. Then gspec contains the corresponding Gesture Spec keyword or char-code in its data, and the modifier keys are represented in its modifiers.

If object is a Gesture Spec object, it is simply returned.

coerce-to-gesture-spec does not create wild gesture specs.
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  

Function

Summary
Copies uses preferences.

Package
system

Signature
copy-preferences-from-older-version old-path new-path
&optional flag-name

Arguments
old-path A preference path.
new-path A preference path.
flag-name A string.

Description

The function `copy-preferences-from-older-version` copies uses preferences from one part of the registry to another.

`old-path` and `new-path` are the paths of preferences for the old and the new version, corresponding to the paths that were passed to `(setf product-registry-path)`.

`flag-name` is a name of the flag to use to record in the registry that the copy is already done. `flag-name` must be a valid registry value name on Microsoft Windows, and a valid filename on all other platforms. The default value of `flag-name` is the string "copied-old-preferences".

`copy-preferences-from-older-version` performs several checks:

1. It checks if it already copied to `new-path` in the current session, and if so does nothing.
2. It checks if the `flag-name` entry exists, and if so it does nothing.
3. It checks if another call to `copy-preferences-from-older-version` is already executing (in another thread), and if so it just waits for the other call to finish.

Then if all the checks above indicate that copying is still needed, `copy-preferences-from-older-version` copies the values from the tree below `old-path` to a tree below `new-path`. It traverses the entire tree below `old-path`, and checks each key to see if it has any values.

For a key that has values, it checks if the key exists under `new-path`, 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 `new-path`, it creates the key and copies the values.
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`

### count-gen-num-allocation  
**Function**

**Summary**
Returns the amount of allocated data in a generation in 64-bit LispWorks.

**Package**
`system`

**Signature**
`count-gen-num-allocation gen-num &optional include-lower-generations`

**Arguments**
- `gen-num`  
  An integer between 0 and 7, inclusive.
- `include-lower-generations`  
  A generalized boolean.

**Values**
- `allocation`  
  An integer.

**Description**
The function `count-gen-num-allocation` returns the amount of allocated data in generation `gen-num`. If `include-lower-generations` is non-nil, the returned value `allocation` 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 `room-values` instead.

See also `room-values`
**default-eol-style**

*Function*

- **Summary**: Provides a default end of line style for a file.
- **Package**: system
- **Signature**: `default-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**:
  - `new-ef-spec`: A new external format spec created by merging `ef-spec` with the encoding that was found.
- **Description**: Merge `ef-spec` with `(:default :eol-style :crlf)` on Microsoft Windows, `(:default :eol-style :lf)` on UNIX/Linux/Mac OS X. This is usually used as the last function on its list.
- **See also**:
  - `*file-eol-style-detection-algorithm*`

**default-stack-group-list-length**

*Variable*

- **Summary**: The size of the stack cache.
- **Package**: system
- **Initial Value**: 10
- **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 \texttt{*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 \texttt{(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 \texttt{mark-and-sweep}

\textbf{define-top-loop-command} \hspace{1cm} \textit{Macro}

\textbf{Summary} \hspace{1cm} Defines a top level loop command.

\textbf{Package} \hspace{1cm} \texttt{system}

\textbf{Signature} \hspace{1cm} \texttt{define-top-loop-command name-and-options lambda-list form*}

\texttt{name-and-options ::= name}

| \hspace{1cm} (name option*)

\texttt{option ::= (:aliases alias*)}

| \hspace{1cm} (:result-type result-type)

\textbf{Arguments} \hspace{1cm} \texttt{name} \hspace{1cm} A keyword naming the command.

\texttt{alias} \hspace{1cm} A keyword naming an alias for the command.

\texttt{lambda-list} \hspace{1cm} A destructuring lambda list.

\texttt{result-type} \hspace{1cm} One of the symbols \texttt{values, eval} and \texttt{nil}.
Description

The macro `define-top-loop-command` defines a top level loop command called `name` which takes the parameters specified by `lambda-list`. If `&whole` is used in `lambda-list` 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 `lambda-list` otherwise.

If any aliases are specified in `option`, these keywords will also invoke the command.

When the command is used, each form is evaluated in sequence with the variables from `lambda-list` bound to the subsequent forms on the command line.

If `result-type` is `values` (the default), then the values of the last form will be returned to the top level loop.

If `result-type` is `eval`, 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.

If `result-type` is `nil`, then the last form should return two values. If the second value is `nil` then the first value is treated as a list of values to returned to the top level loop. If the second value is non-`nil` 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.

Note: for details of pre-defined top level loop commands, enter `:?` at the Listener prompt.

Example

Given this definition:

```lisp
(define-top-loop-command (:lave (:result-type eval)) (form)
  (reverse form))
```

then the command line

`:lave (1 2 list)`

will evaluate the form `(list 2 1)`. 
Here are definitions for two commands both of which will run `apropos`:

```lisp
(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

```lisp
(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**

```lisp
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  

new-ef-spec  
A new external format spec created by merging ef-spec with the encoding that was found.

Description  

When the encoding in ef-spec has foreign type (unsigned-byte 8), search buffer up to length for the first occurrence of the byte (10). If found, and it is preceded in buffer by (13), merge ef-spec with

(\:default :eol-style :crf)
If found and is not preceded by (13), merge ef-spec with

(\:default :eol-style :lf)
Thus a complete external format spec is constructed. Otherwise, return ef-spec.

When the encoding in ef-spec has foreign type (unsigned-byte 16), search buffer up to length for the first occurrence of the byte sequence (13 0 10). If found, merge ef-spec with

(\:default :eol-style :crf)
If (13 0 10) is not found, search buffer up to length for (10 0) or (0 10). If found, merge ef-spec with

(\:default :eol-style :lf)
Thus a complete external format spec is constructed. Otherwise, return ef-spec.

See also  

*file-eol-style-detection-algorithm*

**detect-japanese-encoding-in-file**  

*Function*

Summary  

Determines which type of Japanese encoding is used in a buffer.

Package  

system
**Signature**

```
detect-japanese-encoding-in-file 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**

- `new-ef-spec`  A new external format spec created by merging `ef-spec` with the Japanese encoding that was found.

**Description**

Assume the encoding is one of `jis`, `sjis`, `euc`, `unicode` and `ascii`, and try to determine which of these it is, by looking for distinctive byte sequences in `buffer` up to `length`. If found, merge `ef-spec` with that encoding.

**See also**

*file-encoding-detection-algorithm*

---

**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**

`system`

**Signature**

```
detect-unicode-bom 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**

`new-ef-spec`  
A new external format spec created by merging `ef-spec` with the encoding that was found.

**Description**

When called as part of `open`’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` (:unicode :little-endian nil).

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` (:unicode :little-endian t).

**See also**  
*file-encoding-detection-algorithm*  

---

**`*directory-link-transparency*`**  

**Variable**

**Summary**

Controls whether `directory` returns truenames on Unix-like systems.

**Package**

`system`

**Initial Value**

`t` on Unix-like systems, `nil` on Microsoft Windows.

**Description**

In line with the ANSI Common Lisp standard, `directory` returns truenames by default.

Setting `*directory-link-transparency*` to `nil` allows you to get the old behavior of `directory`, whereby soft links are not resolved in the pathnames returned.

`*directory-link-transparency*` is the default value of the `link-transparency` argument to `directory`.

**See also**

`directory`
*extended-spaces*

Variable

Summary
Extends the notion of space to include more than just the space character.

Package
system

Initial value
nil

Description
When this variable is true, the concept of “space” is extended from just \Space to include other appropriate characters. The default is nil, for ANS compliance, but we recommend that you set it to t.

This variable controls how the format directives ~:C and ~:@C output graphic characters which have an empty glyph. When this variable is t, all such characters are output using the name:

(format nil "~:C" \No-break-space) -> "No-Break-Space"
(format nil "~:C" (code-char #x3000)) -> "Ideographic-Space"

When false, only one such character is output using the name:

(format nil "~:C" \Space) -> "Space"
(format nil "~:C" \No-break-space) -> " "
(format nil "~:C" (code-char #x3000)) -> " "

It also affects whitespace-char-p.

See also
extended-character-p

*file-encoding-detection-algorithm*

Variable

Summary
List of functions to call to work out an encoding.

Package
system
The SYSTEM Package

Initial value

\begin{verbatim}
(find-filename-pattern-encoding-match
 find-encoding-option
 detect-unicode-bom
 locale-file-encoding)
\end{verbatim}

Description

Functions on this list take four arguments—the pathname of the file; an external format spec; a vector of element-type \(\text{(unsigned-byte 8)}\) 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 :output. They return an external format spec, which normally is either \text{ef-spec} unmodified, or the result of merging \text{ef-spec} with another external format spec via \text{merge-ef-specs}.

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:

\begin{verbatim}
(find-encoding-option locale-file-encoding)
\end{verbatim}

See also

\begin{verbatim}
find-filename-pattern-encoding-match
 find-encoding-option
 detect-unicode-bom
 detect-japanese-encoding-in-file
 guess-external-format
 locale-file-encoding
\end{verbatim}

file-encoding-resolution-error

Condition

Summary

An error type to signal when an external file format cannot be deduced.

Package

\text{system}

Superclasses

\text{error}

Initargs

\begin{verbatim}
:ef-spec
\end{verbatim} \hspace{1em} An external format specification.
An error type signalled when `open`, `load` or `compile-file` fail to detect an external format to use.

The `ef-spec` slot contains the incomplete external format specification argument constructed by `guess-external-format`.

See also `guess-external-format`

### *file-eol-style-detection-algorithm* Variable

**Summary** List of functions for determining the end of line style of a file.

**Package** `system`

**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** `(detect-eol-style default-eol-style)`

**See also**

- `detect-eol-style`
- `default-eol-style`
- `guess-external-format`

### *filename-pattern-encoding-matches* Variable

**Summary** An association of filename patterns to external format specs.

**Package** `system`

**Initial value** `(("TAGS" . (:latin-1 :eol-style :lf)))`

**Description** An alist of filename patterns to external format specs.
**find-encoding-option**

*Function*

**Summary**
Examines a buffer for an encoding option.

**Package**
*system*

**Signature**
find-encoding-option pathname ef-spec buffer length => result

**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**
- *result*
  The result of reading the value returned from the encoding or external-format option as a Lisp expression in the keyword package.

**Description**
Looks in the file options (EMACS-style -*- line) for an option called encoding or external-format, with value value. If found, it reads value as a Lisp expression in the keyword package and merges ef-spec with value and returns the result as result. Thus it does not override a supplied ef-spec.

**See also**
*file-encoding-detection-algorithm*

**find-filename-pattern-encoding-match**

*Function*

**Summary**
Finds the encoding of a file based on the filename.
find-filename-pattern-encoding-match  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

new-ef-spec  An external format spec.

Description

Compares pathname (using pathname-match-p) with elements of *filename-pattern-encoding-matches*.

If a match is found, merges ef-spec with the corresponding external format spec and returns the result as new-ef-spec. Thus it does not override a supplied ef-spec.

See also

*file-encoding-detection-algorithm*
*filename-pattern-encoding-matches*

---

gen-num-segments-fragmentation-state  Function

Summary

Shows the fragmentation state in a generation in 64-bit Lisp-Works.

Package

system

Signature

gen-num-segments-fragmentation-state  gen-num  &optional
statics-too  =>  fragmentation-state

Arguments

gen-num  A number.

statics-too  A generalized boolean?
The function `gen-num-segments-fragmentation-state` shows the fragmentation state in a generation in 64-bit LispWorks. `gen-num-segments-fragmentation-state` returns a list, where each element is a sub-list showing the fragmentation state in a segment. The sub-list is of the form `(allocation-type allocated free)` where `allocation-type` is the allocation type of the segment, `allocated` is the amount of allocated data in the segment, and `free` is the total size of free areas in the segment that cannot be easily used.

The ratio `free/allocated` 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 `set-blocking-gen-num` and `marking-gc`).

Allocation types :cons-static, :non-pointer-static, :mixed-static, :other-big and :non-pointer-big are included in the result only if `statics-too` is non-nil. The default value of `statics-too` is nil.

Note: The implementation of `set-blocking-gen-num` is intended to solve any fragmentation issues automatically.

Note: `gen-num-segments-fragmentation-state` is implemented only in 64-bit LispWorks. It is not relevant to the Memory Management API in 32-bit implementations, where `check-fragmentation` is available instead.

See also `check-fragmentation`
`marking-gc`
`set-blocking-gen-num`
**gesture-spec-accelerator-bit**  
*Constant*

**Summary**  
Used in the representation of a keystroke with the accelerator key.

**Package**  
*system*

**Description**  
The constant `gesture-spec-accelerator-bit` is used to represent the accelerator key in a Gesture Spec object.


**See also**  
coerce-to-gesture-spec  
gesture-spec-modifiers  
make-gesture-spec

---

**gesture-spec-control-bit**  
*Constant*

**Summary**  
Used in the representation of a keystroke with the Control key.

**Package**  
*system*

**Description**  
The constant `gesture-spec-control-bit` is used to represent the Control modifier key in a Gesture Spec object.


**See also**  
coerce-to-gesture-spec  
gesture-spec-modifiers  
make-gesture-spec
**gesture-spec-data**

*Function*

**Summary**
Returns the key in a Gesture Spec object.

**Package**
*system*

**Signature**
`gesture-spec-data gspec => data`

**Arguments**
`gspec` A Gesture Spec object

**Values**
`data` A non-negative integer or a keyword.

**Description**
The function `gesture-spec-data` returns an integer or keyword representing the key in the Gesture Spec object `gspec`. When `data` is an integer, it is a non-negative integer less than `char-code-limit`, and `gspec` represents a keystroke with the key indicated by the character which is the value of `(code-char data)`.

`data` can also be a keyword such as `:f6`, when `gspec` represents a keystroke with `F6` pressed.

**See also**
gesture-spec-modifiers
make-gesture-spec

---

**gesture-spec-hyper-bit**

*Constant*

**Summary**
Used in the representation of a keystroke with the `Hyper` key.

**Package**
*system*

**Description**
The constant `gesture-spec-hyper-bit` is used to represent the `Hyper` modifier key in a Gesture Spec object.

**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 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
- `gesture-spec-accelerator-bit`
- `gesture-spec-control-bit`
- `gesture-spec-data`
- `gesture-spec-meta-bit`
- `gesture-spec-hyper-bit`
- `gesture-spec-shift-bit`
- `gesture-spec-super-bit`
- `make-gesture-spec`

---

**gesture-spec-p**

Function

Summary
The predicate for Gesture Spec objects.

Package
`system`

Signature
`gesture-spec-p object => result`

Arguments
- `object` A Lisp object

Values
- `result` A boolean.

Description
The function `gesture-spec-p` is the predicate for whether the object `object` is a Gesture Spec object.

See also
- `coerce-to-gesture-spec`
- `make-gesture-spec`
**gesture-spec-shift-bit**

*Constant*

**Summary**

Used in the representation of a keystroke with the `Shift` key.

**Package**

`system`

**Description**

The constant `gesture-spec-shift-bit` is used to represent the `Shift` modifier key in a Gesture Spec object.

Note that you may not construct a Gesture Spec with a `both-case-p` character represented in the `data` and with `modifiers` equal to `gesture-spec-shift-bit`. See `make-gesture-spec` for details and examples.


See also

- `coerce-to-gesture-spec`
- `gesture-spec-modifiers`
- `make-gesture-spec`

---

**gesture-spec-super-bit**

*Constant*

**Summary**

Used in the representation of a keystroke with the `Super` key.

**Package**

`system`

**Description**

The constant `gesture-spec-super-bit` is used to represent the `Super` modifier key in a Gesture Spec object.


See also

- `coerce-to-gesture-spec`
- `gesture-spec-modifiers`
- `make-gesture-spec`
**gesture-spec-to-character**  
*Function*

**Summary**  
Returns the character corresponding to a Gesture Spec object.

**Package**  
`system`

**Signature**  
`gesture-spec-to-character gspec => char`

**Arguments**  
`gspec`  
A Gesture Spec object

**Values**  
`char`  
A Lisp character.

**Description**  
The function `gesture-spec-to-character` returns the Lisp character object corresponding to the Gesture Spec object `gspec`. 

Modifier bits in `gspec` are mapped to Lisp character bits attributes where possible. `gesture-spec-accelerator-bit` is ignored.

**See also**  
`coerce-to-gesture-spec`

`make-gesture-spec`

---

**get-file-stat**  
*Function*

**Summary**  
Provides read access to the C stat structure which describes files.

**Note:** not applicable on Microsoft Windows.

**Package**  
`system`

**Signature**  
`get-file-stat filename-or-fd => file-stat (errno)`

**Arguments**  
`filename-or-fd`  
A string denoting a file, or a file descriptor.
Values

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file-stat</td>
<td>On success, an object representing the stat values. On failure, nil is returned together with a second value.</td>
</tr>
<tr>
<td>errno</td>
<td>Indicates the errno value returned by the system call. This second value is returned only in the case of failure.</td>
</tr>
</tbody>
</table>

Description

_file-stat_ is an object representing the stat values, as would be returned by the system call _stat_ (for a filename) or the system call _fstat_ (for an fd).

The values in _file-stat_ are the raw data, and it is the responsibility of the user to interpret them when needed. See the UNIX manual entry for _stat_ for details.

The values can be read from _file-stat_ by these readers:

- **sys:file-stat_inode**
  The inode of the file.

- **sys:file-stat-device**
  The id of the device where the file is.

- **sys:file-stat-owner-id**
  The user id of the owner of the file.

- **sys:file-stat-group-id**
  The group id of the file’s group.

- **sys:file-stat-size**
  The size of the file in bytes.

- **sys:file-stat-blocks**
  The number of 512-bytes blocks used by the file.

- **sys:file-stat-mode**
  The protection value of the file.

- **sys:file-stat-last-access**

- **sys:file-stat-last-change**
**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).

---

### get-folder-path

**Function**

**Summary**

Gets the path of a special folder on a Microsoft Windows or Mac OS X machine.

**Package**

`system`

**Signature**

`get-folder-path what &key create => result`

**Arguments**

- `what` A keyword.
- `create` A boolean.

**Values**

- `result` A directory pathname naming the path, or `nil`.

**Description**

The function `get-folder-path` 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. `get-folder-path` is implemented only on Microsoft Windows and Mac OS X.

`what` indicates the purpose of the special folder. For instance, `:common-appdata` means the folder containing application data for all users.
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

Function

Summary

Gets the root of the user’s profile on a Windows NT-based system.

Package

system

Signature

get-user-profile-directory => result

Values

result

A directory pathname naming the path, or nil.

Description

The function get-user-profile-directory obtains the path to the current user’s profile folder on a Windows NT-based system (including Windows 2000, Windows XP and Windows Vista). get-user-profile-directory is implemented only on Microsoft Windows.

result names the root of the profile directory.

Note that the default path for each user’s profile may differ between versions of the operating system.

Compatibility note

In LispWorks 5.0 and previous versions, get-user-profile-directory returns a string.

Example

On Windows XP:

(sys:get-user-profile-directory)
=>
#P"C:/Documents and Settings/fred/"

On Windows 98 SE:

(sys:get-user-profile-directory)
=>
nil

See also

get-folder-path
**guess-external-format**

**Function**

**Summary**
Tries to work out the external format

**Package**
*system*

**Signature**

`guess-external-format pathname ef-spec buffer length => 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**
- `ef-spec` An external format spec.

**Description**

If `ef-spec` is complete, then it is returned. Otherwise `guess-external-format` calls, in turn, functions on the list `*file-encoding-detection-algorithm*`. 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 `:default`, an error of type `file-encoding-resolution-error` is signalled. The caller offers a restart for trying again with respecified `external-format` and/or `element-type` arguments. Otherwise `guess-external-format` proceeds to guess the `eol-style`.

To guess the `eol-style`, functions on the list `*file-eol-style-detection-algorithm*` 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 `:eol-style`, an error of type `file-encoding-resolution-error` is signalled.
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  
staticp

**int32**  

*Type*

**Summary** A type used to generate optimal 32-bit arithmetic code.
Package system

Signature int32

Description The type int32 is used to generate optimal 32-bit arithmetic code. Objects of type int32 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 int32 objects to produce efficient raw 32-bit code.

See the section "Fast 32-bit arithmetic" in the LispWorks User Guide for more information.

See also int32*
int32+
int32-
+int32-0+
+int32-1+
int32-1+
int32-1-
int32/
int32/= int32<
int32<=
int32=
int32>
int32>=
int32-aref
int32-logand
int32-logandc1
int32-logandc2
int32-logeqv
int32-logior
int32-lognand
int32-lognor
int32-lognot
### int32

#### Function

**Summary**  The multiply operator for **int32** objects.

**Package**  system

**Signature**  

\[
\text{int32}^* \quad x \quad y \Rightarrow \text{int32}
\]

**Arguments**

- \( x \)  An **int32** object or an integer of type \((\text{signed-byte} \ 32)\).
- \( y \)  An **int32** object or an integer of type \((\text{signed-byte} \ 32)\).

**Values**  **int32**  An **int32** object.

**Description**  The function **int32**\(^*\) is the multiply 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**

---

\[[16] The SYSTEM Package\]

- int32-logorcl
- int32-logorc2
- int32-logxor
- int32-minusp
- int32-plusp
- int32-to-integer
- int32-zerop
- integer-to-int32
- make-simple-int32-vector
- simple-int32-vector
**Function**

### int32+

**Summary**
The add operator for int32 objects.

**Package**
`system`

**Signature**
`int32+ 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 int32+ is the add 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-

**Summary**
The subtract operator for int32 objects.

**Package**
`system`

**Signature**
`int32- 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.
The function \texttt{int32-} is the subtract operator for \texttt{int32} objects.

See the section "Fast 32-bit arithmetic" in the \textit{LispWorks User Guide} for more information about the INT32 API.

See also \texttt{int32}

\texttt{+int32-0+} \quad \textit{Symbol Macro}

Summary \quad Shorthand for \texttt{(sys:integer-to-int32 0)}.

Package \quad \texttt{system}

Description \quad The symbol macro \texttt{+int32-0+} expands to \texttt{(sys:integer-to-int32 0)}.

See also \texttt{integer-to-int32}

\texttt{+int32-1+} \quad \textit{Symbol Macro}

Summary \quad Shorthand for \texttt{(sys:integer-to-int32 1)}.

Package \quad \texttt{system}

Description \quad The symbol macro \texttt{+int32-1+} expands to \texttt{(sys:integer-to-int32 1)}.

See also \texttt{integer-to-int32}

\texttt{+int32-1+} \quad \textit{Function}

Summary \quad The operator for \texttt{int32} objects corresponding to the function 1+.
Package system
Signature `int32-1+ x => int32`
Arguments `x` An `int32` object or an integer of type `(signed-byte 32)`.
Values `int32` An `int32` object.
Description The function `int32-1+` is the operator for `int32` objects that corresponds to the function `1+`.
See the section "Fast 32-bit arithmetic" in the *LispWorks User Guide* for more information about the INT32 API.
See also `int32`

`int32-1-` Function

Summary The operator for `int32` objects corresponding to the function `1-`.
Package system
Signature `int32-1- x => int32`
Arguments `x` An `int32` object or an integer of type `(signed-byte 32)`.
Values `int32` An `int32` object.
Description The function `int32-1-` is the operator for `int32` objects that corresponds to the function `1-`.
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  The divide operator for int32 objects.

Package  system

Signature  int32/ x y => int32

Arguments  x  An int32 object or an integer of type (signed-byte 32).

y  An int32 object that does not correspond to 0, or a non-zero integer of type (signed-byte 32).

Values  int32  An int32 object.

Description  The function int32/ is the divide operator for int32 objects.

See also  int32

int32/=  

Function

Summary  The /= comparison for int32 objects.

Package  system

Signature  int32/= x y => result

Arguments  x  An int32 object or an integer of type (signed-byte 32).
An int32 object or an integer of type (signed-byte 32).

result A boolean.

The function int32/= is the not equal comparison 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

Function

Summary The < comparison 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).

result A boolean.

The function int32< is the less than comparison 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<<

**Function**

**Summary**
A shift left 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 left 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<=

**Function**

**Summary**
The `<=` comparison 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` A boolean.
### Description
The function `int32<=` is the less than or equal comparison 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=

**Function**

**Summary**  
The `=` comparison 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`  
A boolean.

**Description**  
The function `int32=` is the equal comparison 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>

**Function**

**Summary**  
The `>` comparison for `int32` objects.
<table>
<thead>
<tr>
<th>Package</th>
<th><code>system</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Signature</td>
<td><code>int32&gt; x y =&gt; result</code></td>
</tr>
</tbody>
</table>
| 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` A boolean. |
| Description | The function `int32>` is the greater than comparison 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>=**  
*Function*

| Summary | The `>=` comparison 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` A boolean. |
| Description | The function `int32>=` is the greater than or equal comparison 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>>

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 LispWorks User Guide for more information about the INT32 API.

See also int32 simple-int32-vector

int32-logand Function

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 `int32-logand` is the bitwise logical 'and' 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-logandc1

**Function**

**Summary**
The `logandc1` operator for `int32` objects.

**Package**
`system`

**Signature**
`int32-logandc1 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 `int32-logandc1` is the bitwise logical operator for `int32` objects which 'ands' the complement of `x` with `y`.
See the section "Fast 32-bit arithmetic" in the *LispWorks User Guide* for more information about the INT32 API.

### See also
`int32`

### int32-logandc2

**Function**

**Summary**
The `logandc2` operator for `int32` objects.

687
The SYSTEM Package

Package | system
Signature | int32-logandc2 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 int32-logandc2 is the bitwise logical operator for int32 objects which ‘ands’ x with the complement of y.
            | See the section “Fast 32-bit arithmetic” in the LispWorks User Guide for more information about the INT32 API.
See also | int32

int32-logbitp

Function

Summary | The logbitp operator for int32 objects.
Package | system
Signature | int32-logbitp index x => result
Arguments | index | An int32 object or an integer of type (signed-byte 32).
         | x | An int32 object or an integer of type (signed-byte 32).
Values | result | An boolean.
### Description

The function `int32-logbitp` is the test for `int32` objects which returns `t` if if the bit at index `index` in `x` is 1, and `nil` if it is 0.

See the section "Fast 32-bit arithmetic" in the *LispWorks User Guide* for more information about the INT32 API.

#### See also

`int32`

### int32-logeqv

#### Function

**Summary**

The `logeqv` operator for `int32` objects.

**Package**

`system`

**Signature**

`int32-logeqv 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 `int32-logeqv` is the bitwise logical operator for `int32` objects which returns the complement of the ‘exclusive or’ of `x` and `y`.

See the section "Fast 32-bit arithmetic" in the *LispWorks User Guide* for more information about the INT32 API.

#### See also

`int32`
### int32-logior

**Summary**
The `logior` operator for `int32` objects.

**Package**
`system`

**Signature**
`int32-logior 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 `int32-logior` is the bitwise logical ‘inclusive or’ 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-lognand

**Summary**
The `lognand` operator for `int32` objects.

**Package**
`system`

**Signature**
`int32-lognand 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).`
The function `int32-lognand` is the bitwise logical operator for `int32` objects which returns the complement of the 'and' of `x` and `y`.

See the section "Fast 32-bit arithmetic" in the *LispWorks User Guide* for more information about the INT32 API.

**int32-lognor**

Function

Summary

The `lognor` operator for `int32` objects.

Package

`system`

Signature

`int32-lognor 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 `int32-lognor` is the bitwise logical operator for `int32` objects which returns the complement of the 'inclusive or' of `x` and `y`.

See the section "Fast 32-bit arithmetic" in the *LispWorks User Guide* for more information about the INT32 API.

See also

`int32`
**int32-lognot**  
*Function*

**Summary**  
The \texttt{lognot} operator for an \texttt{int32} object.

**Package**  
\texttt{system}

**Signature**  
\texttt{int32-lognot} \( x \Rightarrow \texttt{int32} \)

**Arguments**  
\( x \)  
An \texttt{int32} object or an integer of type \texttt{(signed-byte 32)}.

**Values**  
\texttt{int32}  
An \texttt{int32} object.

**Description**  
The function \texttt{int32-lognot} is the bitwise logical operator for \texttt{int32} objects which returns the complement of its argument \( x \).

See the section “Fast 32-bit arithmetic” in the \textit{LispWorks User Guide} for more information about the INT32 API.

See also  
\texttt{int32}

**int32-logorc1**  
*Function*

**Summary**  
The \texttt{logorc1} operator for \texttt{int32} objects.

**Package**  
\texttt{system}

**Signature**  
\texttt{int32-logorc1} \( x \ y \Rightarrow \texttt{int32} \)

**Arguments**  
\( x \)  
An \texttt{int32} object or an integer of type \texttt{(signed-byte 32)}.

\( y \)  
An \texttt{int32} object or an integer of type \texttt{(signed-byte 32)}.

**Values**  
\texttt{int32}  
An \texttt{int32} object.
The function `int32-logorc1` is the bitwise logical operator for `int32` objects which 'inclusive ors' the complement of `x` with `y`.

See the section "Fast 32-bit arithmetic" in the *LispWorks User Guide* for more information about the INT32 API.

See also `int32`
Function

**int32-logtest**

**Summary**
The `logtest` operator for `int32` objects.

**Package**
`system`

**Signature**
`int32-logtest 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 boolean.

**Description**
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.

See the section "Fast 32-bit arithmetic" in the *LispWorks User Guide* for more information about the INT32 API.

**See also**
`int32`

Function

**int32-logxor**

**Summary**
The `logxor` operator for `int32` objects.

**Package**
`system`

**Signature**
`int32-logxor 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).`
### int32-logxor

**Description**
The function `int32-logxor` is the bitwise logical 'exclusive or' 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-minusp

**Function**

The `minusp` test for an `int32` object.

**Summary**
The `minusp` test for an `int32` object.

**Package**
`system`

**Signature**
`int32-minusp x => result`

**Arguments**
- `x`: An `int32` object or an integer of type `(signed-byte 32)``.

**Values**
- `result`: A boolean.

**Description**
The function `int32-minusp` tests whether its argument `x` is `int32<` than 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`

### int32-plusp

**Function**

The `plusp` test for an `int32` object.

**Summary**
The `plusp` test for an `int32` object.

**Package**
`system`

---

Values

| An `int32` object. |

Description

The function `int32-logxor` is the bitwise logical 'exclusive or' 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`
The function \texttt{int32-plusp} tests whether its argument \texttt{x} is greater than the value of \texttt{+int32-0+}. See the section "Fast 32-bit arithmetic" in the \textit{LispWorks User Guide} for more information about the INT32 API.

See also \texttt{int32}

\textbf{int32-to-integer} \quad \textit{Function}

The destructor converting an \texttt{int32} object to an integer.

\textbf{Signature} \quad \texttt{int32-to-integer int32 \Rightarrow integer}

\textbf{Arguments} \quad \texttt{int32} \quad An \texttt{int32} object or an integer of type \texttt{(signed-byte 32)}.

\textbf{Values} \quad \texttt{integer} \quad An integer of type \texttt{(signed-byte 32)}.

\textbf{Description} \quad The function \texttt{int32-to-integer} returns an integer \texttt{integer} of type \texttt{(signed-byte 32)} corresponding to the \texttt{int32} object \texttt{int32}. The argument \texttt{int32} can also be an integer of type \texttt{(signed-byte 32)}, in which case it is simply returned.

An error is signalled if \texttt{int32} is not of type \texttt{int32} or \texttt{(signed-byte 32)}.

See the section "Fast 32-bit arithmetic" in the \textit{LispWorks User Guide} for more information about the INT32 API.
See also \texttt{int32}

\textbf{int32-zerop}

\textit{Function}

\textbf{Summary} \hspace{1cm} The \texttt{zerop} test for an \texttt{int32} object.

\textbf{Package} \hspace{1cm} system

\textbf{Signature} \hspace{1cm} \texttt{int32-zerop} \hspace{0.5cm} x \Rightarrow \texttt{result}

\textbf{Arguments} \hspace{1cm} \texttt{x} \hspace{1cm} An \texttt{int32} object or an integer of type \texttt{(signed-byte 32)}.

\textbf{Values} \hspace{1cm} \texttt{result} \hspace{1cm} A boolean.

\textbf{Description} \hspace{1cm} The function \texttt{int32-zerop} tests whether its argument \texttt{x} is \texttt{int32=} to the value of \texttt{+int32-0+}.

See the section "Fast 32-bit arithmetic" in the \textit{LispWorks User Guide} for more information about the INT32 API.

See also \texttt{int32}

\textbf{integer-to-int32}

\textit{Function}

\textbf{Summary} \hspace{1cm} The constructor for \texttt{int32} objects.

\textbf{Package} \hspace{1cm} system

\textbf{Signature} \hspace{1cm} \texttt{integer-to-int32} \hspace{0.5cm} \texttt{integer} \Rightarrow \texttt{int32}

\textbf{Arguments} \hspace{1cm} \texttt{integer} \hspace{1cm} An integer of type \texttt{(signed-byte 32)}.

\textbf{Values} \hspace{1cm} \texttt{int32} \hspace{1cm} An \texttt{int32} object.
The function integer-to-int32 constructs an int32 object from an integer. An error is signalled if integer is not of type (signed-byte 32).

See the section "Fast 32-bit arithmetic" in the LispWorks User Guide for more information about the INT32 API.

See also int32

*line-arguments-list*

Variable

Summary List of the command line arguments used when LispWorks was invoked.

Package system

Initial Value nil

Description 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.

You can implement command line processing in your application by testing elements in *line-arguments-list*. Use a string comparison function such as string= to compare them.

For a description of the command line arguments processed by LispWorks, see "The Command Line" in the LispWorks User Guide.

See also lisp-image-name
**load-data-file**

**Function**

**Summary**
Loads a fasl file created by `dump-forms-to-file` or `with-output-to-fasl-file`.

**Package**
`system`

**Signature**
`load-data-file file &rest args => result`

**Arguments**
- `file` A pathname designator.
- `args` Arguments passed to `load`.

**Values**
- `result` A generalized boolean.

**Description**
The function `load-data-file` loads a fasl file created by `dump-forms-to-file` or `with-output-to-fasl-file`.

`load-data-file` has the same semantics as `load`, but treats fasl files differently:

- It cannot load a fasl generated by `compile-file`.
- It allows loading of fasls generated by `dump-forms-to-file` or `with-output-to-fasl-file`, including those generated by a previous version of LispWorks.

`load-data-file` 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.

Fasl files generated by `dump-forms-to-file` or `with-output-to-fasl-file` must only be loaded using `load-data-file`.

The pathname specified by `file` must be recognized as a fasl file type, otherwise `load-data-file` will load it as a text file.

**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 `compile-file`. 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 **fixnum** type in LispWorks 5.0 and later is larger than in LispWorks 4.x on Windows and Linux. A **bignum** dumped in a LispWorks 4.x data file will be loaded as a **fixnum** in LispWorks 5.0 and later if its value is within the **fixnum** range.

See also

- **dump-forms-to-file**
- **with-output-to-fasl-file**

---

### locale-file-encoding

**Function**

**Summary**

Provides an encoding corresponding to the current code page on Microsoft Windows, and the locale on Unix.

**Package**

*system*

**Signature**

`locale-file-encoding 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**

- **new-ef-spec**
  
  Default external format spec created by merging `ef-spec` with the encoding that was found.
The function `locale-file-encoding` consults the ANSI code page on Microsoft Windows. If the code page identifier is in `win32:*latin-1-code-pages*`, `locale-file-encoding` merges `ef-spec` with `:latin-1`. 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 `win32:*latin-1-code-pages*` then `locale-file-encoding` merges `ef-spec` with an encoding corresponding to the current code page that gives an error for characters that cannot be encoded.

`locale-file-encoding` merges `ef-spec` with `:latin-1` on Unix.

See also
* `*file-encoding-detection-algorithm*`
* `*latin-1-code-pages*`
* `*multibyte-code-page-ef*`
* `safe-locale-file-encoding`

### make-gesture-spec

**Function**

**Summary**
Create a Gesture Spec object.

**Package**
`system`

**Signature**
`make-gesture-spec data modifiers => gspec`

**Arguments**
- `data` A non-negative integer less than `char-code-limit`, or a Gesture Spec keyword, or `nil`.
- `modifiers` A non-negative integer less than 64, or `nil`.

**Values**
- `gspec` A Gesture Spec object

**Description**
The function `make-gesture-spec` returns a new Gesture Spec object `gspec`. This can be used to represent a keystroke consisting of the key indicated by `data`, modified by the modifier keys indicated by `modifiers`.
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 simple-int32-vector objects.

Package
system

Signature
make-simple-int32-vector length &key initial-contents initial-element => vector

Arguments
length A non-negative fixnum.
initial-contents A sequence of integers of type (signed-byte 32), or nil.
initial-element An integer of type (signed-byte 32).

Values
vector A simple-int32-vector.

Description
The function make-simple-int32-vector is the constructor for simple-int32-vector objects.
The argument initial-contents, if supplied, should be a sequence of length length. It specifies the contents of vector.
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  

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte-length</td>
<td>A non-negative fixnum.</td>
<td></td>
</tr>
</tbody>
</table>

Values  

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vector</td>
<td>A vector.</td>
</tr>
</tbody>
</table>

Description  
The function `make-typed-aref-vector` returns a vector which is suitable for efficient access at compiler optimization level safety = 0. Use `typed-aref` to access `vector` efficiently.

See also  

<table>
<thead>
<tr>
<th>See also</th>
</tr>
</thead>
<tbody>
<tr>
<td>typed-aref</td>
</tr>
</tbody>
</table>

marking-gc

Function

Summary  
Performs a Marking GC in 64-bit LispWorks.

Package  
	system

Signature  

marking-gc gen-num &key what-to-copy max-size fragmentation-threshold

Arguments  

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gen-num</td>
<td>An integer in the inclusive range [0, 7].</td>
<td></td>
</tr>
<tr>
<td>max-size</td>
<td>A positive number or nil.</td>
<td></td>
</tr>
<tr>
<td>fragmentation-threshold</td>
<td>A number in the inclusive range [0, 10].</td>
<td></td>
</tr>
</tbody>
</table>

Description  
The function `marking-gc` garbage collects (GCs) the generation specified by `gen-num`, 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  

```lisp
(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  

```lisp
(set-maximum-memory)
```
**merge-ef-specs**

*Function*

**Summary**

Creates a new external format spec from two other external format specs.

**Package**

`system`

**Signature**

`merge-ef-specs ef-spec1 ef-spec2 => ef-spec`

**Arguments**

- `ef-spec1` An external format spec.
- `ef-spec2` An external format spec.

**Values**

- `ef-spec` The resultant external format spec created from information in `ef-spec1` and `ef-spec2`.

**Description**

The function `merge-ef-specs` returns an external format spec constructed by adding information not supplied in `ef-spec1` from `ef-spec2`.

Each external format spec argument is either a symbol or a list.

If `ef-spec1` and `ef-spec2` have the same value for their name component (whether they are lists or symbols), return `ef-spec1` combined with any parameters from `ef-spec2` that are not specified in `ef-spec1`.

Otherwise, if `ef-spec1` is `:default` or a list beginning with `:default`, return `ef-spec2` with parameters modified to be a union of the parameters from `ef-spec1` and `ef-spec2`, with those from `ef-spec1` taking priority.

Otherwise, return `ef-spec1` with any `:eol-style` parameter from `ef-spec2` if `ef-spec1` does not specify `:eol-style`.

---

**object-address**

*Function*

**Summary**

Returns the address of the given `object` as an integer.
Package system

Signature object-address object => address

Arguments object The object whose address should be returned.

Values address The address of object. An integer.

Description Returns the address of the given object 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 sys:object-address is the same as the one printed by the print-object method for generic-function.

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 >

See also pointer-from-address

open-pipe Function

Summary Runs a command in a subshell.

Package system

Signature open-pipe command &key direction element-type interrupt-off shell-type => stream
Arguments

- **command**: A string, a list of strings, a simple-vector of strings, or **nil**.
- **direction**: `:input`, `:output` or `:io`.
- **element-type**: A type specifier.
- **interrupt-off**: A boolean. Not implemented on Microsoft Windows.
- **shell-type**: A shell type.

Values

- **stream**: A pipe stream.

Description

On Unix/Linux/Mac OS X the behaviour of `open-pipe` is analogous to that of `popen` 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.

On Microsoft Windows `open-pipe` calls `CreateProcess` and `CreatePipe` and returns a bidirectional stream.

If `command` is a string then it is passed to the shell as the command to run without any arguments. If `command` 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 argv[0] in C, and so on). If `command` 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 argv[0] in C, and so on). If `command` is **nil**, then the shell is run.

`direction` is a keyword for the stream direction. The default value is `:input`. Bidirectional (I/O) pipes may be created by pasing `:io`. See the example below. This argument is ignored on Microsoft Windows.

`element-type` specifies the type of the stream as with `open`. The default value is `base-char`. This argument is ignored on Microsoft Windows.
interrupt-off, if t, ensures that Ctrl+C (SIGINT) to the LispWorks 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

display-url

Function
display-url

Summary Displays a HTML page in a web browser.

Package system

Signature display-url url

Arguments url A string.

Description The function display-url displays the page at the URL url 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 hqn-web:browse please change to use open-url in LispWorks 5.0 and later.

Examples
(sys:open-url "www.lispworks.com")

See also *browser-location*

**pid-exit-status**

*Function*

**Summary**
Returns the exit status of a process executed with run-shell-command.

**Package**
system

**Signature**
 pid-exit-status pid &key wait name => exit-status

**Arguments**

- **pid** A process ID.
- **wait** A boolean, default value t.
- **name** A Lisp object, default value pid.

**Values**
exit-status An integer, or nil.

**Description**
The function pid-exit-status returns the exit status of a process executed by run-shell-command with argument save-exit-status passed a non-nil value.

If wait is true then pid-exit-status waits until the process exits, using name in the wait message. If wait is nil and the process has not terminated, then pid-exit-status returns nil immediately.
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

```lisp
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

**print-pretty-gesture-spec**

*Function*

**Summary**
Prints a Gesture Spec object as a keystroke.

**Package**
*system*

**Signature**
```
print-pretty-gesture-spec gspec stream &key force-meta-to-alt force-shift-for-upcase => gspec
```

**Arguments**
- `gspec`: A Gesture Spec object.
- `stream`: An output stream.
- `force-meta-to-alt`: A boolean.
- `force-shift-for-upcase`: A boolean.

**Values**
- `gspec`: The Gesture Spec object that was passed.

**Description**
The function `print-pretty-gesture-spec` prints the keystroke represented by the Gesture Spec object `gspec` to the stream `stream`.

If `force-meta-to-alt` is true, then `gesturespec-meta-bit` is represented as `Alt` in the output; otherwise it is represented as `Meta`. `force-meta-to-alt` defaults to `nil`.

If `force-shift-for-upcase` is true and `gspec` represents uppercase input such as `A`, then the `Shift` modifier is printed, indicating that `Shift` is pressed to obtain the `A` character. `force-shift-for-upcase` defaults to `t`.

If `gspec` has a wild modifiers or data component (that is, `gesturespec-modifiers` and/or `gesturespec-data` return `nil`) then `<Wild>` appears in the output.
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 readtable 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**

**Function**

**Summary**
Gets or sets a registry path for use with your software.

**Package**
**system**

**Signature**
`product-registry-path product => path-string`

**Signature**
`(setf product-registry-path) path product => path`

**Arguments**
`product` A Lisp object.

**Values**
`path` The path as a string or a list of strings.
`path-string` The path as a string.

**Description**
The function **product-registry-path** returns the registry subpath defined for the product denoted by `product`, as a string.

The function `(setf product-registry-path)` sets the registry subpath for the product denoted by `product`.

If `path` 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.

If `path` is a list of strings, then it is interpreted like the directory component of a pathname.

This registry subpath is used when reading and storing user preferences with **user-preference**.

Note that while `product` can be any Lisp object, values of `product` are compared by **eq**, so you should use keywords.

**Note:** to store CAPI window geometries under the registry path for your product, see the entry for **capi:top-level-**
The SYSTEM Package

**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

**run-shell-command**

*Function*

**Package**

system

**Signature**

```lisp
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
```

**Arguments**

- **command**
  
  A string, a list of strings, a simple-vector of strings, or `nil`.

- **input**
  
  `nil`, `:stream` or a file designator. Default value `nil`.

- **output**
  
  `nil`, `:stream` or a file designator. Default value `nil`.

- **error-output**
  
  `nil`, `:stream`, `:output` or a file designator. Default value `nil`.

- **separate-streams**
  
  A boolean. True value not currently supported.

- **wait**
  
  A boolean, default value `t`.

- **if-input-does-not-exist**
  
  `:error`, `:create` or `nil`. Default value `:error`.

- **if-output-exists**
  

- **if-error-output-exists**
  
  ...
The function **run-shell-command** allows Unix shell commands to be called from Lisp code with redirection of the std-out, stdin and stderr to Lisp streams. It creates a subprocess which executes the command `command`.

The argument `command` is interpreted as by `call-system`. 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.

If `wait` is true, then `run-shell-command` executes `command` and does not return until the process has exited. In this case none of `input`, `output` or `error-output` may have the value `:stream`, and the single value `result` is the exit status of the process that ran `command`.

If `wait` is `nil` and none of `input`, `output` or `error-output` have the value `:stream` then `run-shell-command` executes `command` and returns a single value `result` which is the process ID of the process running `command`.

### Values

- **result**: The exit status of the process running command, or a process ID.
- **stream**: A stream, or `nil`.
- **error-stream**: A stream, or `nil`.
- **process**: A process ID.

### Description


**show-window**: A boolean. True value not currently supported.

**environment**: An alist of strings naming environment variables and values. Default value `nil`.

**element-type**: Default value `base-char`.

**save-exit-status**: A boolean, default value `nil`.
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 pro-
cess.

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 combina-
tion of stream arguments:
### Table 16.1 The streams created by run-shell-command

<table>
<thead>
<tr>
<th>Arguments</th>
<th>stream</th>
<th>error-stream</th>
</tr>
</thead>
<tbody>
<tr>
<td>input is :stream output is :stream</td>
<td>An I/O stream connected to stdin and stdout</td>
<td>An input stream connected to stderr</td>
</tr>
<tr>
<td>input is not :stream output is :stream error-output is :stream</td>
<td>An input stream connected to stdout</td>
<td>An input stream connected to stderr</td>
</tr>
<tr>
<td>input is :stream output is not :stream error-output is :stream</td>
<td>An output stream connected to stdin</td>
<td>An input stream connected to stderr</td>
</tr>
<tr>
<td>input is not :stream output is not :stream error-output is :stream</td>
<td>nil</td>
<td>An input stream connected to stderr</td>
</tr>
<tr>
<td>input is :stream output is :output error-output is :output</td>
<td>An I/O stream connected to stdin, stdout and stderr</td>
<td>nil</td>
</tr>
<tr>
<td>input is not :stream output is :stream error-output is :output</td>
<td>An input stream connected to stdout and stderr</td>
<td>nil</td>
</tr>
<tr>
<td>input is :stream output is not :stream error-output is :output</td>
<td>An output stream connected to stdin</td>
<td>nil</td>
</tr>
<tr>
<td>input is not :stream output is not :stream error-output is :output</td>
<td>nil</td>
<td>nil</td>
</tr>
<tr>
<td>input is :stream output is not :stream error-output is not :stream or :output</td>
<td>An I/O stream connected to stdin and stdout</td>
<td>nil</td>
</tr>
<tr>
<td>input is not :stream output is :stream error-output is not :stream or :output</td>
<td>An input stream connected to stdout</td>
<td>nil</td>
</tr>
</tbody>
</table>
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"
Function

safe-locale-file-encoding

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
set-automatic-gc-callback

Function

Summary
Sets a function or functions to call after an automatic GC in 64-bit LispWorks.

Package
system

Signature
set-automatic-gc-callback blocking-gen-num-func &optional other-func => other-func

Arguments
blocking-gen-num-func
A function designator for a function of two arguments, or nil.

other-func
A function designator for a function of one argument, or nil.

Values
other-func
A function designator for a function of one argument, or nil.

Description
The function set-automatic-gc-callback sets a function or functions to call after an automatic garbage collection (GC).

If blocking-gen-num-func is a function designator it should take two arguments: the generation number and, if do-gc in the last call to set-blocking-gen-num was a number, the number of copied segments. It is called whenever the blocking generation is GCed automatically. If blocking-gen-num-func is nil, then this callback is switched off.

If other-func is a function designator it should take one argument, the generation number that was GCed. It is called whenever an automatic GC occurred and blocking-gen-num-func was not called, either because the blocking generation was not GCed, or because blocking-gen-num-func was passed as nil. If other-func is nil (the default) then this callback is switched off.

The calls occur after the GC has finished and there is no restriction on what they can do. If the call ends up allocating
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
=> 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.
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 \texttt{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 \texttt{gen-num} during the operation. The macro \texttt{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 \texttt{gc-generation} explicitly to promote long living objects to a higher generation. The advantage of doing this is that you can call \texttt{gc-generation} in places where you know there are not many short-lived objects alive.

Passing a \texttt{do-gc} value other than \texttt{t} is useful when the blocking generation can be large enough that copying it all may cause very serious paging. Passing \texttt{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 \texttt{gc-generation} or \texttt{marking-gc}.

\texttt{set-blocking-gen-num} returns four values: the old blocking generation number, the old value of \texttt{do-gc}, the \texttt{max-size}, and the old value of \texttt{gc-threshold}. It can be called with \texttt{gen-num nil} to query the values without changing any of them.

\textbf{Note}: this function is implemented only in 64-bit LispWorks. It is not relevant to the Memory Management API in 32-bit implementations.

\textbf{See also}

- \texttt{block-promotion}
- \texttt{gc-generation}
- \texttt{marking-gc}
- \texttt{set-automatic-gc-callback}
- \texttt{set-gen-num-gc-threshold}
**set-default-segment-size**

**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.
- `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**  
`system`

**Signature**  
`set-delay-promotion gen-num on => on`

**Arguments**  
- `gen-num`  
  An integer between 0 and 7, inclusive.
- `on`  
  A generalized boolean.

**Values**  
- `on`  
  A generalized boolean.

**Description**  
The function `set-delay-promotion` delays promotion for generation `gen-num`, which means that objects are promoted to the next generation in the second garbage collection (GC) that they survive in generation `gen-num`. By default, objects are promoted in the first GC.

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.

**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-file-dates

Summary
Sets the modification and access times of a file.

Package
system

Signature
(set-file-dates file &key creation modification access)

Arguments
- file: A pathname designator.
- creation: A non-negative integer, or nil.
- modification: A non-negative integer, or nil.
- access: A non-negative integer, or nil.

Description
The function set-file-dates sets the modification and access times of the file file for each of modification and access that is non-nil.

On Microsoft Windows, if creation is non-nil, the creation time of the file is also set. creation is ignored on other platforms.

Each keyword argument is interpreted as a universal time representing the time to set, unless it is nil in which case the corresponding time for file is not changed. Each keyword argument has default value nil.

An error of type file-error is signalled on failure.

See also
open

set-gen-num-gc-threshold

Summary
Sets the additional allocation threshold that triggers a GC in the blocking generation in 64-bit LispWorks.

Package
system
The function `set-gen-num-gc-threshold` sets the threshold for additional allocation that triggers a garbage collection (GC) in generation `gen-num` when this is the blocking generation (as set by `set-blocking-gen-num`). A GC is triggered when the allocation in generation `gen-num` grows more than `threshold` over the allocation after the last GC of this generation (or a GC of a higher generation).

To set the threshold, `threshold` can be an integer greater than 12800, which is interpreted as the absolute value. Alternatively `threshold` 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.

The default threshold for all generations is 1. That is, for all generations `gen-num`, when generation `gen-num` is the blocking generation and allocation in it has doubled since the previous GC, generation `gen-num` is collected automatically.

`set-gen-num-gc-threshold` can be called when the generation `gen-num` is not the blocking generation, and will set the value for that `gen-num`. Such a call will not take effect until the generation `gen-num` becomes the blocking generation, as set by a call to `set-blocking-gen-num` (with `:do-gc non-nil`).

Increasing the threshold reduces the number of GC calls, but may increase the virtual memory usage.
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
- `check-fragmentation`
- `mark-and-sweep`
- `memory-growth-margin`
- `room-values`

### set-maximum-segment-size

**Function**

**Summary**

Defines the maximum segment size for a generation and allocation type in 64-bit LispWorks.

**Package**

`system`

**Signature**

`set-maximum-segment-size gen-num allocation-type size-in-mb`

**Arguments**

- `gen-num`
  
  An integer between 0 and 7, inclusive.

- `allocation-type`
  

- `size-in-mb`
  
  An integer between 1 and 256 inclusive, or `nil`.

**Values**

- `max-segment-size`
  
  A number.
The function `set-maximum-segment-size` sets the maximum segment size for a generation and allocation type in 64-bit LispWorks.

`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.

`size-in-mb` is the size in megabytes.

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.

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 `marking-gc` with `what-to-copy` non-nil or `set-blocking-gen-num` with `do-gc` 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 `max-size` parameter is set to a low number to avoid using too much virtual memory.

The returned value, `max-segment-size`, is the previous maximum segment size.

If `size-in-mb` is a number, it specifies the size in megabytes. If `size-in-mb` is `nil` then `set-maximum-segment-size` returns the maximum segment size without altering 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  
marking-gc  
set-blocking-gen-num  
set-default-segment-size

set-memory-check

Function

Summary  
Sets a memory check in 64-bit LispWorks.

Package  
system

Signature  
set-memory-check size function

Arguments  
size        An integer.
function    A function designator.

Description  
The function set-memory-check sets a memory check.

size must be an integer. It specifies the total size in bytes of the mapped areas of Lisp at which the check is triggered.

function is a function of no arguments.

After each automatic garbage collection (GC) the system checks whether the mapped area (excluding stacks) is larger than size. If it is larger, function is called with no arguments.

Inside the dynamic scope of the call, the check is disabled. There are no restrictions or special considerations on what the function function does.

The current mapped area can be found by the :total-size value returned by room-values.

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-exhausted-callback
**set-memory-exhausted-callback**

*Function*

**Summary**
Sets a callback that is called when memory is exhausted in 64-bit LispWorks.

**Package**
`system`

**Signature**
```lisp
(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

```lisp
(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 LispWorks 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  system

Signature  simple-augmented-string length

Arguments  length  The length of the string (or *, meaning any).

Description  This is the simple version of augmented-string, that is, the string itself is simple. Equivalent to:
               (simple-vector character length)

See also  augmented-string

simple-augmented-string-p

Function

Summary  Tests if an object is a simple augmented string.

Package  system

Signature  simple-augmented-string-p object => bool

Arguments  object  The object to be tested.
Values

bool t if object is a simple augmented string; nil otherwise.

Description

This is the predicate for simple augmented strings.

See also

simple-augmented-string

**simple-int32-vector**

Type

Summary

A type for simple vectors of int32 objects.

Package

system

Signature

simple-int32-vector

Description

The type simple-int32-vector provides simple vectors of int32 objects and can be used to generate optimal 32-bit arithmetic code. Create a simple-int32-vector by calling make-simple-int32-vector.

See the section "Fast 32-bit arithmetic" in the LispWorks User Guide for more information.

See also

int32
int32-aref
make-simple-int32-vector

**stack-overflow-behaviour**

Variable

Summary

Controls behavior when stack overflow occurs.

Package

system

Initial Value

:error
Description

The variable *stack-overflow-behaviour* controls behavior when stack overflow occurs.

When *stack-overflow-behaviour* is set to :error, LispWorks signals an error.

When it is set to :warn, LispWorks increases the stack size automatically to accommodate the overflow, but prints a warning message to signal that this has happened.

When it is set to nil, LispWorks increases stack size silently.

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

*sg-default-size*

---

**staticp**

Function

Summary

Specifies whether a given object has been allocated in static memory.

Package

system

Signature

staticp obj => bool

Arguments

obj An object.

Values

bool t if the object is allocated in static memory; nil otherwise.

Description

This predicate can be used on an object to find out whether it is allocated in static memory.

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 not,
however. Therefore `staticp` 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**  
`storage-condition`

**Initargs**

- `:gen-num`  
The number of the generation in which the system was trying to allocate.

- `:size`  
The size in bytes which the system was trying to allocate.

- `:type`  
A string naming the allocation type the system was trying to allocate.

- `:static`  
A boolean, true if the system was trying to allocate a static object, and false otherwise.

**Accessors**

- `storage-exhausted-gen-num`
- `storage-exhausted-size`
- `storage-exhausted-static`
- `storage-exhausted-type`

**Description**  
The class `storage-condition` is a condition class used for reporting failures to map memory. Allocation types are as described in `set-maximum-segment-size`.

**See also**

- `set-memory-exhausted-callback`
**sweep-gen-num-objects**

*Function*

**Summary**

Applies a function to all the live objects in a generation in 64-bit LispWorks.

**Package**

`system`

**Signature**

`sweep-gen-num-objects gen-num function`

**Arguments**

`gen-num` An integer in the inclusive range [0,7].

`function` A designator for a function of one argument, the object.

**Values**

`sweep-gen-num-objects` returns `nil`.

**Description**

The function `sweep-gen-num-objects` applies `function` to all the live objects in the generation `gen-num`.

`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:** `sweep-gen-num-objects` is not implemented in 32-bit LispWorks, where you can use `sweep-all-objects` instead.

**See also**

`sweep-all-objects`

**typed-aref**

*Function*

**Summary**

Accesses a typed aref vector efficiently.

**Package**

`system`

**Signature**

`typed-aref type vector byte-index => value`

`(setf typed-aref) value type vector byte-index => value`
### Arguments

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>A type specifier.</td>
</tr>
<tr>
<td>vector</td>
<td>A vector created by <code>make-typed-aref-vector</code>.</td>
</tr>
<tr>
<td>byte-index</td>
<td>A non-negative fixnum.</td>
</tr>
</tbody>
</table>

### Values

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>An object of type type.</td>
</tr>
</tbody>
</table>

### Description

The function `typed-aref` allows efficient access to a typed aref vector.

**type** must evaluate to one of: `double-float`, `float`, `single-float`, `(unsigned-byte 32)`, `(signed-byte 32)`, `(unsigned-byte 16)`, `(signed-byte 16)`, `(unsigned-byte 8)` or `(signed-byte 8)`.

**vector** must be an object returned by `make-typed-aref-vector`.

**byte-index** 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 **byte-length** argument passed to `make-typed-aref-vector`.

`typed-aref` and `(setf typed-aref)` will be inlined to code which is as efficient as possible when compiled with `(optimize (safety 0))` and a constant type. As usual, you need to add `(optimize (float 0))` to remove boxing for the float types.

**Note:** Efficient access to foreign arrays is also available. See `fli:foreign-typed-aref` in the LispWorks Foreign Language Interface User Guide and Reference Manual

### Example

```lisp
(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)
```

### See also

`make-typed-aref-vector`
**wait-for-input-streams Function**

**Summary**
Waits for input on a list of socket streams, returning those that are ready.

**Package**
`system`

**Signature**
`wait-for-input-streams 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 list of `socket-streams` or `nil`.

**Description**
The function `wait-for-input-streams` waits for any of the streams in the argument `streams` 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.

When any of the streams is ready for input, `wait-for-input-streams` returns a list of all the streams that are ready, in the same order that they appear in `streams`.

If `timeout` is non-`nil` it must be a real number, specifying a timeout in seconds. If `timeout` seconds pass and none of the streams is ready, `wait-for-input-streams` returns `nil`.

If `timeout` is `0`, `wait-for-input-streams` returns all of the streams that are ready immediately, without waiting at all. That is, it behaves like `listen` on many streams.

If `wait-function` is supplied, it is called periodically with no arguments, and if it returns non-`nil` then `wait-for-input-streams`...
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

**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 \texttt{wait-for-input-streams}
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

dismiss-splash-screen

Function

Summary       Makes a startup screen disappear.
Package   win32
Signature  dismiss-splash-screen &optional forcep
Arguments  forcep   A generalized boolean.
Description The function dismiss-splash-screen makes a startup screen
(as specified via the :startup-bitmap-file delivery keyword) disappear.

If forcep is nil then the startup screen is displayed for a minimum of 5 seconds before disappearing. If forcep is true then the startup screen disappears when dismiss-splash-screen is called. The default value of forcep is nil.

If dismiss-splash-screen 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 :startup-bitmap-file in the LispWorks Delivery User Guide.

*latin-1-code-pages*  Variable

Summary   Windows Code Pages for which Latin-1 encoded files are used.

Package   win32
Initial Value   (1252 28591)
Description The value of *latin-1-code-pages* 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 (:latin-1 :encoding-error-action 63) 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
17 The WIN32 Package (including DDE)  This chapter applies only to LispWorks for Windows

Signature  set-application-themed on/off
Arguments  on/off A generalized boolean.
Description The function set-application-themed controls whether a LispWorks application should be themed.
On Windows XP, LispWorks is “themed”, that is it uses the current theme of the desktop. You can switch this off by calling

(win32:set-application-themed nil)

On non-XP systems, or when the application does not have Common Controls 6, this call has no effect.
set-application-themed 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, set-application-themed can be called multiple times in the same run.

17.2 Windows registry API

close-registry-key Function

Summary Closes a handle to an open registry key.
Signature close-registry-key handle &key errorp => successp, error-code
Arguments handle A handle to an open registry key.
Values successp A boolean.
error-code An integer error code or nil.
Description

The function *close-registry-key* closes `handle`, which should be an open registry key handle.

The return value on success is `t`.

If an error occurs and `errorp` is true then an error is signalled. Otherwise, the return values are `nil` and the Windows `error-code`. The default value of `errorp` is `t`.

See also

*create-registry-key*
*open-registry-key*

---

**collect-registry-subkeys**

*Function*

**Summary**

Returns names of the subkeys of a registry key.

**Signature**

```
collect-registry-subkeys subkey &key root max-name-size
max-names errorp value-function => subsubkeys
```

**Arguments**

- `subkey` A string specifying the name of the key.
- `root` A keyword or handle.
- `max-name-size` An integer.
- `max-names` An integer.
- `errorp` A boolean.
- `value-function` A function designator or `nil`.

**Values**

- `subsubkeys` A list.

**Description**

The function `collect-registry-subkeys` returns a list of names which are subsubkeys of `subkey` under the key `root`. `subkey` and `root` are interpreted as described for `create-registry-key`. The default value of `root` is `:user`.

---

This chapter applies only to LispWorks for Windows
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

\[
\text{value-function} \ handle \ subsubkey-name \Rightarrow \ name, \ collectp
\]

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**

*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.
max-name-size An integer.
max-buffer-size An integer.
expected-type A keyword or `t`.
errorp A boolean.
value-function A function or symbol.

Values

values-alist An alist.

Description

The function `collect-registry-values` returns an alist of all the values of `subkey` under the key `root`.

`subkey` and `root` are interpreted as described for `create-registry-key`. The default value of `root` is `:user`.

`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-buffer-size` specifies the maximum length in bytes of the data. If the data is longer than this, an error is signalled. The default value of `max-buffer-size` is 1024.

If `value-function` is `nil`, the returned `values-alist` is an association list containing pairs `(name . data)` consisting of the names and data of the values of `subkey`. `expected-type` controls how certain types are converted to Lisp objects as described for `enum-registry-value`. The default value of `expected-type` is `t`.

If `value-function` is non-`nil`, it should be a function with signature

```
(value-function handle subsubkey-name-and-value => name-and-value, collectp)
```

`value-function` is funcalled for each subsubkey with the handle of `subkey` and a cons of the name and value of the subsubkey. If `collectp` is non-`nil` then `name-and-value` is collected into
the alist \texttt{values-alist} to return from \texttt{collect-registry-values}. Otherwise \texttt{name-and-value} is ignored.

If an error occurs and \texttt{errorp} is true, then an error is signalled. Otherwise, \texttt{values-alist} is returned as \texttt{nil} if \texttt{subkey} could not be opened at all or contains \texttt{nil} for the data of any particular pair that cannot be read. The default value of \texttt{errorp} is \texttt{t}.

\textbf{See also}
\begin{itemize}
  \item \texttt{collect-registry-subkeys}
  \item \texttt{create-registry-key}
  \item \texttt{enum-registry-value}
\end{itemize}

\section*{create-registry-key}

\textit{Function}

\textbf{Summary}
Creates a new registry key.

\textbf{Signature}
\begin{verbatim}
create-registry-key subkey &key class root access errorp =>
handle, disposition, error-code
\end{verbatim}

\textbf{Arguments}
\begin{itemize}
  \item \texttt{subkey} A string specifying the name of the key.
  \item \texttt{class} A string.
  \item \texttt{root} A keyword or handle.
  \item \texttt{access} A keyword or an integer.
  \item \texttt{errorp} A generalized boolean.
\end{itemize}

\textbf{Values}
\begin{itemize}
  \item \texttt{handle} The handle of the new key.
  \item \texttt{disposition} A keyword, either \texttt{:created-new-key} or \texttt{:opened-existing-key}.
  \item \texttt{error-code} An integer error code or \texttt{nil}.
\end{itemize}

\textbf{Description}
The function \texttt{create-registry-key} creates a new registry key named \texttt{subkey} under the parent key \texttt{root}. 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.
errorp A generalized boolean.

Values
successp A boolean.
error-code An integer error code or nil.

Description The function delete-registry-key deletes the registry key named subkey under the parent key root.

subkey and root are interpreted as described for create-registry-key. The default value of root is :user.
The value t is returned if the key is deleted successfully.
If an error occurs and errorp is true then an error is signalled. Otherwise, the return values are nil and the Windows error-code. The default value of errorp is t.

See also create-registry-key

enum-registry-value Function

Summary Enumerates the values of a registry key.

Signature enum-registry-value subkey index &key root max-name-size max-buffer-size expected-type errorp => name, data-type, data, error-code

Arguments subkey A string specifying the name of the key.
index An integer.
root A keyword or handle.
max-name-size An integer.
max-buffer-size An integer.
expected-type A keyword or t.
errorp A boolean.
Values

- **name**: A string.
- **data-type**: A keyword.
- **data**: A lisp object.
- **error-code**: An integer error code or **nil**.

Description

The function **enum-registry-value** allows the values of sub-key under the key **root** to be enumerated.

- **subkey** and **root** are interpreted as described for **create-registry-key**. The default value of **root** is **:user**.
- **index** specifies which value to return, with 0 being the first item.
- **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-buffer-size** specifies the maximum length in bytes of the value. The value is longer than this, an error is signalled. The default value of **max-buffer-size** is 1024.

If the value exists (that is, **index** is not too large), then the return values are the name, data type and data associated with the value in the registry. The argument **expected-type** controls how certain data types are converted to Lisp objects as follows:

<table>
<thead>
<tr>
<th>data-type</th>
<th>expected-type</th>
<th>Description of converted data</th>
</tr>
</thead>
<tbody>
<tr>
<td>:string</td>
<td>:lisp-object</td>
<td>String made with <strong>read-from-string</strong></td>
</tr>
<tr>
<td>:string</td>
<td>Not supplied</td>
<td>String, exactly as in the registry</td>
</tr>
</tbody>
</table>
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`

### Table 17.1 Conversion of registry values to Lisp objects

<table>
<thead>
<tr>
<th>data-type</th>
<th>expected-type</th>
<th>Description of converted data</th>
</tr>
</thead>
<tbody>
<tr>
<td>:environment-string</td>
<td>:string</td>
<td>String, exactly as in the registry</td>
</tr>
<tr>
<td>:environment-string</td>
<td>Not supplied</td>
<td>String, environment variables expanded</td>
</tr>
<tr>
<td>:integer</td>
<td>Not supplied</td>
<td>Integer</td>
</tr>
<tr>
<td>:little-endian-integer</td>
<td>Not supplied</td>
<td>Integer</td>
</tr>
<tr>
<td>:binary</td>
<td>Not supplied</td>
<td>A newly allocated foreign object</td>
</tr>
<tr>
<td>:binary</td>
<td>:lisp-object</td>
<td>Vector, element type (\text{unsigned-byte}\ 8)</td>
</tr>
</tbody>
</table>

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.
This chapter applies only to LispWorks for Windows

access An integer or keyword.
errorp A generalized boolean.

Values handle The handle of the key.
error-code An integer error code or nil.

Description The function open-registry-key opens a registry key named subkey under the parent key root. subkey and root are interpreted as described for create-registry-key. If subkey is an empty string, then the root key is returned. The default value of root is :user.

If access is :read, then it opens the key with KEY_READ permissions. If access is :write, then it opens the key with KEY_WRITE permissions. If access is an integer, then the value access specifies the desired Win32 access rights. If access is omitted and root is :user, then open-registry-key uses KEY_ALL_ACCESS. Otherwise it uses KEY_READ.

The return value on success is the handle of the opened key.

If an error occurs and errorp is true, then an error is signalled. Otherwise, the return values are nil and the Windows error-code. The default value of errorp is t.

See also create-registry-key

query-registry-key-info Function

Summary Returns information about an open registry key handle.

Signature query-registry-key-info key => info, error-code

Arguments key A handle.

Values info A property list.
The WIN32 Package (including DDE)

This chapter applies only to LispWorks for Windows

error-code  An integer error code or nil.

Description

The function query-registry-key-info returns a plist of information about the open registry key handle key. The elements of the plist info 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.

query-registry-value

Function

Summary

Returns a value stored in the registry.

Signature

query-registry-value  subkey name &key root expected-type
errorp => data, successp, error-code

Arguments

subkey  A string specifying the name of the key.

name  A string specifying the name of the value.

root  A keyword or handle.
expected-type A keyword or t.
errorp A boolean.

Values
data A Lisp object.
successp A boolean.
error-code An integer error code or nil.

Description
The function \texttt{query-registry-value} returns the value associated with \textit{name} in \textit{subkey} under the key \textit{root}. \textit{subkey} and \textit{root} are interpreted as described for \texttt{create-registry-key}. If \textit{subkey} is an empty string, then the \textit{root} key is returned. The default value of \textit{root} is :user.

If the value exists, then the return values are the data and true. \textit{expected-type} controls how certain types are converted to the Lisp object \textit{data} as described for \texttt{enum-registry-value}. The default value of \textit{expected-type} is t.

If an error occurs and \textit{errorp} is true then an error is signalled. Otherwise, the return values are nil, nil and the Windows \textit{error-code}. The default value of \textit{errorp} is t.

See also \texttt{create-registry-key} \texttt{enum-registry-value}

\begin{verbatim}
registry-key-exists-p

Function
Summary
The predicate for whether a registry key can be opened.

Signature
registry-key-exists-p subkey &key root access => existsp

Arguments
subkey A string specifying the name of the key.
root A keyword or handle.
access An integer or keyword.
\end{verbatim}
Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>existsp</td>
<td>A boolean.</td>
</tr>
</tbody>
</table>

Description

The function `registry-key-exists-p` checks whether the registry key named `subkey` can be opened under the parent key `root` with the supplied `access` permissions.

`subkey` and `root` are interpreted as described for `create-registry-key`. The default value of `root` is `:user`.

If `access` is `:read`, then it opens the key with `KEY_READ` permissions. If `access` is `:write`, then it opens the key with `KEY_WRITE` permissions. If `access` is an integer, then the value `access` specifies the desired Win32 access rights. If `access` is omitted and `root` is `:user`, then `registry-key-exists-p` uses `KEY_ALL_ACCESS`. Otherwise it uses `KEY_READ`.

`registry-key-exists-p` closes the key before returning, but the return value is `t` if the key could actually be opened and `nil` otherwise.

See also `create-registry-key`

**registry-value**

*Accessor*

Summary

Gets or sets a value in the registry.

Signature

```lisp
(registry-value subkey name &key root expected-type errorp =>
data, successp, error-code)
```

```lisp
(setf registry-value) value subkey name &key root expected-type errorp => value
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>subkey</td>
<td>A string specifying the name of the key.</td>
</tr>
<tr>
<td>name</td>
<td>A string specifying the name of the value.</td>
</tr>
<tr>
<td>root</td>
<td>A keyword or handle.</td>
</tr>
<tr>
<td>expected-type</td>
<td>A keyword or <code>t</code>.</td>
</tr>
<tr>
<td>errorp</td>
<td>A boolean.</td>
</tr>
</tbody>
</table>
This chapter applies only to LispWorks for Windows

### Values
- **data**: A Lisp object.
- **successp**: A boolean.
- **error-code**: An integer error code or `nil`.

### Description
The function `registry-value` returns the value associated with `name` in `subkey` under the key `root`.

`subkey` and `root` are interpreted as described for `create-registry-key`. The default value of `root` is `:user`.

If the value exists, then the return values are the data and true. `expected-type` controls how certain types are converted to Lisp objects as described for `enum-registry-value`. 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` and the Windows `error-code`. The default value of `errorp` is `t`.

The function `(setf registry-value)` sets the value associated with `name` in `subkey` under the key `root`, creating the subkey if necessary. The default value of `root` is `:user`.

### See also
- `set-registry-value`
The function `set-registry-value` sets the value associated with `name` in `subkey` under the key `root`. The default value of `root` is `:user`.

The stored value is derived from `data`, converted according to `expected-type` as follows:

<table>
<thead>
<tr>
<th>Lisp data</th>
<th>expected-type</th>
<th>Registry type</th>
</tr>
</thead>
<tbody>
<tr>
<td>A string</td>
<td>:string</td>
<td>REG_SZ exactly as in <code>data</code></td>
</tr>
<tr>
<td>Lisp value</td>
<td>:lisp-object</td>
<td>REG_SZ made with <code>prin1-to-string</code> of <code>data</code></td>
</tr>
<tr>
<td>An integer</td>
<td>:integer</td>
<td>REG_DWORD containing <code>data</code></td>
</tr>
<tr>
<td>A foreign pointer</td>
<td>:binary</td>
<td>REG_BINARY containing bytes of one element at the pointer</td>
</tr>
<tr>
<td>An array</td>
<td>:binary</td>
<td>REG_BINARY containing bytes from the array</td>
</tr>
</tbody>
</table>

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`.

expected-type A keyword or `t`.
errorp A boolean.

Values
error-code An integer error code or `nil`.
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**

**Function**

**Summary**
Sets up an advise loop on a specified data item for a conversation.

**Package**
win32

**Signature**
\[
\text{dde-advise-start} \ conversation \ item \ &key \ key \ function \ format \ datap \ type \ errorp \ \Rightarrow \ result
\]

**Arguments**

- **conversation**: A conversation object.
- **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.

**Values**

- **result**: A boolean.

**Description**

The `dde-advise-start` function sets up an advise loop for the data item specified by `item` on the specified `conversation`. 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 – 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:

```lisp
(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**  
```lisp
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.
new-conversation-p

A boolean.

Values

result

A boolean.

Description

The \texttt{dde-advise-start*} function is similar to the \texttt{dde-advise-start}, and sets up an advise loop for the data item specified by \texttt{item} on a conversation recognizing the \texttt{service/topic} pair.

See \texttt{dde-advise-start} for information on the \texttt{format}, \texttt{type}, and \texttt{datap} arguments.

The argument \texttt{key} is used to identify this link. If specified as \texttt{nil} (the default value), it defaults to the conversation. Multiple links are permitted on a conversation with the same \texttt{item} and \texttt{format} values, as long as their \texttt{key} values differ.

If the link is established, the return value \texttt{result} is \texttt{t}. If the link could not be established, the behavior depends on the value of \texttt{errorp}. If \texttt{errorp} is \texttt{t} (the default value), LispWorks signals an error. If it is \texttt{nil}, the function returns \texttt{nil} to indicate failure.

If the link is established, the function \texttt{function} will be called whenever the data changes. If \texttt{function} is \texttt{nil} (the default value), the generic function \texttt{dde-client-advise-data} will be called.

The function specified by \texttt{function} should have a lambda list similar to the following:

\texttt{key item data &key conversation &allow-other-keys}

The arguments \texttt{key} and \texttt{item} identify the link. The argument \texttt{data} contains the new data for hot links; for warm links it is \texttt{nil}.

See also

\texttt{dde-advise-start}

\texttt{dde-advise-stop}
Function

dde-advise-stop

Summary
Removes a link from a conversation specified by a given item and key.

Package
win32

Signature
dde-advise-stop conversation item &key key format errorp disconnectp no-advice-ok => result

Arguments
- conversation: A conversation object.
- item: A string or symbol.
- key: An object.
- format: A clipboard format specifier.
- errorp: A boolean.
- disconnectp: A boolean.
- no-advice-ok: A boolean.

Values
- result: A boolean.

Description
The function dde-advise-stop removes a particular link from conversation specified 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, and the last advise loop for the conversation is terminated, the conversation is disconnected.

Attempting to remove a link that does not exist raises an error, unless no-advice-ok is t.
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`

### `dde-advise-stop*`

**Function**

**Summary**
Removes a link from an automatically managed conversation specified by a given item and key.

**Package**
`win32`

**Signature**
```
dde-advise-stop* service topic item &key key format errorp disconnectp => result
```

**Arguments**
- `service` A string or symbol.
- `topic` A string or symbol.
- `item` A string or symbol.
- `key` An object.
- `format` A clipboard format specifier.
- `errorp` A boolean.
- `disconnectp` A boolean.

**Values**
- `result` A boolean.

**Description**
The function `dde-advise-stop*` is similar to the function `dde-advise-stop`, 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**
`win32`

**Signature**

```
dde-connect service topic &key class errorp => object
```

**Arguments**

- `service` A symbol or string.
- `topic` A symbol or string.
- `class` The class of the conversation object to create.
- `errorp` A boolean.

**Values**

- `object` A conversation object.

**Description**

The function `dde-connect` attempts to create a conversation with a DDE server. 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.

The `class` argument specifies the class of the conversation object to create. It must be a subclass of `dde-client-`
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.
This chapter applies only to LispWorks for Windows 17.3 DDE client interface reference entries

Package win32

Signature dde-execute conversation command &rest {args}* => result

Arguments
- conversation: A conversation object.
- command: A string or symbol.
- args: An argument.

Values result: A boolean.

Description The function dde-execute provides an alternative syntax for dde-execute-command. Unlike dde-execute-command, dde-execute takes the arguments for command as a sequence of args following &rest, and does not have an argument for specifying how to handle an error.

See also dde-execute*
dde-execute-command*
dde-execute-string

**dde-execute**

Function

Summary An alternative syntax for dde-execute-command*.

Package win32

Signature dde-execute* service topic command &rest {args}* => result

Arguments
- service: A string or symbol.
- topic: A string symbol.
- command: A string or symbol.
- args: An argument.
Values

result A boolean.

Description

The function `dde-execute
` provides an alternative syntax for `dde-execute-command
`. Unlike `dde-execute-command
`, `dde-execute
` takes the arguments for `command` as a sequence of `args` following `&rest`, and does not have any arguments for specifying how to handle errors.

See also

dde-execute

dde-execute-command

dde-execute-string

### dde-execute-command

**Function**

**Summary**

Sends a command string to a specified conversation.

**Package**

`win32`

**Signature**

```
dde-execute-command conversation command arg-list &key errorp => result
```

**Arguments**

- `conversation` A conversation object.
- `command` A string or symbol.
- `arg-list` A list of strings, integers, and floats.
- `errorp` A boolean.

**Values**

result A boolean.

**Description**

The function `dde-execute-command` sends a command string to the conversation specified by `conversation`. 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, ...)]
```
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

\[ \text{command(arg1,arg2,...)} \]

If \textit{server} names a client service registered with \texttt{define-dde-client}, the registered service name is used as the DDE service name. If \textit{server} is any other symbol, the print name of the symbol is used as the DDE service name. If \textit{server} is a string, that string is used as the DDE service name.

The \textit{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 \texttt{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 \textit{service} and \textit{topic}, placing them inside a \texttt{with-dde-conversation} prevents a new conversation being established for each transaction.

If \textit{new-conversation-p} is set to \texttt{t} a new conversation is always established for the transaction. This new conversation is always automatically disconnected when the transaction is completed.

If \textit{connect-error-p} is \texttt{t} (the default value) and a conversation cannot be established, then LispWorks signals an error. If it is \texttt{nil}, \texttt{dde-execute-command*} returns \texttt{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 \texttt{t}. On failure, the behavior depends on the value of the \textit{errorp} argument. If \textit{errorp} is \texttt{t} (the default value), LispWorks signals an error. If it is \texttt{nil}, the function returns \texttt{nil} to indicate failure.

See also \texttt{dde-execute}
**dde-execute-string**  

Function

**Summary**  
Issues an execute transaction consisting of a specified string.

**Package**  
**win32**

**Signature**  
```
dde-execute-string conversation command &key errorp => result
```

**Arguments**  
- `conversation`: A conversation object.
- `command`: A string or symbol.
- `errorp`: A boolean.

**Values**  
- `result`: A boolean.

**Description**  
The function `dde-execute-string` issues an execute transaction consisting of the string `command`. No processing of the string is performed.

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-execute`
- `dde-execute-command`
- `dde-execute-string`
**dde-execute-string**

**Function**

### Summary
Issues an execute transaction consisting of a specified string on an automatically managed conversation.

### Package
`win32`

### Signature
```
dde-execute-string* service topic command &key errorp
connect-error-p new-conversation-p => result
```

### Arguments
- **service**: A symbol or string.
- **topic**: A symbol or string.
- **command**: A string or symbol.
- **errorp**: A boolean.
- **connect-error-p**: A boolean.
- **new-conversation-p**: A boolean.

### Values
- **result**: A boolean.

### Description
The function `dde-execute-string*` is similar to `dde-execute-string`, in that it issues an execute transaction consisting of the string `command`. However, the conversation across which `command` is issued is managed automatically. No processing of the string is performed.

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-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`
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>conversation</td>
<td>A conversation object.</td>
</tr>
<tr>
<td>item</td>
<td>A string or symbol.</td>
</tr>
<tr>
<td>format</td>
<td>A clipboard format specifier.</td>
</tr>
<tr>
<td>type</td>
<td>A keyword.</td>
</tr>
<tr>
<td>errorp</td>
<td>A boolean.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>result</td>
<td>A boolean.</td>
</tr>
</tbody>
</table>

The accessor `dde-item` performs a request transaction when read. It performs a poke transaction when set.

To illustrate, the following `dde-request` command

```lisp
(dde-request conversation item :format format :type type :errorp errorp)
```

can also be issued using `dde-item` as follows:

```lisp
(dde-item conversation item :FORMAT format :TYPE type :ERRORP errorp)
```

Similarly, the following `dde-poke` command

```lisp
(dde-poke conversation item data :format format :type type :errorp errorp)
```

can be issued using `dde-item` as follows:

```lisp
(setf (dde-item conversation item :format format :type type :errorp errorp) data)
```

except that the `format` always returns `data`.

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-item*` `dde-poke` `dde-request`
**dde-item**

*Accessor*

**Summary**
An accessor which can perform a request transaction or a poke transaction on an automatically managed conversation.

**Package**
win32

**Signature**
```
dde-item* service topic item &key format 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.
- **format** A clipboard format specifier.
- **type** A keyword.
- **errorp** A boolean.
- **connect-error-p** A boolean.
- **new-conversation-p** A boolean.

**Values**
- **result** A boolean.

**Description**
The accessor **dde-item** is similar to **dde-item**, and performs a request transaction when read. It performs a poke transaction when set.

To illustrate, the following **dde-request** command

```
(dde-request* service topic item :format format :type type :errorp errorp connect-error-p new-conversation-p)
```

can also be issued using **dde-item** as follows:

```
(dde-item* service topic item :FORMAT format :TYPE type :ERRORP errorp connect-error-p new-conversation-p)
```

Similarly, the following **dde-poke** command
(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**

**Function**

**Summary**
Issues a poke transaction on a conversation, to set the value of a specified item.

**Package**
win32

**Signature**
dde-poke conversation item data &key format type errorp => result

**Arguments**
- **conversation** A conversation object.
- **item** A string or symbol.
- **data** A string.
- **format** A clipboard format specifier.
- **type** A keyword.
- **errorp** A boolean.

**Values**
- **result** A boolean.

**Description**
The function **dde-poke** issues a poke transaction on *conversation* to set the value of the item specified by *item* to the value specified by *data*. The argument *item* 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.

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

Function

dde-poke*  

Summary  Issues a poke transaction on an automatically managed conversation, to set the value of a specified item.

Package  win32

Signature  
dde-poke* service topic item data &key format type errorp connect-error-p new-conversation-p => result

Arguments  

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>service</td>
<td>A symbol or string.</td>
</tr>
<tr>
<td>topic</td>
<td>A symbol or string.</td>
</tr>
<tr>
<td>item</td>
<td>A string or symbol.</td>
</tr>
<tr>
<td>data</td>
<td>A string.</td>
</tr>
<tr>
<td>format</td>
<td>A clipboard format specifier.</td>
</tr>
</tbody>
</table>
This chapter applies only to LispWorks for Windows

791

Description

The function dde-poke* is the same as dde-poke, except that conversations are managed automatically. The function issues a poke transaction to set the value of the item specified by item to the value specified by data. The argument item 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-poke.

If necessary, the function dde-poke* 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 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), 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`

**Function**

**Summary**

Issues a request transaction on a conversation for a specified item.

**Package**

`win32`

**Signature**

`dde-request conversation item &key format type errorp => result successp`

**Arguments**

- `conversation`: A conversation object.
- `item`: A string or symbol.
- `format`: A clipboard format specifier.
- `type`: A keyword.
- `errorp`: A boolean.

**Values**

- `result`: The return value of the transaction.
- `successp`: A boolean.

**Description**

The function `dde-request` issues a request transaction on `conversation` for the specified `item`. The argument `item` should
This chapter applies only to LispWorks for Windows

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**
win32

**Signature**
dde-request* service topic item &key format type errorp connect-error-p new-conversation-p => result successp

**Arguments**
- service: A symbol or string.
- topic: A symbol or string.
- item: A string or symbol.
- format: A clipboard format specifier.
- type: A keyword.
- errorp: A boolean.
- connect-error-p: A boolean.
- new-conversation-p: A boolean.

**Values**
- result: The return value of the transaction.

**Description**
The function dde-request* is similar to dde-request, except that conversations are managed automatically. The function issues a request transaction for the specified item. The argument item 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`
define-dde-client  

**Macro**

**Summary** Registers a client service.

**Package** *win32*

**Signature**  

\[
\text{define-dde-client } \text{name} \text{ &key service class } \Rightarrow \text{name}
\]

**Arguments**

- **name** A symbol.
- **service** A string.
- **class** A subclass of *dde-client-conversation*.

**Values**

- **name** A symbol.

**Description**

The macro `define-dde-client` defines a mapping from the symbol `name` to the DDE service name with which to establish a conversation, and the conversation class to use for this conversation. The argument `service` is a string which names the DDE service. It defaults to the print-name of `name`. The argument `class` is a subclass of *dde-client-conversation* which is used for all conversations with this service. It defaults to *dde-client-conversation*. Specifying a subclass allows various aspects of the behavior of the conversation to be specialized.

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.

If the macro executes successfully, the `name` of the DDE service is returned.

**See also**

- *dde-connect*
- *dde-disconnect*
- *with-dde-conversation*
with-dde-conversation

Macro

Summary
Dynamically binds a conversation to a server across a given body of code.

Package
win32

Signature
with-dde-conversation (conv service topic &key errorp new-conversation-p) &body body => result

Arguments
conv A conversation object.
service A symbol or string.
topic A symbol or string.
errorp A boolean.

new-conversation-p
A boolean.

body A list of Lisp forms.

Values
result A boolean.

Description
The macro with-dde-conversation dynamically binds a conversation with a server across the scope of a body of code specified by body. The argument conv is bound to a conversation with the server specified by service, and the topic specified by topic.

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.
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

dde-server-poke

Generic Function

Summary Called when a poke transaction is received.

Package win32

Signature dde-server-poke server topic item data &key format &allow-other-keys => successp

Arguments server A server object.
topic A topic object.
item A string.
data A string.
format A keyword.

Values successp A boolean.

Description The generic function dde-server-poke 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.
The WIN32 Package (including DDE)

This chapter applies only to LispWorks for Windows

Description

The generic function `dde-server-request` is called in response to a request transaction. A method specializing on the classes of `server` and `topic` should return the data in `item`.

The expected format of the data is given by `format`, which defaults to `: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 request fails, `nil` should be returned.

See also

`dde-poke`

`dde-request`

`dde-server-poke`

dde-server-topic

Generic Function

Summary

Called whenever a client attempts to connect to a server with a given topic.

Package

`win32`

Signature

`dde-server-topic` `server` `topic-name` `=>` `topic`

Arguments

`server` A server.

`topic-name` A string.

Values

`topic` A topic.

Description

The generic function `dde-server-topic` is called whenever a client attempts to make a connection to the server. The argument `topic-name` 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 \texttt{nil}.

\textbf{See also} \texttt{dde-server-topics} \\
\texttt{dde-topic-items}

\begin{description}
\item[\textbf{dde-server-topics}] \textit{Generic Function}
\item[Summary] Returns a list of the available general topics on a given server.
\item[Package] \texttt{win32}
\item[Signature] \texttt{dde-server-topics server => topic-list}
\item[Arguments] $server$ \hspace{1em} A server object.
\item[Values] $topic-list$ \hspace{1em} A list of strings.
\item[Description] The generic function \texttt{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 \texttt{define-dde-dispatch-topic}) should not be returned, as they are handled automatically by LispWorks. If you do not provide a \texttt{dde-server-topics} method, the default method returns \texttt{: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 \texttt{foo}, \texttt{foo.doc} and \texttt{c:\foo.doc} may all be acceptable strings for referring to the same topic; however \texttt{dde-server-topics} should return each topic once only.

The application must also provide a method on the \texttt{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

Arguments  

server  A server object.

topic  A topic object.

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

\texttt{define-dde-dispatch-topic name \&key server topic-name \=> name}

Arguments

\begin{itemize}
  \item \texttt{name} \hspace{1cm} A symbol.
  \item \texttt{server} \hspace{1cm} A server class.
  \item \texttt{topic-name} \hspace{1cm} A string.
\end{itemize}

Values

\texttt{name} \hspace{1cm} A symbol.

Description

The macro \texttt{define-dde-dispatch-topic} 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.

The name of the dispatching topic object is specified by \texttt{name}.

The topic is identified by the string \texttt{topic-name}.

The class of the server to attach the topic to is given by \texttt{server}.

The macro \texttt{define-dde-dispatch-topic} returns the name of the dispatching topic, \texttt{name}.

Use \texttt{define-dde-server-function} with the :topic option to define items for a dispatch topic.

Example:

\begin{verbatim}
(define-dde-dispatch-topic topic1 :server demo-server)
(define-dde-server-function (item1 :topic topic1) :request ()
  \ldots handle topic1.item1 request\ldots)
\end{verbatim}

See also

\begin{itemize}
  \item \texttt{dde-server-topic}
  \item \texttt{dde-server-topics}
  \item \texttt{define-dde-server-function}
\end{itemize}
**define-dde-server**

**Macro**

**Summary**
Defines a class for a Lisp DDE server.

**Package**
*win32*

**Signature**
```
define-dde-server class-name service-name => class-name
define-dde-server class-name superclasses slot-specs options => class-name
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>class-name</td>
<td>A class name.</td>
</tr>
<tr>
<td>service-name</td>
<td>A string.</td>
</tr>
<tr>
<td>superclasses</td>
<td>A list of superclasses.</td>
</tr>
<tr>
<td>slot-specs</td>
<td>The specifications for the class’ slots.</td>
</tr>
<tr>
<td>options</td>
<td>A keyword option.</td>
</tr>
</tbody>
</table>

**Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>class-name</td>
<td>A class name.</td>
</tr>
</tbody>
</table>

**Description**
The macro `define-dde-server` defines a class for a Lisp DDE server. The class inherits from `dde-server`.

The long form of the macro is similar to `defclass`, but with one extra option, `:service`, which is used to specify the service name string to which this server will respond.

The short form is provided to handle the common simple case; `class-name` is the name of the Lisp class to be defined, and `service-name` is the service name string to which this server will respond.

**Example**
The first example uses the short version of `define-dde-server` to define a class, called `lisp-server`, which has the service name “LISP”.

```
(define-dde-server lisp-server "LISP")
```
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 |
  :advisep advisep

- `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**

- name: A symbol.
- transaction: A keyword.
- server: A server object.
- topic-class: A topic class.
- topic: A symbol naming a dispatch topic.
- item: A string.
- format: A keyword.
- command: A string.
- result-type: A data type.
- advisep: A boolean.
- var: A variable.
- data-type: A data type.
- type-spec: A data type.
- form: A Lisp form.

**Values**

- name: A symbol.

**Description**

The macro `define-dde-server-function` is used to define a server function, called `name`, 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.

- 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, \texttt{:topic-class} defaults to \texttt{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 \texttt{dde-dispatch-topic}, previously defined by \texttt{define-dde-dispatch-topic}. This is the main use of dispatching topics. In this case \texttt{: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, \texttt{item} is a string defining the item name for which this definition should be invoked. It defaults to the capitalized print-name of \texttt{name}, with hyphens removed.

For request transactions, the \texttt{:format} option is used to specify the format understood. It defaults to \texttt{:text}. It can be specified as \texttt{:all}, in which case the \texttt{:format} binding may be used to determine the actual format requested (see below).

In the case of an execute transaction, \texttt{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 \texttt{name}, with hyphens removed.

The \texttt{execute-arg-bindings} are only used with execute transactions. They specify the arguments expected. \texttt{type-spec} should be one of \texttt{t}, \texttt{string}, \texttt{number}, \texttt{integer} or \texttt{float}. If not specified, \texttt{t} is assumed.

The \texttt{var-bindings} may appear anywhere in the binding list, and in any order. Binding variables to \texttt{:server} and \texttt{:topic} is useful with all transaction types. A \texttt{:server} binding causes the variable to be bound to the server object, whereas a \texttt{: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 data-type 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:

[code]
[command1(arg1, arg2, ...)] [command2(arg1, arg2, ...)]...
[/code]
Note that multiple commands may be packed into a single execute transaction. However, \texttt{dde-execute-command} does not currently generate such strings.

See also

\begin{itemize}
  \item \texttt{dde-execute-command}
  \item \texttt{define-dde-client}
  \item \texttt{define-dde-dispatch-topic}
  \item \texttt{define-dde-server}
\end{itemize}

\textbf{start-dde-server} \hspace{2cm} \textit{Function}

\textbf{Summary} \hspace{2cm} Creates and starts an instance of a DDE server.

\textbf{Package} \hspace{2cm} \texttt{win32}

\textbf{Signature} \hspace{2cm} \texttt{start-dde-server name \Rightarrow server}

\textbf{Arguments} \hspace{2cm} \begin{itemize}
  \item \texttt{name} \hspace{2cm} A DDE server class
\end{itemize}

\textbf{Values} \hspace{2cm} \begin{itemize}
  \item \texttt{server} \hspace{2cm} A server object
\end{itemize}

\textbf{Description} \hspace{2cm} The function \texttt{start-dde-server} creates an instance of a server of the class specified by \texttt{name} which then starts accepting transactions. If successful the function returns the server, otherwise \texttt{nil} is returned.

You need to call \texttt{start-dde-server} in a thread that will process Windows messages. This can either be done by using \texttt{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

\begin{verbatim}
(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"))))
\end{verbatim}

See also define-dde-server
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:

```c
int __stdcall InitLispWorks (int MilliTimeOut, void *BaseAddress, size_t ReserveSize)
```

On Linux/Macintosh/FreeBSD:
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 \texttt{deliver} and \texttt{save-image}.

\textbf{Note:} Once \texttt{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.

\textbf{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 \texttt{dll-added-files in deliver and save-image}. Such wrappers can be used to add calls to \texttt{InitLispWorks} before actually calling into Lisp.

\texttt{InitLispWorks} is defined in each LispWorks dynamic library. For information about creating a LispWorks dynamic library, see \texttt{deliver and save-image}. For an overview of LispWorks as a dynamic library, see the section “LispWorks as a dynamic library” in the \textit{LispWorks User Guide}.

\begin{itemize}
  \item \texttt{deliver}
  \item \texttt{LispWorksState}
  \item \texttt{save-image}
  \item \texttt{QuitLispWorks}
\end{itemize}

\textbf{LispWorksDlsym} \hfill \textit{C function}

\textbf{Summary} \hfill Returns the address of a foreign callable.

\textbf{Signature} \hfill On Windows:

\begin{verbatim}
void __stdcall *LispWorksDlsym (const char * name)
\end{verbatim}

On Linux/Macintosh/FreeBSD:

\begin{verbatim}
void *LispWorksDlsym (const char * name)
\end{verbatim}
The C function `LispWorksDlsym` returns the address of a foreign callable `name` which is defined in Lisp using `fli:define-foreign-callable`.

`LispWorksDlsym` first checks if the LispWorks dynamic library finished initializing, and if not uses `InitLispWorks` to initialize it (with `MilliTimeOut` 200). If this fails `LispWorksDlsym` returns NULL. When the LispWorks dynamic library is initialized, `LispWorksDlsym` returns the address of `name`, or NULL if it is not defined.

`LispWorksDlsym` 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`

---

### LispWorksState

**C function**

**Summary**

Returns the state of a LispWorks dynamic library.

**Signature**

On Windows:

```c
int __stdcall LispWorksState (int MilliTimeOut)
```

On Linux/Macintosh/FreeBSD:

```c
int LispWorksState (int MilliTimeOut)
```

**Description**

The C function `LispWorksState` returns the state of a LispWorks dynamic library.

`MilliTimeOut` specifies the time to wait in milliseconds if LispWorks is in the process of initialization.

If LispWorks has not been initialized, or has been quit by `QuitLispWorks`, `LispWorksState` returns -100. Otherwise, it returns the same values as `InitLispWorks`. In particular, if
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

Summary
Initializes a LispWorks dynamic library.

Signature
On Windows:

```c
int __stdcall SimpleInitLispWorks (void)
```

On Linux/Macintosh/FreeBSD:

```c
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**

**Summary**
Allows a LispWorks dynamic library to be unloaded.

**Signature**
On Windows:

```c
int __stdcall QuitLispWorks(int Force, int MilliTimeOut)
```

On Linux/Macintosh/FreeBSD:

```c
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

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-go 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
Index

C functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>InitLispWorks</td>
<td>811</td>
</tr>
<tr>
<td>LispWorksDlsym</td>
<td>814</td>
</tr>
<tr>
<td>LispWorksState</td>
<td>815</td>
</tr>
<tr>
<td>QuitLispWorks</td>
<td>817</td>
</tr>
<tr>
<td>SimpleInitLispWorks</td>
<td>816</td>
</tr>
<tr>
<td>cache-table-queries</td>
<td>482</td>
</tr>
<tr>
<td><em>cache-table-queries-default</em> variable</td>
<td>483</td>
</tr>
<tr>
<td>call-next-advice</td>
<td>313</td>
</tr>
<tr>
<td>calls-who</td>
<td>200</td>
</tr>
<tr>
<td>call-system</td>
<td>639</td>
</tr>
<tr>
<td>call-system-showing-output function</td>
<td>641</td>
</tr>
<tr>
<td>canonicalize-dspec</td>
<td>153</td>
</tr>
<tr>
<td>cd</td>
<td>200</td>
</tr>
<tr>
<td>cdr-assoc</td>
<td>644</td>
</tr>
<tr>
<td>change-directory</td>
<td>201</td>
</tr>
<tr>
<td>change-process-priority</td>
<td>403</td>
</tr>
<tr>
<td>char-external-code</td>
<td>185</td>
</tr>
<tr>
<td>check-fragmentation</td>
<td>202</td>
</tr>
<tr>
<td><em>check-network-server</em></td>
<td>645</td>
</tr>
<tr>
<td>class options</td>
<td></td>
</tr>
<tr>
<td>:base-table</td>
<td>502</td>
</tr>
<tr>
<td>:extra-initargs</td>
<td>3, 5, 78</td>
</tr>
<tr>
<td>:optimize-slot-access</td>
<td>14, 78</td>
</tr>
<tr>
<td>classes</td>
<td></td>
</tr>
<tr>
<td>buffered-stream</td>
<td>603</td>
</tr>
<tr>
<td>fund callable-standard-object</td>
<td>6</td>
</tr>
<tr>
<td>fundamental-binary-input-stream</td>
<td>605</td>
</tr>
<tr>
<td>fundamental-binary-output-stream</td>
<td>606</td>
</tr>
<tr>
<td>fundamental-binary-stream</td>
<td>606</td>
</tr>
<tr>
<td>fundamental-character-input-stream</td>
<td>607</td>
</tr>
<tr>
<td>fundamental-character-output-stream</td>
<td>607</td>
</tr>
<tr>
<td>fundamental-character-stream</td>
<td>607</td>
</tr>
<tr>
<td>fundamental-input-stream</td>
<td>609</td>
</tr>
<tr>
<td>fundamental-output-stream</td>
<td>610</td>
</tr>
<tr>
<td>fundamental-stream</td>
<td>610</td>
</tr>
<tr>
<td>lob-stream</td>
<td>523</td>
</tr>
<tr>
<td>socket-error</td>
<td>45</td>
</tr>
<tr>
<td>socket-stream</td>
<td>45</td>
</tr>
<tr>
<td>ssl-closed</td>
<td>53</td>
</tr>
<tr>
<td>ssl-condition</td>
<td>53</td>
</tr>
<tr>
<td>ssl-error</td>
<td>54</td>
</tr>
<tr>
<td>ssl-failure</td>
<td>54</td>
</tr>
<tr>
<td>ssl-x509-lookup</td>
<td>55</td>
</tr>
</tbody>
</table>

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 882
    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-adviser-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
declare alias 74
  :explain 74
  invisible-frame 74
  lambda-list 74
  values 74
  special form 74
declai..
Index

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-style 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
Index

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-algorithm* 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-option function 660
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
Index

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-advise-start 770
dde-advise-start* 772
dde-advise-stop 774
dde-advise-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
dissmiss-splash-screen 751
dl-quits 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-undefined 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
Index

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-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/ 680
int32/= 680
int32<< 682
int32<= 682
int32=> 682
int32>= 683
int32> 683
int32>= 684
int32>> 685
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-lognot 692
int32-logor 692
int32-logorc1 692
int32-logorc2 693
int32-logtest 694
int32-logxor 694
int32-minusp 695
int32-plusp 695
int32-to-integer 696
int32-xor 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-classes 521
list-numeric-p 522
list-numeric-p 523
load-all-patches 366
load-data-file 699
load-logical-pathname-translations 94
Index

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-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-ge 705
memory-growth-margin 707
merge 102
merge-of-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
openssl-version 35
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-locilator 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
precompile-regexp 374
open-serial-port 469
ora-lob-lob-append 531
ora-lob-lob-assign 532
ora-lob-lob-char-set-form 532
ora-lob-lob-char-set-id 533
ora-lob-lob-close 534
ora-lob-lob-copy 535
ora-lob-lob-create-empty 536
ora-lob-lob-create-temporary 537
ora-lob-lob-disable-buffering 538
ora-lob-lob-element-type 539
ora-lob-lob-enable-buffering 540
ora-lob-lob-env-handle 540
ora-lob-lob-erase 541
ora-lob-lob-file-close 542
ora-lob-lob-file-close-all 543
ora-lob-lob-file-exists 543
ora-lob-lob-file-get-name 544
ora-lob-lob-file-is-open 545
ora-lob-lob-file-open 546
ora-lob-lob-file-set-name 546
ora-lob-lob-flush-buffer 547
ora-lob-lob-free 548
ora-lob-lob-free-temporary 549
ora-lob-lob-get-buffer 550
ora-lob-lob-get-chunk-size 552
ora-lob-lob-get-length 553
ora-lob-lob-internal-lob-p 554
ora-lob-lob-is-equal 554
ora-lob-lob-is-open 555
ora-lob-lob-is-temporary 555
ora-lob-load-from-file 556
ora-lob-lob-locilator 557
ora-lob-lob-locator-is-init 558
ora-lob-lob-open 559
ora-lob-lob-read-buffer 560
ora-lob-lob-read-foreign-buffer 563
ora-lob-lob-read-into-plain-file 562
ora-lob-lob-svc-ctx-handle 564
ora-lob-lob-trim 565
ora-lob-lob-write-buffer 566, 568
ora-lob-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
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 733
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 267
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-reference-who 250
set-registry-value 767
set-sql-reader-syntax-state 573
set-serial-port-state 475
Index

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-topics 800
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
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
Index

*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
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+ symbol macro 678
int32-aref function 685
int32-logand function 686
int32-logandc1 function 687
int32-logandc2 function 687
int32-logb function 688
int32-logeqv function 689
int32-logior function 690
int32-lognand function 690
int32-lognot function 691
int32-lognor 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-zero? 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-depec-alias 155
define-depec-class 156
define-form-parser 159
define-top-loop-command 651
depackage 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 379
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

Index
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
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-frees 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-locatoe 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.deepeuc 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
Index

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 334, 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 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
set-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
Index

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
SimpleInitializeWorks C function 816
simple-int32-vector type 743
simple-process-p function 457
simple-text-string type 387
simple-text-string-p function 387
single-float type 113
simple-do-query macro 579
SimpleInitializeLispWorks C function 816
simple-int32-vector type 743
simple-process-p function 457
simple-text-string type 387
simple-text-string-p function 387
simple-do-query macro 579
SimpleInitializeLispWorks C function 816
simple-int32-vector type 743
simple-process-p function 457
simple-text-string type 387
simple-text-string-p function 387
single-float type 113
single-form-form-parser function 179
single-form-with-options-form-parser function 180
: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* 412
    *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-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 accessor 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-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
Index

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-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 611
stream-fill-buffer generic function 614
stream-finish-output generic function 615
stream-force-output 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-write-char generic function 627
stream-write-byte generic function 629
stream-write-line generic function 629
stream-write-sequence generic function 630
stream-write-timeout accessor 46
string type 120
string-list generic 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-multiprocessing* 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
Index

*trace-print-pretty* variable 291
*trace-verbose* variable 292
tracing-enabled-p function 182
true function 392
truename function 129
try-compact-in-generation function 293
try-move-in-generation function 295
tty 255
:typinitarg 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
  307
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

837
Index

*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-equiv* 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-alloca-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