
CAPI Reference Manual

Version 5.1



Copyright and Trademarks

LispWorks CAPI Reference Manual

Version 5.1

March 2008

Copyright © 2008 by LispWorks Ltd.

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

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

LispWorks and KnowledgeWorks are registered trademarks of LispWorks Ltd.

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

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

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

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

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

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

ICU License - ICU 1.8.1 and later

COPYRIGHT AND PERMISSION NOTICE

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

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

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

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

US Government Restricted Rights

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

Address

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

Telephone

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

Fax

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

www.lispworks.com

Contents

Preface xxi

1 CAPI Reference Entries 1

abort-callback 1
abort-dialog 2
abort-exit-confirmer 3
accepts-focus-p 4
activate-pane 5
append-items 6
apply-in-pane-process 6
arrow-pinboard-object 7
attach-interface-for-callback 9
attach-simple-sink 10
attach-sink 11
beep-pane 12
button 13
button-panel 18
calculate-constraints 23
calculate-layout 24

callbacks 24
call-editor 27
capi-object 28
capi-object-property 29
check-button 30
check-button-panel 31
choice 31
choice-selected-item 35
choice-selected-item-p 37
choice-selected-items 37
choice-update-item 39
clipboard 40
clipboard-empty 41
clone 42
cocoa-default-application-interface 42
cocoa-view-pane 45
cocoa-view-pane-view 47
collect-interfaces 48
collection 49
collection-find-next-string 52
collection-find-string 53
collection-last-search 54
collection-search 54
collector-pane 55
color-screen 56
column-layout 57
component-name 60
confirm-quit 60
confirm-yes-or-no 61
confirmer-pane 62
contain 63
convert-relative-position 64
convert-to-screen 65
count-collection-items 67
current-dialog-handle 68
current-document 69
current-pointer-position 70

current-printer 71
default-editor-pane-line-wrap-marker 71
default-library 72
define-command 72
define-interface 74
define-layout 80
define-ole-control-component 81
define-menu 83
destroy 83
detach-simple-sink 84
detach-sink 85
display 86
display-dialog 88
display-errors 91
display-message 91
display-message-for-pane 92
display-pane 92
display-popup-menu 94
display-replacable-dialog 95
display-tooltip 96
docking-layout 97
docking-layout-pane-docked-p 101
docking-layout-pane-visible-p 101
document-container 102
document-frame 103
double-headed-arrow-pinboard-object 106
double-list-panel 107
drag-pane-object 109
draw-metafile 111
draw-metafile-to-image 111
drawn-pinboard-object 113
draw-pinboard-object 114
draw-pinboard-object-highlighted 115
draw-pinboard-object-unhighlighted 115
drop-object-allows-drop-effect-p 116
drop-object-drop-effect 116
drop-object-get-object 117

drop-object-pane-x 118
drop-object-pane-y 119
drop-object-provides-format 120
echo-area-cursor-inactive-style 120
echo-area-pane 121
editor-cursor-color 121
editor-cursor-active-style 122
editor-cursor-drag-style 122
editor-cursor-inactive-style 123
editor-pane 123
editor-pane-blink-rate 131
editor-pane-buffer 131
editor-pane-native-blink-rate 132
editor-pane-selected-text 133
editor-pane-selected-text-p 133
editor-pane-stream 134
editor-window 134
element 135
element-container 140
element-interface-for-callback 141
element-screen 141
ellipse 142
ensure-area-visible 142
ensure-interface-screen 143
execute-with-interface 143
execute-with-interface-if-alive 144
exit-confirmer 145
exit-dialog 146
expandable-item-pinboard-object 147
extended-selection-tree-view 147
filtering-layout 148
filtering-layout-match-object-and-exclude-p 152
find-graph-edge 152
find-graph-node 153
find-interface 154
find-string-in-collection 155
force-screen-update 155

force-update-all-screens 156
foreign-owned-interface 156
form-layout 157
free-metafile 158
free-sound 158
get-collection-item 159
get-constraints 159
get-horizontal-scroll-parameters 160
get-page-area 161
get-printer-metrics 162
get-scroll-position 163
get-vertical-scroll-parameters 164
graph-edge 165
graph-node 165
graph-node-children 166
graph-object 166
graph-pane 167
graph-pane-add-graph-node 171
graph-pane-delete-object 172
graph-pane-delete-objects 172
graph-pane-delete-selected-objects 173
graph-pane-direction 173
graph-pane-edges 174
graph-pane-nodes 175
graph-pane-object-at-position 175
graph-pane-select-graph-nodes 176
graph-pane-update-moved-objects 176
grid-layout 177
hide-interface 182
hide-pane 182
highlight-pinboard-object 183
image-list 183
image-pinboard-object 185
image-set 186
install-postscript-printer 187
installed-libraries 189
interactive-pane 190

- interactive-pane-execute-command 191
- interface 192
 - interface-display 207
 - interface-display-title 208
 - interface-editor-pane 209
 - interface-extend-title 209
 - interface-geometry 210
 - interface-iconified-p 211
 - interface-keys-style 211
 - interface-match-p 214
 - interface-menu-groups 214
 - interface-reuse-p 215
 - interface-visible-p 216
- interpret-description 217
- invalidate-pane-constraints 218
- invoke-command 218
- invoke-untranslated-command 219
- item 220
- itemp 221
- item-pinboard-object 222
- labelled-arrow-pinboard-object 222
- labelled-line-pinboard-object 223
- layout 224
- line-pinboard-object 226
- line-pinboard-object-coordinates 227
- list-panel 228
- list-panel-enabled 233
- list-view 234
- listener-pane 240
- listener-pane-insert-value 240
- load-cursor 241
- load-sound 243
- locate-interface 244
- lower-interface 246
- make-container 246
- make-docking-layout-controller 247
- make-foreign-owned-interface 248

make-general-image-set 249
make-icon-resource-image-set 250
make-image-locator 251
make-menu-for-pane 252
make-pane-popup-menu 253
make-resource-image-set 255
make-scaled-general-image-set 256
make-scaled-image-set 257
make-sorting-description 258
manipulate-pinboard 260
map-collection-items 263
map-pane-children 263
map-pane-descendant-children 266
map-typeout 266
maximum-moving-objects-to-track-edges 267
menu 267
menu-component 272
menu-item 274
menu-object 279
merge-menu-bars 282
message-pane 283
modify-editor-pane-buffer 284
mono-screen 285
move-line 285
multi-column-list-panel 286
multi-line-text-input-pane 290
ole-control-add-verbs 291
ole-control-close-object 292
ole-control-component 292
ole-control-doc 294
ole-control-frame 295
ole-control-i-dispatch 296
ole-control-insert-object 297
ole-control-ole-object 298
ole-control-pane 298
ole-control-pane-frame 301
ole-control-pane-simple-sink 302

- ole-control-user-component 302
- option-pane 303
- output-pane 306
- over-pinboard-object-p 316
- page-setup-dialog 316
- pane-adjusted-offset 317
- pane-adjusted-position 318
- pane-close-display 320
- pane-got-focus 321
- pane-has-focus-p 321
- pane-initial-focus 322
- pane-popup-menu-items 323
- pane-string 325
- parse-layout-descriptor 326
- password-pane 327
- play-sound 328
- pinboard-layout 329
- pinboard-object 331
- pinboard-object-at-position 335
- pinboard-object-graphics-arg 336
- pinboard-object-overlap-p 337
- pinboard-pane-position 338
- pinboard-pane-size 339
- popup-confirmer 340
- popup-menu-button 348
- print-capi-button 349
- print-collection-item 350
- print-dialog 351
- print-editor-buffer 352
- print-file 353
- print-rich-text-pane 354
- print-text 355
- printer-configuration-dialog 355
- printer-metrics 356
- *ppd-directory* 358
- printer-port-handle 358
- printer-port-supports-p 359

printer-search-path 360
process-pending-messages 361
progress-bar 361
prompt-for-color 362
prompt-for-confirmation 363
prompt-for-directory 364
prompt-for-file 366
prompt-for-files 369
prompt-for-font 370
prompt-for-form 371
prompt-for-forms 373
prompt-for-integer 374
prompt-for-items-from-list 375
prompt-for-number 376
prompt-for-string 377
prompt-for-symbol 379
prompt-for-value 381
prompt-with-list 382
prompt-with-message 384
push-button 385
push-button-panel 386
quit-interface 388
radio-button 389
radio-button-panel 390
raise-interface 391
range-pane 392
read-sound-file 393
rectangle 393
redisplay-collection-item 394
redisplay-interface 394
redisplay-menu-bar 395
redraw-pinboard-layout 396
redraw-pinboard-object 396
reinitialize-interface 397
remove-capi-object-property 397
remove-items 398
replace-dialog 399

replace-items 400
report-active-component-failure 401
reuse-interfaces-p 402
rich-text-pane 403
rich-text-pane-character-format 405
rich-text-pane-operation 406
rich-text-pane-paragraph-format 410
rich-text-version 411
right-angle-line-pinboard-object 411
row-layout 412
screen 414
screen-active-interface 416
screen-active-p 417
screen-logical-resolution 417
screen-internal-geometry 418
screens 418
scroll 419
scroll-bar 421
search-for-item 423
selection 423
selection-empty 424
set-application-interface 425
set-button-panel-enabled-items 426
set-clipboard 426
set-confirm-quit-flag 428
set-default-editor-pane-blink-rate 428
set-default-interface-prefix-suffix 429
set-drop-object-supported-formats 431
set-geometric-hint 432
set-hint-table 432
set-horizontal-scroll-parameters 433
set-pane-focus 434
set-object-automatic-resize 435
set-rich-text-pane-character-format 439
set-rich-text-pane-paragraph-format 442
set-selection 444
set-printer-metrics 445

set-printer-options 446
set-text-input-pane-selection 447
set-top-level-interface-geometry 447
set-vertical-scroll-parameters 448
shell-pane 450
show-interface 451
show-pane 452
simple-layout 452
simple-network-pane 453
simple-pane 453
simple-pane-handle 460
simple-pane-visible-height 461
simple-pane-visible-size 461
simple-pane-visible-width 462
simple-pinboard-layout 463
simple-print-port 464
slider 464
sort-object-items-by 466
sorted-object 466
sorted-object-sort-by 467
start-gc-monitor 468
stop-gc-monitor 469
switchable-layout 469
switchable-layout-switchable-children 471
tab-layout 471
tab-layout-panes 474
tab-layout-visible-child 475
text-input-choice 475
text-input-pane 476
text-input-pane-complete-text 485
text-input-pane-copy 486
text-input-pane-cut 486
text-input-pane-delete 487
text-input-pane-paste 487
text-input-pane-selected-text 488
text-input-pane-selection 489
text-input-pane-selection-p 489

text-input-range 490
title-pane 492
titled-menu-object 493
titled-object 494
titled-pinboard-object 497
toolbar 500
toolbar-button 502
toolbar-component 507
toolbar-object 509
top-level-interface 509
top-level-interface-display-state 510
top-level-interface-geometry 511
top-level-interface-geometry-key 512
top-level-interface-p 514
top-level-interface-save-geometry-p 514
tracking-pinboard-layout 514
tree-view 517
tree-view-ensure-visible 526
tree-view-expanded-p 526
tree-view-item-checkbox-status 527
tree-view-item-children-checkbox-status 527
tree-view-update-an-item 528
tree-view-update-item 529
undefine-menu 529
unhighlight-pinboard-object 530
uninstall-postscript-printer 530
unmap-typeout 531
update-all-interface-titles 531
update-interface-title 532
update-pinboard-object 532
update-screen-interface-titles 533
update-screen-interfaces-hooks 533
update-toolbar 534
with-atomic-redisplay 534
with-busy-interface 535
with-dialog-results 536
with-document-pages 538

- with-external-metafile 538
- with-geometry 540
- with-internal-metafile 542
- with-output-to-printer 543
- with-page 544
- with-page-transform 545
- with-print-job 545
- with-random-typeout 546
- wrap-text 547
- wrap-text-for-pane 547
- x-y-adjustable-layout 549

2 GP Reference Entries 551

- analyze-external-image 551
- apply-rotation 552
- apply-scale 552
- apply-translation 553
- augment-font-description 553
- clear-external-image-conversions 554
- clear-graphics-port 555
- clear-graphics-port-state 555
- clear-rectangle 556
- compress-external-image 556
- compute-char-extents 557
- convert-external-image 557
- convert-to-font-description 558
- copy-external-image 559
- copy-pixels 559
- copy-transform 560
- create-pixmap-port 561
- *default-image-translation-table* 562
- define-font-alias 562
- destroy-pixmap-port 563
- dither-color-spec 563
- draw-arc 564
- draw-arcs 565
- draw-character 566

draw-circle 566
draw-ellipse 567
draw-image 568
draw-line 569
draw-lines 570
draw-point 571
draw-points 571
draw-polygon 572
draw-polygons 573
draw-rectangle 574
draw-rectangles 575
draw-string 576
ensure-gdiplus 576
external-image 578
external-image-color-table 578
external-image-color-table 579
externalize-image 580
find-best-font 581
find-matching-fonts 581
font-description 582
font-description-attributes 583
font-description-attribute-value 583
font-fixed-width-p 584
free-image 584
free-image-access 585
get-bounds 585
get-character-extent 586
get-char-ascent 587
get-char-descent 587
get-char-width 588
get-enclosing-rectangle 588
get-font-ascent 589
get-font-average-width 589
get-font-descent 590
get-font-height 590
get-font-width 591
get-graphics-state 591

get-origin 592
get-string-extent 593
get-transform-scale 593
graphics-port-transform 594
image 594
image-access-pixel 595
image-access-pixels-from-bgra 596
image-access-pixels-to-bgra 597
image-access-transfer-from-image 598
image-access-transfer-to-image 599
image-freed-p 600
image-loader 600
image-translation 601
initialize-dithers 601
inset-rectangle 602
inside-rectangle 603
invalidate-rectangle 603
invert-transform 604
list-all-font-names 604
load-icon-image 605
load-image 607
make-dither 609
make-font-description 610
make-graphics-state 611
make-image 612
make-image-access 613
make-image-from-port 614
make-sub-image 615
make-transform 616
merge-font-descriptions 616
offset-rectangle 617
ordered-rectangle-union 618
pixblt 619
pixmap-port 619
port-height 620
port-string-height 620
port-string-width 621

port-width 621
postmultiply-transforms 622
premultiply-transforms 622
read-and-convert-external-image 623
read-external-image 624
rectangle-bind 625
rectangle-bottom 626
rectangle-height 626
rectangle-left 627
rectangle-right 627
rectangle-top 628
rectangle-union 628
rectangle-width 629
rect-bind 630
register-image-load-function 630
register-image-translation 631
reset-image-translation-table 632
separation 633
set-default-image-load-function 633
set-graphics-port-coordinates 634
set-graphics-state 635
transform 635
transform-area 636
transform-distance 636
transform-distances 637
transform-is-rotated 637
transform-point 638
transform-points 638
transform-rect 639
undefine-font-alias 640
union-rectangle 640
unit-transform 641
unit-transform-p 641
unless-empty-rect-bind 642
untransform-distance 642
untransform-distances 643
untransform-point 643

- untransform-points 644
- validate-rectangle 645
- with-dither 646
- with-graphics-mask 646
- with-graphics-rotation 647
- with-graphics-scale 648
- with-graphics-state 649
- with-graphics-transform 650
- with-graphics-translation 651
- with-inverse-graphics 651
- without-relative-drawing 652
- with-pixmap-graphics-port 652
- with-transformed-area 653
- with-transformed-point 654
- with-transformed-points 655
- with-transformed-rect 656
- write-external-image 656

3 COLOR Reference Entries 659

- apropos-color-alias-names 659
- apropos-color-names 660
- apropos-color-spec-names 661
- color-alpha 662
- color-*<component>* 662
- *color-database* 663
- color-level 664
- color-model 665
- colors= 665
- convert-color 666
- define-color-alias 667
- define-color-models 668
- delete-color-translation 669
- ensure-*<command>* 670
- get-all-color-names 671
- get-color-alias-translation 672
- get-color-spec 673
- load-color-database 674

make-gray 675
make-hsv 676
make-rgb 677
read-color-db 678
unconvert-color 679

Index 681

Preface

This manual contains reference entries for the functions, classes, macros and accessors in the `capi` package, and the `graphics-ports` and `color` packages. Entries are listed alphabetically, and the typographical conventions used are similar to those used in *Common Lisp: the Language* (2nd Edition). Further details on the conventions used are given below. For a more tutorial approach to the CAPI with further examples see the *LispWorks CAPI User Guide*.

Note: Although the `graphics-ports` and `color` packages are not strictly part of the CAPI, they are included in this manual because the functionality is usually called from CAPI elements such as output panes. Please also see the relevant chapters in the *LispWorks CAPI User Guide* for further information on Graphics Ports and the LispWorks Color System.

Conventions used for reference entries

Each entry is headed by the symbol name and type, followed by a number of fields providing further details. These fields consist of a subset of the following: “Package”, “Summary”, “Signature”, “Arguments”, “Values”, “Initial Value”, “Superclasses”, “Subclasses”, “Initargs”, “Accessors”, “Readers”, “Compatibility Note”, “Description”, “Examples”, and “See also”.

The default package containing each symbol is the `capi` package in the CAPI reference chapter, and so on, unless stated otherwise in the “Package” section of an entry.

Throughout, variable arguments are italicised. They look *like this* in the Description.

Throughout, exported symbols are printed `like-this`. The package qualifier is usually omitted, as if the current package is `capi` (or `graphics-ports` or `color`.)

Entries with a long “Description” section usually have as their first field a short “Summary” providing a quick overview of the purpose of the symbol being described.

The “Signature” section provides details of the arguments taken by the functions and macros.

The “Subclasses” section of each CAPI class entry lists the external subclasses, though not subclasses of those.

The “Superclasses” sections of each CAPI class entry lists the external superclasses, though not superclasses of those.

The “Initargs” section describes the initialization arguments of the class. Initargs of superclasses are also valid.

Note: in LispWorks4.2 and previous versions, the “Initargs” section was headed “Slots”.

Examples of the use of commands are given under the “Examples” heading. The code is written with explicit package qualifiers such as `capi:interface`, so that it can be run as-is, regardless of the current package. Some example files can also be found in your installation directory under `examples/capi/`.

Finally, the “See also” section provides a reference to other related symbols.

The LispWorks manuals

The LispWorks manual set comprises the following books:

- The *LispWorks User Guide* describes the features and tools available in LispWorks.
- The *LispWorks Reference Manual* contains detailed information on all functions, macros, variables and classes available in LispWorks, in alphabetical order.

- The *Common LispWorks User Guide* describes Common LispWorks, the user interface for LispWorks. Common LispWorks is a set of windowing tools that let you develop and test Common Lisp code more easily and quickly.
- The *LispWorks Editor User Guide* describes the keyboard commands and programming interface to the Common LispWorks editor tool.
- The *LispWorks CAPI User Guide* and the *LispWorks CAPI Reference Manual* describe the CAPI. This is a library of classes, functions, and macros for developing graphical user interfaces for your applications. The *LispWorks CAPI User Guide* is a tutorial guide to the CAPI, and the *LispWorks CAPI Reference Manual* is an in-depth reference text.
- The *LispWorks Foreign Language Interface User Guide and Reference Manual* explains how you can use C source code in applications developed using LispWorks.
- The *LispWorks Delivery User Guide* describes how you can deliver working, standalone versions of your LispWorks applications for distribution to your customers.
- The *KnowledgeWorks and Prolog User Guide* describes the LispWorks toolkit for building knowledge-based systems. Prolog is a logic programming system within Common Lisp.
- The *Common Lisp Interface Manager 2.0 User's Guide* describes the portable Lisp-based GUI toolkit.

These books are all available in online form, in both HTML format and PDF format. Also in PDF and plain text format is:

- The *LispWorks Release Notes and Installation Guide* which contains notes explaining how to install LispWorks and get it running. It also contains a set of release notes which lists new features and any last minute issues that could not be included in the main manual set.

Commands in the **Help** menu of any of the Common LispWorks tools give you direct access to the online documentation in HTML format, using the HTML browser that is supplied with LispWorks. Details of how to use these commands can be found in the *Common LispWorks User Guide*.

Documentation is also provided in PDF form. You can use Adobe® Reader® to browse the PDF documentation online or to print it. Adobe Reader is available from Adobe's web site, <http://www.adobe.com/>.

Please let us know at lisp-support@lispworks.com if you find any mistakes in the LispWorks documentation, or if you have any suggestions for improvements.

1

CAPI Reference Entries

The following chapter documents symbols exported from the `capi` package.

abort-callback	<i>Function</i>
Summary	Aborts out of the context of the current callback.
Package	<code>capi</code>
Signature	<code>abort-callback &optional <i>always-abort</i></code>
Arguments	<i>always-abort</i> A generalized boolean.
Description	<p>The function <code>abort-callback</code> aborts out of the context of the current callback, returning <code>nil</code> when it is relevant (for example in an interface <i>confirm-destroy-callback</i>).</p> <p>If called outside the context of a callback, if <i>always-abort</i> is <code>t</code> then <code>abort-callback</code> calls <code>(abort)</code>, otherwise it just returns.</p> <p>The default value of <i>always-abort</i> is <code>t</code>.</p>

See also `callbacks`
 `interface`

abort-dialog

Function

Summary The `abort-dialog` function aborts the current dialog.

Package `capi`

Signature `abort-dialog &rest ignored-args`

Description This function is used to abort the current dialog. For example, it can be made a selection callback from a **Cancel** button so that pressing the button aborts the dialog. In a similar manner the complementary function `exit-dialog` can be used as a callback for an **OK** button.

If there is no current dialog then `abort-dialog` does nothing and returns `nil`. If there is a current dialog then `abort-dialog` either returns non-`nil` or does a non-local exit. Therefore code that depends on `abort-dialog` returning must be written carefully. Constructs like this can be useful:

```
(unless (capi:abort-dialog)
  (foo))
```

Above, *foo* will be called only if there is no current dialog.

It is not useful to do either:

```
(when (capi:abort-dialog)
  (foo))
```

OR

```
(progn
  (capi:abort-dialog)
  (foo))
```

as in both cases it is not well-defined whether *foo* will be called if there is a current dialog.

Example

```
(capi:display-dialog
  (capi:make-container
    (make-instance 'capi:push-button
      :text "Cancel"
      :callback 'capi:abort-dialog)
    :title "Test Dialog"))
```

Also see the examples in the directory
`examples/capi/dialogs/`.

See also

```
exit-dialog
display-dialog
popup-confirmer
interface
```

abort-exit-confirmer

Function

Summary Aborts the exiting of a dialog.

Package `capi`

Signature `abort-exit-confirmer`

Description The function `abort-exit-confirmer` can be used to abort the exiting of a confirmer. It can be used in the *ok-function* of a confirmer, to abort the exit and return to the dialog.

If `abort-exit-confirmer` is called outside the exiting of a confirmer, it does nothing.

Example This example asks the user for a string. If the string is longer than 20 characters, it confirms with the user that they really want such a long string, and if they do not it returns to the dialog.

```
(capi:popup-confirmer
 (make-instance 'capi:text-input-pane)
 "New Name"
 :value-function 'capi:text-input-pane-text
 :ok-function
 #'(lambda (value)
      (when (and (> (length value) 20)
                (not (capi:prompt-for-confirmation
                      "Name is very long. Use it?")))
          (capi:abort-exit-confirmer))
      value))
```

See also `popup-confirmer`

accepts-focus-p

Generic Function

Summary Determines if an element accepts the focus.

Package `capi`

Signature `accepts-focus-p element => result`

Arguments *element* A CAPI element.

Values *result* A boolean.

Description Determines if the element *element* accepts the focus for user input, and controls tabstops.

The method on `element` uses the value of the *accepts-focus-p* slot, but methods some subclasses override this.

`accepts-focus-p` also influences whether a pane is a tabstop. On Microsoft Windows a pane acts as a tabstop if and only if the function `accepts-focus-p` returns true and the `element` *accepts-focus-p* initarg value is `:force`. On Motif and Cocoa, a pane acts as a tabstop if and only if the function `accepts-focus-p` returns true.

See also `element`
`pane-has-focus-p`
`set-object-automatic-resize`

activate-pane

Function

Summary The `activate-pane` function gives the focus to a pane and brings the window containing it to the front.

Package `capi`

Signature `activate-pane` *pane*

Description This brings the window containing *pane* to the front, and gives the focus to the pane (or a sensible alternative inside the same interface if that pane cannot accept the focus).

Example This example demonstrates how to swap the focus from one window to another.

```
(setq text-input-pane
      (capi:contain (make-instance
                    'capi:text-input-pane)))

(setq button
      (capi:contain (make-instance
                    'capi:push-button
                    :text "Press Me")))

(capi:activate-pane text-input-pane)

(capi:activate-pane button)
```

See also `hide-interface`
`raise-interface`
`set-object-automatic-resize`
`show-interface`
`quit-interface`
`simple-pane`

append-items*Generic Function*

Summary	Adds to the items in a collection.
Signature	<code>append-items collection new-items</code>
Arguments	<i>collection</i> A collection. <i>new-items</i> A sequence.
Description	<p>The generic function <code>append-items</code> adds the items in <i>new-items</i> to the collection <i>collection</i>.</p> <p>This is logically equivalent to recalculating the collection items and calling <code>(setf collection-items)</code>. However, <code>append-items</code> is more efficient and causes less flickering on screen.</p> <p><code>append-items</code> can only be used when the <i>collection</i> has the default <i>items-get-function</i> <code>svref</code>.</p>
See also	<code>collection</code> <code>remove-items</code> <code>replace-items</code>

apply-in-pane-process*Function*

Summary	Applies a function in the process associated with a pane.
Package	<code>capi</code>
Signature	<code>apply-in-pane-process pane function &rest args => nil</code>
Description	<p>The function <code>apply-in-pane-process</code> applies <i>function</i> to <i>args</i> in the process that is associated with <i>pane</i>. This is required when <i>function</i> modifies <i>pane</i> or changes how it is displayed. If <i>pane</i> has not been displayed yet, then <i>function</i> is called immediately.</p>

Note: All accesses (reads as well as writes) on a pane should be performed in the panes' process. Within a callback on the pane's interface this happens automatically, but `apply-in-pane-process` is a useful utility in other circumstances.

Example Editor commands must be called in the correct process:

```
(setq editor
  (capi:contain
    (make-instance 'capi:editor-pane
      :text "Once upon a time...")))

(capi:apply-in-pane-process
  editor 'capi:call-editor editor "End Of Buffer")

(capi:apply-in-pane-process
  editor 'capi:call-editor editor "Beginning Of Buffer")
```

arrow-pinboard-object

Class

Summary	A <code>pinboard-object</code> that draws itself as an arrow.
Package	<code>capi</code>
Superclasses	<code>line-pinboard-object</code>
Subclasses	<code>double-headed-arrow-pinboard-object</code> <code>labelled-arrow-pinboard-object</code>
Initargs	<code>:head</code> A keyword specifying the position of the arrowhead on the line. <code>:head-direction</code> A keyword specifying the direction of the arrowhead. <code>:head-length</code> The length of the arrowhead. <code>:head-breadth</code> The breadth of the arrowhead, or <code>nil</code> . <code>:head-graphics-args</code> A graphics args plist.

Description

An instance of the class `arrow-pinboard-object` is a `pinboard-object` that draws itself as an arrow.

head must be `:end`, `:middle` or `:start`. The default is `:end`.

head-direction must be `:forwards`, `:backwards` or `:both`. The default is `:forwards`.

head-length is the length of the arrowhead in pixels. It defaults to 12.

head-breadth is the breadth of the arrowhead in pixels, or `nil` which means that the breadth is half of *head-length*. The default is `nil`.

head-graphics-args is a plist of graphics state parameters and values used when drawing the arrow head. For information about the graphics state, see the section "Graphics State" in the *LispWorks CAPI User Guide*.

Example

```
(capi:contain
 (make-instance
  'capi:pinboard-layout
  :description
  (list
   (make-instance 'capi:arrow-pinboard-object
                  :start-x 5 :start-y 10
                  :end-x 105 :end-y 60 )
   (make-instance 'capi:arrow-pinboard-object
                  :start-x 5 :start-y 110
                  :end-x 105 :end-y 160
                  :head :middle)
   (make-instance 'capi:arrow-pinboard-object
                  :start-x 5 :start-y 210
                  :end-x 105 :end-y 260
                  :head-direction :both )
   (make-instance 'capi:arrow-pinboard-object
                  :start-x 5 :start-y 310
                  :end-x 105 :end-y 360
                  :head-graphics-args
                  '(:foreground :pink)
                  :head-length 30)
   (make-instance 'capi:arrow-pinboard-object
                  :start-x 5 :start-y 410
                  :end-x 105 :end-y 460
                  :head-length 30 :head-breadth 5)
   (make-instance 'capi:arrow-pinboard-object
                  :start-x 5 :start-y 510
                  :end-x 105 :end-y 560
                  :head-breadth 10
                  :head-direction :backwards))
  :visible-min-width 120
  :visible-min-height 620))
```

attach-interface-for-callback

Function

Summary	The <code>attach-interface-for-callback</code> function changes the interface that is passed when a callback is made.
Package	<code>capi</code>
Signature	<code>attach-interface-for-callback</code> <i>element interface</i>

Description	Callbacks for <i>element</i> get passed <i>interface</i> instead of <i>element's</i> parent interface.
See also	<code>callbacks</code> <code>element</code> <code>element-interface-for-callback</code> <code>interface</code>

attach-simple-sink

Function

Summary	Attaches a sink to the active component in an <code>ole-control-pane</code> .
Package	<code>capi</code>
Signature	<code>attach-simple-sink invoke-callback pane interface-name &key sink-class => sink</code>
Arguments	<p><i>invoke-callback</i> A function designator.</p> <p><i>pane</i> An <code>ole-control-pane</code>.</p> <p><i>interface-name</i> A refguid or the symbol <code>:default</code>.</p> <p><i>sink-class</i> A symbol naming a class.</p>
Values	<i>sink</i> The sink object.
Description	<p>The function <code>attach-simple-sink</code> make a sink object and attaches it to the active component in <i>pane</i>.</p> <p>When an event callback is triggered for the source interface named by <i>interface-name</i>, the sink object will call the <i>invoke-callback</i> with four arguments: the <i>pane</i> (see <i>sink-class</i> below), the source method name as a string, the source method type (either <code>:method</code>, <code>:get</code> or <code>:put</code>) and a vector of the remaining callback arguments.</p>

interface-name is either a string naming a source interface that the component in *pane* supports or `:default` to connect to the default source interface.

sink-class can be used to control the class of the sink object. This defaults to `ole-control-pane-simple-sink`, but can be a subclass of this class to allow the first argument of the *invoke-callback* to be chosen by a method on the generic function `com:simple-i-dispatch-callback-object`.

Attached sinks are automatically disconnected when the object is closed or can be manually disconnected by calling `detach-simple-sink`.

Note: this function is implemented only in LispWorks for Windows. Load the functionality by `(require "embed")`.

See also `detach-simple-sink`
`ole-control-pane`
`ole-control-pane-simple-sink`

attach-sink

Function

Summary	Attaches a sink to the active component in an <code>ole-control-pane</code> .
Package	<code>capi</code>
Signature	<code>attach-sink sink pane interface-name</code>
Arguments	<i>sink</i> A class instance. <i>pane</i> An <code>ole-control-pane</code> . <i>interface-name</i> A refguid or the symbol <code>:default</code> .
Description	The function <code>attach-sink</code> attaches a sink to the active component in the the <code>ole-control-pane</code> <i>pane</i> .

sink is an instance of a class that implements the source interface *interface-name*.

pane is an `ole-control-pane` which is the pane where the component is.

interface-name is either a string naming a source interface that the component in *pane* supports or `:default` to connect to the default source interface.

Attached sinks are automatically disconnected when the object is closed or can be manually disconnected by calling `detach-sink`.

Note: this function is implemented only in LispWorks for Windows. Load the functionality by `(require "embed")`.

See also `detach-sink`
`ole-control-pane`

beep-pane

Function

Summary	The <code>beep-pane</code> function sounds a beep on a screen.
Package	<code>capi</code>
Signature	<code>beep-pane &optional pane</code>
Description	This sounds a beep on the screen associated with <i>pane</i> or on the current screen if <i>pane</i> is <code>nil</code> .
Example	<code>(capi:beep-pane)</code>
See also	<code>simple-pane</code> <code>screen</code>

button

Class

Summary	A <code>button</code> is a pane that displays either a piece of text or an image, and that performs an action when pressed. Certain types of buttons can also be selected and deselected.	
Package	<code>capi</code>	
Superclasses	<code>simple-pane</code> <code>item</code>	
Subclasses	<code>push-button</code> <code>radio-button</code> <code>check-button</code>	
Initargs	<code>:interaction</code>	The interaction style for the button.
	<code>:selected</code>	For radio button and check button styles, if <code>selected</code> is set to <code>t</code> , the button is initially selected.
	<code>:callback</code>	Specifies the callback to use when the button is selected.
	<code>:image</code>	An image for the button (or <code>nil</code>).
	<code>:selected-image</code>	The image used when the button is selected.
	<code>:enabled</code>	If <code>nil</code> the button cannot be selected.
	<code>:cancel-p</code>	If true the button is the "Cancel" button, that is, the button selected by the <code>Escape</code> key.
	<code>:default-p</code>	If true the button is the default button, that is, the button selected by the <code>Return</code> key.
	The following initargs controlling alternate images do not apply on Microsoft Windows:	
	<code>:disabled-image</code>	The image for the button when disabled (or <code>nil</code>).

`:selected-disabled-image`

The image used when the button is selected and disabled.

`:armed-image` The image used when the button is pressed and *interaction* is `:no-selection`.

The following initargs controlling mnemonics apply only on Microsoft Windows:

`:mnemonic` A character, integer or symbol specifying a mnemonic for the button.

`:mnemonic-text`

A string specifying the text and a mnemonic.

`:mnemonic-escape`

A character specifying the mnemonic escape. The default value is `#\&`.

Accessors

`button-selected`
`button-image`
`button-armed-image`
`button-selected-image`
`button-disabled-image`
`button-selected-disabled-image`
`button-enabled`
`button-cancel-p`
`button-default-p`

Description

The class `button` is the class that `push-button`, `radio-button`, and `check-button` are built on. It can be displayed either with text or an image, and a callback is called when the button is clicked. It inherits all of its textual behavior from `item`, including the slot `text` which is the text that appears in the button.

Rather than creating direct instances of `button`, you usually create instances of its subclasses, each of which has a specific interaction style. Occasionally it may be easier to instantiate `button` directly with the appropriate value of *interaction* (for

instance, when the interaction style is only known at run-time) but you may not use such a button as an item in a `button-panel`.

The values allowed for *interaction* are as follows:

`:no-selection` A push button.

`:single-selection`
A radio button.

`:multiple-selection`
A check button.

Both radio buttons and check buttons can have a selection which can be set using the initarg `:selected` and the accessor `button-selected`.

The button's callback gets called when the user clicks on the button, and by default gets passed the data in the button and the interface. This can be changed by specifying a callback type as described in the description of callbacks. The following callbacks are accepted by buttons:

`:callback` Called when the button is pressed.

`:selection-callback`
Called when the button is selected.

`:retract-callback`
Called when the button is deselected.

By default, *image* and *disabled-image* are `nil`, meaning that the button is a text button, but if *image* is provided then the button displays an image instead of the text. The image can be an `external-image` or any object accepted by `load-image`. The disabled image is the image that is shown when the button is disabled (or `nil`, meaning that it is left for the window system to decide how to display the image as disabled). On Microsoft Windows, the system computes the disabled image and so *disabled-image* is ignored.

The button's actions can be enabled and disabled with the *enabled* slot, and its associated accessor `button-enabled`. This means that when the button is disabled, pressing on it does not call any callbacks or change its selection.

Note that the class `button-pane1` provides functionality to group buttons together, and should normally be used in preference to creating individual buttons yourself. For instance, a `radio-button-pane1` makes a number of radio buttons and also controls them such that only one button is ever selected at a time.

A mnemonic is an underlined character within the button *text* or the printed representation of the button *data* which can be entered to select the button. The value *mnemonic* is interpreted as described for `menu`.

An alternative way to specify a mnemonic is to pass *mnemonic-text*. This is a string which provides the text for the button and also specifies the mnemonic character. *mnemonic-text* and *mnemonic-escape* are interpreted in just the same way as the *mnemonic-title* and *mnemonic-escape* of `menu`.

Example

In the following example a button is created. Using the `button-enabled` accessor the button is then enabled and disabled.

```
(setq button
  (capi:contain (make-instance
                'capi:push-button
                :text "Press Me")))

(capi:apply-in-pane-process
 button #'(setf capi:button-enabled) nil button)

(capi:apply-in-pane-process
 button #'(setf capi:button-enabled) t button)
```

In the next example a button with an image instead of text is created.

```
(setq button
  (capi:contain
    (make-instance
      'capi:push-button
      :image
      (merge-pathnames
        "capi/applications/images/info.bmp"
        (sys:lispworks-dir "examples")))))
```

The following examples illustrate mnemonics:

```
(defun egg (&rest ignore)
  (declare (ignore ignore))
  (capi:display-message "Egg"))
```

```
(capi:contain
  (make-instance 'capi:push-button
    :selection-callback 'egg
    :mnemonic-text "Chicken && Rice"))
```

```
(capi:contain
  (make-instance 'capi:push-button
    :data "Chicken"
    :selection-callback 'egg
    :mnemonic #\k))
```

Compare this with the previous example: the `#\k` does not appear and the `#\e` becomes the mnemonic:

```
(capi:contain
  (make-instance 'capi:push-button
    :selection-callback 'egg
    :mnemonic-escape #\k
    :mnemonic-text "Chicken"))
```

Also see the example in the directory `examples/capi/buttons/`.

See also

`button-panel`
`callbacks`

button-panel*Class*

Summary	The class <code>button-panel</code> is a pane containing a number of buttons that are laid out in a particular style, and that have group behavior.
Package	<code>capi</code>
Superclasses	<code>choice</code> <code>titled-object</code> <code>simple-pane</code>
Subclasses	<code>push-button-panel</code> <code>radio-button-panel</code> <code>check-button-panel</code>
Initargs	<p><code>:layout-class</code> The type of layout for the buttons.</p> <p><code>:layout-args</code> Initialization arguments for the layout.</p> <p><code>:callbacks</code> The selection callbacks for each button.</p> <p><code>:button-class</code> The class of the buttons.</p> <p><code>:images</code> A list.</p> <p><code>:disabled-images</code> A list.</p> <p><code>:armed-images</code> A list.</p> <p><code>:selected-images</code> A list.</p> <p><code>:selected-disabled-images</code> A list.</p> <p><code>:help-keys</code> A list.</p> <p><code>:default-button</code> Specifies the default button.</p> <p><code>:cancel-button</code> Specifies the cancel button.</p>

The following initargs controlling mnemonics apply only on Microsoft Windows:

:mnemonics A list specifying mnemonics for the buttons.

:mnemonic-items
A list of strings, each specifying the text and a mnemonic.

:mnemonic-escape
A character specifying the mnemonic escape. The default value is `#\&`.

:mnemonic-title
A string specifying the title and a mnemonic.

Accessors `pane-layout`

Description The class `button-panel` inherits most of its behavior from `choice`, which is an abstract class providing support for handling items and selections. By default, a button panel has single selection interaction style (meaning that only one of the buttons can be selected at any one time), but this can be changed by specifying an *interaction*.

The subclasses `push-button-panel`, `radio-button-panel` and `check-button-panel` are provided as convenience classes, but they are just button panels with different interactions (`:no-selection`, `:single-selection` and `:multiple-selection` respectively).

The layout of the buttons is controlled by a layout of class *layout-class* (which defaults to `row-layout`) but this can be changed to be any other CAPI layout. When the layout is created, the list of initargs *layout-args* is passed to `make-instance`.

Each button uses the callbacks specified for the button panel itself, unless the argument *callbacks* is specified. *callbacks* should be a list (one element per button). Each element of *callbacks*, if non-`nil`, will be used as the selection callback of the corresponding button.

button-class, if supplied, determines the class used for each of the buttons. This should be the class appropriate for the *interaction*, or a subclass of it. The default behavior is to create buttons of the class appropriate for the *interaction*.

Each of *images*, *disabled-images*, *armed-images*, *selected-images*, *selected-disabled-images* and *help-keys*, if supplied, should be a list of the same length as *items*. The values are passed to the corresponding item, and interpreted as described for `button`. The `button-panel` *images* values map to `button image` arguments, and so on.

For `button-panel` and its subclasses, the *items* supplied to the `:items` initarg and `(setf collection-items)` function can contain button objects. In this case, the button is used directly in the button panel rather than a button being created by the CAPI.

This allows button size and spacing to be controlled explicitly. Note that the button must be of the appropriate type for the subclass of `button-panel` being used, as shown in the following table:

Button panel class	Button class
<code>push-button-panel</code>	<code>push-button</code>
<code>radio-button-panel</code>	<code>radio-button</code>
<code>check-button-panel</code>	<code>check-button</code>

Table 1.1 Button and panel classes

For example,

```
(let ((button1 (make-instance 'capi:push-button
                             :text "button1"
                             :internal-border 20
                             :visible-min-width 200))
      (button2 (make-instance 'capi:push-button
                             :text "button2"
                             :internal-border 20
                             :visible-min-width 200)))
      (capi:contain (make-instance 'capi:push-button-panel
                                   :items (list button1 button2)
                                   :layout-args '(:x-gap 30))))
```

default-button specifies which button is the default (selected by pressing **Return**). It should be equal to a member of *items* when compared by *test-function*. If the items are non-immediate objects such as strings or `button` objects, you must ensure either that the same (`eq`) object is passed in *items* as in *default-button*, or that a suitable *test-function* is supplied.

cancel-button specifies which button is selected by pressing **Escape**. The comparison with members of *items* is as for *default-button*.

mnemonics is a list of the same length as *items*. Each element is a character, integer or symbol specifying the mnemonic for the corresponding button in the same way as described for `menu`.

mnemonic-items is an alternate way to specify the mnemonics in a button panel. It is a list of the same length as *items*. Each element is a string which is interpreted for the corresponding `button` as its *mnemonic-text* `initarg`.

mnemonic-title and *mnemonic-escape* are interpreted as for `menu`. *mnemonic-escape* specifies the escape character for mnemonics both in the buttons and in the pane's title.

Compatibility Note

Button panels now default to having a maximum size constrained to their minimum size as this is useful when attempting to layout button panels into arbitrary spaces without them changing size. To get the old behavior, specify `:visible-max-width nil` in the `make-instance`.

Example

```
(capi:contain (make-instance
              'capi:button-panel
              :items '(:red :green :blue)
              :print-function 'string-capitalize))

(setq buttons
  (capi:contain
   (make-instance
    'capi:button-panel
    :items '(:red :green :blue)
    :print-function 'string-capitalize
    :interaction :multiple-selection)))

(capi:apply-in-pane-process
 buttons #'(setf capi:choice-selected-items)
 '(:red :green) buttons)

(capi:contain (make-instance
              'capi:button-panel
              :items '(1 2 3 4 5 6 7 8 9)
              :layout-class 'capi:grid-layout
              :layout-args '(:columns 3)))
```

This example illustrates use of *default-button* and *test-function*:

```
(capi:contain
 (make-instance 'capi:push-button-panel
               :items '("one" "two" "three")
               :default-button "two"
               :test-function 'equalp
               :selection-callback
               'capi:display-message))
```

Also see the example in the directory
`examples/capi/buttons/`.

See also

```
radio-button
check-button
push-button
set-button-panel-enabled-items
```

calculate-constraints

Generic Function

Summary	The <code>calculate-constraints</code> generic function calculates the minimum and maximum size of a pane.
Package	<code>capi</code>
Signature	<code>calculate-constraints</code> <i>pane</i>
Arguments	<i>pane</i> A CAPI pane or layout.
Description	<p>The generic function <code>calculate-constraints</code> calculates the minimum and maximum size for <i>pane</i> according to the sizes of its children, and sets these values into <i>pane</i>'s geometry cache.</p> <p>The CAPI calls <code>calculate-constraints</code> for each pane and layout that it displays.</p> <p>When creating your own layout, you should define a method for <code>calculate-constraints</code> that sets the values of the following geometry slots based on the constraints of its children.</p> <p><code>%min-width%</code> The minimum width of pane. <code>%max-width%</code> The maximum width of pane. <code>%min-height%</code> The minimum height of pane. <code>%max-height%</code> The maximum height of pane.</p> <p>(See <code>with-geometry</code>.)</p> <p>The constraints of any CAPI element can be found by calling <code>get-constraints</code>.</p>
See also	<code>calculate-layout</code> <code>define-layout</code> <code>get-constraints</code>

`element`
`layout`
`with-geometry`

calculate-layout

Generic Function

Summary The `calculate-layout` generic function is used to provide a method for laying out the children of a new layout.

Package `capi`

Signature `calculate-layout layout x y width height`

Description The generic function `calculate-layout` is called by the CAPI to layout the children of a layout. When defining a new class of layout using `define-layout`, a `calculate-layout` method must be provided that sets the `x`, `y`, `width` and `height` of each of the layout's children. This method must try to obey the constraints specified by its children (its minimum and maximum size) and should only break them when it becomes impossible to fit the constraints of all of the children.

To set the `x`, `y`, `width` and `height` of the layout, use the macro `with-geometry` which works in a similar way as `with-slots`.

See also `get-constraints`
`with-geometry`
`interpret-description`

callbacks

Class

Summary The class `callbacks` is used as a mixin by classes that provide callbacks.

Package `capi`

Superclasses	<code>capi-object</code>
Subclasses	<code>collection</code> <code>item</code> <code>menu-object</code>
Initargs	<p><code>:callback-type</code> The type of arguments for the callbacks.</p> <p><code>:selection-callback</code> The callback for selecting an item.</p> <p><code>:extend-callback</code> The callback for extending the selection.</p> <p><code>:retract-callback</code> The callback for deselecting an item.</p> <p><code>:action-callback</code> The callback for an action.</p>
Accessors	<code>callbacks-callback-type</code> <code>callbacks-selection-callback</code> <code>callbacks-extend-callback</code> <code>callbacks-retract-callback</code> <code>callbacks-action-callback</code>
Description	<p>Each callback function can be one of the following:</p> <p><i>function</i> Call the function.</p> <p><i>list</i> Apply the head of the list to the tail.</p> <p><code>:redisplay-interface</code> Call <code>redisplay-interface</code> on the top-level interface.</p> <p><code>:redisplay-menu-bar</code> Call <code>redisplay-menu-bar</code> on the top-level interface.</p>

The slot value *callback-type* determines which arguments get passed to each of the callbacks. It can be any of the following values, and passes the corresponding data to the callback function:

<code>:collection-data</code>	<i>(collection data)</i>
<code>:data</code>	<i>(item-data)</i>
<code>:data-interface</code>	<i>(item-data interface)</i>
<code>:interface-data</code>	<i>(interface item-data)</i>
<code>:item</code>	<i>(item)</i>
<code>:item-interface</code>	<i>(item interface)</i>
<code>:interface-item</code>	<i>(interface item)</i>
<code>:interface</code>	<i>(interface)</i>
<code>:full</code>	<i>(item-data item interface)</i>
<code>:focus</code>	The pane with the current input focus.
<code>:none</code>	<i>()</i>
<code>nil</code>	<i>()</i>

callback-type can also be a list containing any of `:focus`, `:data`, `:interface`, `:collection`, `:item`.

The *item-data* variable is the item's data if the item is of type `item`, otherwise it is the item itself, as for *item*. The *item* variable means the item itself. The *interface* is the `element-interface` of the element. *collection* is the element's `collection`, if there is one.

See also `abort-callback`
`choice`
`attach-interface-for-callback`

call-editor

Generic Function

Summary The `call-editor` generic function executes an editor command in an `editor-pane`.

Package `capi`

Signature `call-editor editor-pane command`

Description This executes the editor command *command* in the current buffer in *editor-pane*.

It can be used directly in a callback in *editor-pane*'s interface. See the demo interface example in the *LispWorks CAPI User Guide*. In other cases, take care to modify displayed CAPI interfaces only in their own process: `execute-with-interface` and `apply-in-pane-process` are useful for this.

The *before-input-callback* and *after-input-callback* of the `editor-pane` are called when `call-editor` is called.

Example

```
(setq editor (capi:contain
              (make-instance 'capi:editor-pane
                            :text "abc")))

(capi:apply-in-pane-process
 editor 'capi:call-editor editor "End Of Buffer")
```

Also see the example in the directory `examples/capi/editor/`.

See also `apply-in-pane-process`
`editor-pane`
`execute-with-interface`

capi-object*Class*

Summary	The class <code>capi-object</code> is the superclass of all CAPI classes.	
Package	<code>capi</code>	
Superclasses	<code>standard-class</code>	
Subclasses	<code>item</code> <code>callbacks</code> <code>element</code> <code>interface</code> <code>pinboard-object</code>	
Initargs	<code>:name</code>	The name of the object.
	<code>:plist</code>	A property list for storing miscellaneous information.
Accessors	<code>capi-object-name</code> <code>capi-object-plist</code>	
Description	<p>The class <code>capi-object</code> provides a name and a property list for general purposes, along with the accessors <code>capi-object-name</code> and <code>capi-object-plist</code> respectively. A <code>capi-object</code>'s name is defaulted by <code>define-interface</code> to be the name of the slot into which the object is put.</p>	
Examples	<pre>(setq object (make-instance 'capi:capi-object :name 'test)) (capi:capi-object-name object) (setf (capi:capi-object-plist object) '(:red 1 :green 2 :blue 3)) (capi:capi-object-property object :green)</pre>	
See also	<code>capi-object-property</code>	

capi-object-property

Function

Summary	The <code>capi-object-property</code> function is used to get and set properties in the property list of a <code>capi-object</code> .
Package	<code>capi</code>
Signature	<code>capi-object-property</code> <i>object property</i>
Signature	<code>(setf capi-object-property) value object property</code>
Description	All CAPI objects contain a property list, similar to the symbol <code>plist</code> . The recommended ways of setting properties are <code>capi-object-property</code> and <code>(setf capi-object-property)</code> . To remove a property, use the function <code>remove-capi-object-property</code> .
Example	<p>In this example a list panel is created, and a test property is set and examined using <code>capi-object-property</code>.</p> <pre>(setq pane (make-instance 'capi:list-panel :items '(1 2 3))) (capi:capi-object-property pane 'test-property) (setf (capi:capi-object-property pane 'test-property) "Test") (capi:capi-object-property pane 'test-property) (capi:remove-capi-object-property pane 'test-property) (capi:capi-object-property pane 'test-property)</pre>
See also	<code>capi-object</code> <code>remove-capi-object-property</code>

check-button*Class*

Summary A check button is a button that can be either selected or deselected, and its selection is independent of the selections of any other buttons.

Package `capi`

Superclasses `button`
`titled-object`

Description The class `check-button` inherits most of its behavior from the class `button`. Note that it is normally best to use a `check-button-panel` rather than make the individual buttons yourself, as the button panel provides functionality for handling groups of buttons. However, `check-button` can be used if you need to have more control over the button's behavior.

Example The following code creates a check button.

```
(setq button (capi:contain
              (make-instance 'capi:check-button
                            :text "Press Me")))
```

The button can be selected and deselected using this code.

```
(capi:apply-in-pane-process
 button #'(setf capi:button-selected) t button)

(capi:apply-in-pane-process
 button #'(setf capi:button-selected) nil button)
```

The following code disables and enables the button.

```
(capi:apply-in-pane-process
 button #'(setf capi:button-enabled) nil button)

(capi:apply-in-pane-process
 button #'(setf capi:button-enabled) t button)
```

See also `push-button`
`radio-button`
`button-panel`

check-button-panel

Class

Summary	A <code>check-button-panel</code> is a pane containing a group of buttons each of which can be selected or deselected.
Package	<code>capi</code>
Superclasses	<code>button-panel</code>
Description	The class <code>check-button-panel</code> inherits all of its behavior from <code>button-panel</code> , which itself inherits most of its behavior from <code>choice</code> . Thus, the <code>check-button-panel</code> can accept <i>items</i> , <i>callbacks</i> , and so on.
Example	<pre>(capi:contain (make-instance 'capi:check-button-panel :title "Select some packages" :items '("CAPI" "LISPWORKS" "CL-USER"))) (setq buttons (capi:contain (make-instance 'capi:check-button-panel :title "Select some packages" :items '("CAPI" "LISPWORKS" "CL-USER") :layout-class 'capi:column-layout))) (capi:choice-selected-items buttons)</pre> <p>Also see the example in the directory <code>examples/capi/buttons/</code>.</p>
See also	<code>check-button</code> <code>push-button-panel</code> <code>radio-button-panel</code>

choice

Class

Summary	A <code>choice</code> is an abstract class that collects together a group of items, and provides functionality for displaying and selecting them.
---------	---

Package	<code>capi</code>
Superclasses:	<code>collection</code>
Subclasses	<code>list-panel</code> <code>button-panel</code> <code>option-pane</code> <code>graph-pane</code> <code>menu-component</code> <code>tree-view</code>
Initargs	<p><code>:interaction</code> The interaction style of the choice.</p> <p><code>:selection</code> The indexes of the choice's selected items.</p> <p><code>:selected-item</code> The selected item for a single selection choice.</p> <p><code>:selected-items</code> A list of the selected items.</p> <p><code>:keep-selection-p</code> If <code>t</code>, retains any selection when the items change.</p> <p><code>:initial-focus-item</code> If supplied, this should be an item in the choice.</p>
Accessors	<code>choice-selection</code>
Readers	<code>choice-interaction</code> <code>choice-initial-focus-item</code>
Description	The class <code>choice</code> inherits most of its behavior from <code>collection</code> , and then provides the selection facilities itself. The classes <code>list-panel</code> , <code>button-panel</code> , <code>option-pane</code> , <code>menu-component</code> and <code>graph-pane</code> inherit from it, and so it plays a key role in CAPI applications.

A `choice` can have one of four different interaction styles, and these control how it behaves when an item is selected by the user. *interaction* can be one of:

`:no-selection` The choice behaves just as a collection.

`:single-selection`

The choice can have only one selected item.

`:multiple-selection`

The choice can have multiple selected items.

`:extended-selection`

An alternative to `multiple-selection`.

With interaction `:no-selection`, the choice cannot have a selection, and so behaves just as a collection would.

With interaction `:single-selection`, the choice can only have one item selected at a time. When a new selection is made, the old selection is cleared and its *selection-callback* is called. The *selection-callback* is also called when the user invokes the selection gesture on the selected item.

With interaction `:multiple-selection`, the choice can have any number of items selected, and selecting an item toggles its selection status. The *selection-callback* is called when an item becomes selected, and the *retract-callback* is called when an item is deselected. `:multiple-selection` is not supported on Mac OS X.

With interaction `:extended-selection`, the choice can have any number of items selected as with `:multiple-selection` interaction, but the usual selection gesture removes the old selection. However, there is a window system-specific means of extending the selection. When an item is selected the *selection-callback* is called, when the selection is extended the *extend-callback* is called, and when an item is deselected the *retract-callback* is called.

On Mac OS X, the selection gesture is mouse (left button) click. De-selection and discontinuous selections are made by `Command+Click`, and a continuous selection is made by `Shift+Click`, regardless of whether if *interaction* is `:multiple-selection` or `:extended-selection`.

The choice's selection stores the indices of the currently selected item, and is a single number for single selection choices and a list for all other interactions. The functions `choice-selected-item` and `choice-selected-items` treat the selection in terms of the items themselves as opposed to their indices.

Usually when a choice's items are changed using `(setf collection-items)` the selection is lost.

However, if the choice was created with `:keep-selection-p t`, then the selection is preserved over the change.

initial-focus-item, if supplied, specifies the item which has the input focus when the choice is first displayed.

Compatibility Note In LispWorks 5.0 and earlier versions, for interaction `:single-selection` the *selection-callback* is called only after a new selection is made.

Example The following example defines a choice with three possible selections.

```
(setq choice (make-instance 'capi:choice
                           :items '("One" "Two" "Three")
                           :selection 0))

(capi:display-message "Selection: ~S"
                     (capi:choice-selection choice))

(capi:choice-selected-item choice)
```

The selection is changed using the following code.

```
(setf (capi:choice-selection choice) 1)

(capi:choice-selected-item choice)
```

Also see the examples in the directory
`examples/capi/choice/` and in
`examples/capi/graphics/graph-pane.lisp`

See also `choice-selected-item`
`choice-selected-item-p`
`choice-selected-items`
`choice-update-item`

choice-selected-item

Generic Function

Summary The function `choice-selected-item` returns the currently selected item in a single selection choice.

Package `capi`

Signature `choice-selected-item choice`

Signature `(setf choice-selected-item) item choice`

Description The function `choice-selected-item` returns the currently selected item in a single selection choice. A `setf` method is provided as a means of setting the selection. Note that the items are compared by *choice's test-function* - see `collection` or the example below.

It is an error to call this function on choices with different interactions — in that case, you should use `choice-selected-items`.

Example This example illustrates setting the selection. First we set up a single selection choice — in this case, a `list-panel`.

```
(setq list (capi:contain
            (make-instance 'capi:list-panel
                          :items '(a b c d e)
                          :selection 2)))
```

The following code line returns the selection of the list panel.

```
(capi:choice-selected-item list)
```

The selection can be changed, and the change viewed, using the following code.

```
(capi:apply-in-pane-process
 list #'(setf capi:choice-selected-item) 'e list)

(capi:choice-selected-item list)
```

This example illustrates the effect of the *test-function*. Make a choice with *test-function* eq:

```
(setf *list*
      (capi:contain
       (make-instance 'capi:list-panel
                     :items (list "a" "b" "c")
                     :selection 0
                     :visible-min-height :text-height)))
```

This call loses the selection since (eq "b" "b") fails:

```
(capi:apply-in-pane-process
 *list* #'(setf capi:choice-selected-item)
 "b" *list*)
```

Change the test function:

```
(capi:apply-in-pane-process
 *list* #'(setf capi:collection-test-function)
 'equal *list*)
```

This call sets the selection since (equal "b" "b") succeeds:

```
(capi:apply-in-pane-process
 *list* #'(setf capi:choice-selected-item)
 "b" *list*)
```

See also

```
choice
choice-selected-items
collection
```

choice-selected-item-p

Function

Summary	Checks if an item is currently selected in a choice.
Package	<code>capi</code>
Signature	<code>choice-selected-item-p</code> <i>choice</i> <i>item</i>
Description	<p>The function <code>choice-selected-item-p</code> is the predicate for whether an item <i>item</i> of the choice <i>choice</i> is selected.</p> <p>Note that the items are compared by <i>choice's test-function</i> - see <code>collection</code> for details.</p>
Example	<pre>(setq list (capi:contain (make-instance 'capi:list-panel :items '(a b c d) :selection 2 :visible-min-height '(:character 4)))) (capi:choice-selected-item-p list 'c) => t Now click on another item. (capi:choice-selected-item-p list 'c) => nil</pre>
See also	<code>choice</code> <code>collection</code>

choice-selected-items

Generic Function

Summary	The function <code>choice-selected-items</code> returns the currently selected items in a choice as a list of the items.
Package	<code>capi</code>

Signature	<code>choice-selected-items</code> <i>choice</i>
Signature	<code>(setf choice-selected-items)</code> <i>items</i> <i>choice</i>
Description	<p>The function <code>choice-selected-items</code> returns the currently selected items in a choice as a list of the items. A <code>setf</code> method is provided as a means of setting the currently selected items. Note that the items are compared by <i>choice's test-function</i> - see <code>collection</code> for details.</p> <p>In the case of <code>:single-selection</code> choices, it is usually easier to use the complementary function <code>choice-selected-item</code>, which returns the selected item as its result.</p>
Examples	<p>First we set up a <code>:multiple-selection</code> choice — in this case, a list panel.</p> <pre>(setq list (capi:contain (make-instance 'capi:list-panel :items '(a b c d e) :visible-min-height '(:character 5) :interaction :multiple-selection :selection '(1 3))))</pre> <p>The following code line returns the selections of the list.</p> <pre>(capi:choice-selected-items list)</pre> <p>The selections of the list panel can be changed and redisplayed using the following code.</p> <pre>(capi:apply-in-pane-process list #'(setf capi:choice-selected-items) '(a c e) list)</pre> <pre>(capi:choice-selected-items list)</pre>
See also	<p><code>choice</code> <code>choice-selected-item</code> <code>collection</code></p>

choice-update-item

Function

Summary	Updates an item in a choice.
Package	<code>capi</code>
Signature	<code>choice-update-item</code> <i>choice</i> <i>item</i>
Description	The function <code>choice-update-item</code> updates the display of the item <i>item</i> in the choice <i>choice</i> . It should be called if the display of <i>item</i> (that is, the string returned by the <i>print-function</i>) changes.

Example Create a list panel that displays the status of something

```
(defun my-print-an-item (item)
  (format nil "~a: ~a"
          (substitute-if-not #\space
                             'alphanumericp
                             (symbol-name item))
          (symbol-value item)))

(defvar *status-one* :on)
(defvar *status-two* :off)

(setq list
      (capi:contain
       (make-instance
        'capi:list-panel
        :items '(*status-one* *status-two*)
        :print-function 'my-print-an-item
        :visible-min-height :text-height
        :visible-min-width :text-width)))
```

Setting the status variables does not change the display:

```
(setq *status-one* :error)
```

Update the `item` to change the display:

```
(capi:choice-update-item list '*status-one*)
```

See also `choice`

clipboard*Function*

Summary	Returns the contents of the system clipboard.	
Package	<code>capi</code>	
Signature	<code>clipboard self &optional format => result</code>	
Arguments	<i>self</i>	A displayed CAPI pane or interface.
	<i>format</i>	A keyword.
Values	<i>result</i>	A string, an <code>image</code> , a Lisp object, or <code>nil</code> .
Description	<p>The function <code>clipboard</code> returns the contents of the system clipboard as a string, or <code>nil</code> if the clipboard is empty.</p> <p><i>format</i> controls what kind of object is read. The following values of <i>format</i> are recognized:</p> <ul style="list-style-type: none"> <code>:string</code> The object is a string. This the default value. <code>:image</code> The object is of type <code>image</code>, converted from whatever format the platform supports. <code>:value</code> The object is the Lisp value. <code>:metafile</code> The object is a metafile. <p>When <i>format</i> is <code>:image</code>, the image returned by <code>clipboard</code> is associated with <i>self</i>, so you can free it explicitly with <code>free-image</code> or it will be freed automatically when the pane is destroyed.</p> <p>When <i>format</i> is <code>:metafile</code> the object is a metafile which should be freed using <code>free-metafile</code> when no longer needed. See also <code>draw-metafile</code> and <code>draw-metafile-to-image</code>. <i>format</i> <code>:metafile</code> is not supported on X11/Motif.</p> <p>The Microsoft Windows clipboard is usually set by the user with the <code>ctrl+c</code> and <code>ctrl+x</code> gestures. Note that LispWorks uses these gestures when in Windows emulation mode.</p>	

On X11/Motif, various gestures may set the clipboard. Note that LispWorks uses `ctrl+c` and `ctrl+x` when in KDE/Gnome editor emulation mode. The X clipboard can also be accessed by running the program `xclipboard` or the Emacs function `x-get-clipboard`.

The Mac OS X clipboard is usually set by the user with the `Command+C` and `Command+x` gestures.

See also

- `clipboard-empty`
- `draw-metafile`
- `draw-metafile-to-image`
- `free-image`
- `free-metafile`
- `image`
- `selection`
- `set-clipboard`
- `text-input-pane-paste`

clipboard-empty

Function

Summary	Determines whether the system clipboard contains an object of the specified kind.	
Package	<code>capi</code>	
Signature	<code>clipboard-empty self &optional format => result</code>	
Arguments	<i>self</i>	A displayed CAPI pane or interface.
	<i>format</i>	A keyword.
Values	<i>result</i>	<code>t</code> or <code>nil</code> .
Description	The function <code>clipboard-empty</code> returns <code>nil</code> if there is an object of the kind indicated by <i>format</i> on the clipboard, or <code>t</code> otherwise.	

format controls what kind of object is checked. The allowed values of *format* are as described for `clipboard`.

See also `clipboard`
`image`

clone

Generic Function

Summary `Creates a copy of a CAPI object.`

Package `capi`

Signature `clone capi-object => cloned-object`

Arguments `capi-object` An instance of a subclass of `capi-object`

Values `cloned-object` A copy of `capi-object`.

Description The generic function `clone` returns a new object `cloned-object` which is a copy of `capi-object`. It does not share any data with `capi-object`, but has a copy of the useful part of its state.

The system contains methods on `clone`. You may add methods on your own interface classes.

See also `capi-object`

cocoa-default-application-interface

Class

Summary A class from which the Macintosh application interface should inherit.

Package `capi`

Superclasses `interface`

Initargs	<p><code>:message-callback</code> A function or <code>nil</code>.</p> <p><code>:application-menu</code> <code>nil</code>, a menu, or the name of a slot containing a menu in the application interface.</p> <p><code>:dock-menu</code> <code>nil</code>, a menu, or a function designator.</p>
Accessors	<p><code>application-interface-message-callback</code> <code>application-interface-application-menu</code> <code>application-interface-dock-menu</code></p>
Description	<p>The class <code>cocoa-default-application-interface</code> supports application messages and the application menu for a Cocoa application.</p> <p>When non-<code>nil</code>, <i>message-callback</i> should be a function with signature</p> <p><i>interface message &rest args</i></p> <p><i>message-callback</i> will be called for various application messages. The <i>interface</i> argument will be the application interface and the <i>message</i> argument will be a keyword. The only currently defined message is <code>:open-file</code>. In this case <i>args</i> will contain the name of the file to open. This message is invoked when the user double-clicks on a document associated with the application or drags a document into the application icon.</p> <p><i>application-menu</i> controls the application's main menu. If this is <code>nil</code>, then a minimal application menu will be made using the title of the application interface, otherwise it should be a menu containing the usual items or the name of a slot containing such a menu in the application interface.</p> <p><i>dock-menu</i> provides a menu for use by the Mac OS X Dock icon. If the value is <code>nil</code> (the default), then the standard menu is used. If <i>dock-menu</i> is a function designator, it is called with the application interface as its argument when the menu is</p>

popped up and should return a menu. Otherwise *dock-menu* should be a menu, which is used directly. The Dock will add the standard items such as **Quit** to the end of the menu you supply.

`interface` `initargs` are interpreted as follows:

- The *activate-callback* is called when the application is activated or deactivated.
- The *destroy-callback* is called when the application shuts down.
- The *confirm-destroy-function* is called to confirm whether the application should shut down.

The application interface also allows you to control aspects of the application. In particular:

- The function `destroy` will cause the application to shut down.
- The function `top-level-interface-display-state` will return `:hidden` if the whole application is hidden and will return `:normal` otherwise.
- The function `(setf top-level-interface-display-state)` can be used to perform some operations typically found on the application menu.

The *display-state* value can one of:

<code>:normal</code>	Show the application and activate it
<code>:restore</code>	Show the application again without activating it
<code>:hidden</code>	Hide
<code>:others-hidden</code>	Hide Others
<code>:all-normal</code>	Show All

To make your application use your `cocoa-default-application-interface`, do not display it explicitly, but call `set-application-interface`.

Note: `cocoa-default-application-interface` is implemented only in LispWorks for Macintosh with the Cocoa IDE.

Examples: See the examples in
`examples/capi/applications/cocoa-application.lisp`
`examples/delivery/macos/simple-application.lisp`
`examples/delivery/macos/full-application.lisp`

See also `set-application-interface`

cocoa-view-pane

Class

Summary A `cocoa-view-pane` allows an arbitrary Cocoa view class to be used on the Macintosh.

Package `capi`

Superclasses `simple-pane`
`titled-object`

Initargs `:view-class` A string naming the view class to use.
`:init-function`
A function that initializes the view class.

Accessors `cocoa-view-pane-view-class`
`cocoa-view-pane-init-function`

Description The `cocoa-view-pane` class allows an instance of an arbitrary Cocoa view class to be displayed within a CAPI interface.

Note: `cocoa-view-pane` is implemented only in LispWorks for Macintosh with the Cocoa IDE.

When the pane becomes visible, the CAPI allocates and initialize a Cocoa view object using the `initargs` as follows:

- If *view-class* is specified, then it should be a string naming the Cocoa view class to allocate. Otherwise the class `NSView` is allocated.
- If *init-function* is not `nil`, then it should be a function which is called with of two arguments, the pane and a foreign pointer to the newly allocated Cocoa view object. The function should initialize the Cocoa view object in whatever way is required, including invoking the appropriate Objective-C initialization method, and return the initialized view. If *init-function* is `nil` then the Objective-C method `init` is called and the result is returned.

After the Cocoa view has been initialized, the function `cocoa-view-pane-view` can be used the retrieve it.

You can use the functions `(setf cocoa-view-pane-view-class)` and `(setf cocoa-view-pane-init-function)` to modify the *view-class* and *init-function*, but the values will be ignored if this is done after the pane becomes visible.

See the *LispWorks Objective-C and Cocoa Interface User Guide and Reference Manual* for details on using Cocoa.

Examples

The following code uses `cocoa-view-pane` to display an `NSMovieView` displaying an existing movie.

```
(defun show-movie (movie)
  (capi:contain
   (make-instance
    'cocoa-view-pane
    :view-class "NSMovieView"
    :init-function
    #'(lambda (pane view)
        (setq view
              (objc:invoke view "init"))
          (objc:invoke view "setMovie:" movie)
          view))))
```

See also `cocoa-view-pane-view`

cocoa-view-pane-view

Function

Summary Returns the Cocoa view of a `cocoa-view-pane`.

Package `capi`

Signature `cocoa-view-pane-view pane => view`

Arguments *pane* A `cocoa-view-pane`.

Values *view* A foreign pointer to a Cocoa view or `nil`.

Description The function `cocoa-view-pane-view` returns the Cocoa view for the `cocoa-view-pane` *pane* as a foreign pointer. This view is only accessible when the pane is visible and `nil` is returned in other cases.

Note: `cocoa-view-pane-view` is implemented only in LispWorks for Macintosh with the Cocoa IDE. See the *LispWorks Objective-C and Cocoa Interface User Guide and Reference Manual* for details on using Cocoa.

Example See the example in `examples/objc/movie-view.lisp`.

See also `cocoa-view-pane`

collect-interfaces*Generic Function*

Summary	Finds all interfaces of a given class.	
Package	<code>capi</code>	
Signature	<code>collect-interfaces proto &key screen current-process-first sort-by => interfaces</code>	
Arguments	<i>proto</i>	A class, class name, or an interface.
	<i>screen</i>	<code>nil</code> , the symbol <code>:any</code> , a screen, or a keyword naming a library.
	<i>current-process-first</i>	A boolean.
	<i>sort-by</i>	<code>:visible</code> OR <code>:create</code> .
Values	<i>interfaces</i>	A list.
Description	<p>The generic function <code>collect-interfaces</code> returns a list of CAPI interfaces which are instances of the class indicated by <i>proto</i>, or subclasses thereof.</p> <p>If <i>screen</i> is <code>nil</code>, the interfaces on the default screen are returned. This is the default. If <i>screen</i> is <code>:any</code>, <i>interfaces</i> includes those on any screen. If <i>screen</i> is a <code>screen</code> object, the interfaces on that screen are returned. <i>screen</i> can also be a library name, currently the accepted values are <code>:win32</code>, <code>:motif</code> and <code>:cocoa</code>.</p> <p>If interfaces on multiple screens are returned, then those on each screen are grouped together in <i>interfaces</i>.</p> <p>Amongst those for each screen, the interfaces are grouped as follows. If <i>current-process-first</i> is true, then the interfaces in the current process appear together at the beginning of the group. If <i>sort-by</i> is <code>:create</code> then these interfaces are sorted by</p>	

creation time, otherwise *sort-by* is `:visible` and they are sorted in Z-order. The interfaces of other processes appear at the end of the group, also sorted according to *sort-by*.

If *current-process-first* is `nil`, then the interfaces for each screen are sorted according to *sort-by*.

The default value of *sort-by* is `:create` and of *current-process-first* is `t`.

See also `find-interface`
`installed-libraries`

collection

Class

Summary A `collection` collects together a set of items, and provides functionality for accessing and displaying them.

Package `capi`

Superclasses `capi-object`
`callbacks`

Subclasses `choice`

Initargs `:items` The items in the collection.
`:print-function`
A function that prints an item.
`:test-function` A comparison function between two items.
`:items-count-function`
A function which returns the length of items.
`:items-get-function`
A function that returns the *n*th item.

`:items-map-function`

A function that maps a function over the items.

`:accepts-focus-p`

Specifies that the collection should accept input. The default value is `t`.

`:help-key`

An object used for lookup of help.

Accessors

`collection-items`

`collection-print-function`

`collection-test-function`

Readers

`collection-items-count-function`

`collection-items-get-function`

`collection-items-map-function`

`help-key`

Description

The main use of `collection` is as a part of the class `choice`, which provides selection capabilities on top of the collection handling, and which is used by list panels, button panels and menus amongst others.

The items in the collection are printed by `print-collection-item`.

Items can be instances of the CAPI class `item` or any Lisp object. The main difference is that non-CAPI items use the callbacks specified for the collection, whilst the CAPI `items` will use their callbacks in preference if these are specified.

By default, `items` must be a sequence, but this can be changed by specifying `items-get-function`, `items-count-function`, and `items-map-function`.

`items-get-function` should take as arguments the items and an index, and should return the indexed item. The default is `svref`.

`items-count-function` should take the items as an argument and should return the number of them.

items-map-function should take as arguments the items, a function *function* and a flag *collect-results-p*, and should call *function* on each of the items in return. If *collect-results-p* is non-`nil`, then it should also return the results of these calls in a list.

test-function should be suitable for comparing the items in your collection. For example, if there are both strings and integers amongst your *items*, you should supply *test-function* `equal`.

You can change the items using `(setf collection-items)`. Note that there is an optimization `append-items` that is sometimes useful when adding items.

accepts-focus-p and *help-key* are interpreted as described in `element`.

Examples

The following code uses `push-button-panel`, a subclass of `collection`.

```
(capi:contain (make-instance 'capi:push-button-panel
                             :items '(one two three)))

(capi:contain (make-instance
               'capi:push-button-panel
               :items '(one two three)
               :print-function 'string-capitalize))
```

The following example provides a collection with all values from 1 to 6 by providing an *items-get-function* and an *items-count-function*.

```
(capi:contain (make-instance
               'capi:push-button-panel
               :items 6
               :items-get-function
                 #'(lambda (items index) (1+ index))
               :items-count-function
                 #'(lambda (items) items)))
```

Here is an example demonstrating the use of CAPI items in a collections list of items to get more specific callbacks.

```

(defun specific-callback (data interface)
  (capi:display-message "Specific callback for ~S"
    data))

(defun generic-callback (data interface)
  (capi:display-message "Ordinary callback for ~S"
    data))

(capi:contain (make-instance
  'capi:list-panel
  :items (list (make-instance
    'capi:item
    :text "Special"
    :data 1000
    :selection-callback
    'specific-callback)
    2 3 4)
  :selection-callback 'generic-callback)
:visible-min-width 200
:visible-min-height 200)

```

See also

```

append-items
count-collection-items
get-collection-item
item
map-collection-items
print-collection-item
search-for-item

```

collection-find-next-string

Generic Function

Summary	Finds the next occurrence of the string that was previously searched for in a collection.	
Package	capi	
Signature	collection-find-next-string <i>collection</i> &key <i>set</i> => <i>index</i>	
Arguments	<i>collection</i>	A collection.
	<i>set</i>	A boolean.

Values	<i>index</i>	A non-negative integer or <code>nil</code> .
Description	<p>The generic function <code>collection-find-next-string</code> must be called after one of <code>collection-search</code>, <code>collection-find-string</code> or <code>find-string-in-collection</code> was called on <i>collection</i>. It searches for the next item in <i>collection</i> with printed representation matching the last string searched for and returns its index, or <code>nil</code> if no match is found.</p> <p>If <i>set</i> is true, then if an item matching the string is found, the selection is set to this item. <i>set</i> defaults to <code>t</code>.</p>	
See also	<p><code>collection-find-string</code> <code>collection-last-search</code> <code>collection-search</code> <code>find-string-in-collection</code></p>	

collection-find-string

Generic Function

Summary	Finds the next occurrence of a string in a collection, prompting for the string if it is not supplied.	
Package	<code>capi</code>	
Signature	<code>collection-find-string collection &key set string => index</code>	
Arguments	<i>collection</i>	A collection.
	<i>set</i>	A boolean.
	<i>string</i>	A string, or <code>nil</code> .
Values	<i>index</i>	A non-negative integer or <code>nil</code> .
Description	The generic function <code>collection-find-string</code> calls <code>find-string-in-collection</code> with <i>collection</i> and <i>set</i> .	

string is also passed if non-`nil`. If *string* is `nil`, `collection-find-string` first prompts the user for a string to pass.

set defaults to `t`.

See also `collection-search`
`find-string-in-collection`

collection-last-search

Generic Function

Summary Returns the last string searched for in a collection.

Package `capi`

Signature `collection-last-search collection => string`

Arguments *collection* A collection.

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

Description The generic function `collection-last-search` returns the last string searched for in *collection* by `collection-search` or `find-string-in-collection`.

If neither of these functions has been called on *collection*, then the return value *string* is `nil`.

See also `collection-search`
`find-string-in-collection`

collection-search

Generic Function

Summary The generic function `collection-search` calls `find-string-in-collection` with a string provided by the user.

Package	<code>capi</code>
Signature	<code>collection-search <i>collection</i> &optional <i>set</i></code>
Description	Prompts the user for a string and calls <code>find-string-in-collection</code> with <i>collection</i> , <i>set</i> and this string. <i>set</i> defaults to <code>t</code> .
See also	<code>collection</code> <code>find-string-in-collection</code>

collector-pane

Class

Summary	A <code>collector-pane</code> is an <code>editor-pane</code> which displays the output sent to a particular type of character stream called an editor stream, the contents of which are stored in an editor buffer.
Package	<code>capi</code>
Superclasses	<code>editor-pane</code>
Initargs	<code>:buffer-name</code> The name of a buffer onto an editor stream. <code>:stream</code> The editor stream to be collected.
Readers	<code>collector-pane-stream</code>
Description	A new <code>collector-pane</code> can be created to view an existing editor stream by passing the stream itself or by passing the buffer name of that stream. To create a new stream, either specify <i>buffer-name</i> which does not match any existing buffer, or do not pass <i>buffer-name</i> in which case the CAPI will create a unique buffer name for you.

To access the stream, use the reader `collector-pane-stream` on the `collector-pane`.

Note that the editor buffer “Background Output” is a buffer onto the output stream `*standard-output*`.

Examples

Here is an example that creates two collector panes onto a new stream (that is created by the first collector pane).

```
(setq collector (capi:contain
                 (make-instance 'capi:collector-pane)))

(setq *test-stream*
      (capi:collector-pane-stream collector))

(capi:contain
 (make-instance 'capi:collector-pane
                :stream *test-stream*))

(format *test-stream* "Hello World~%")
```

Finally, this example shows how to create a collector pane onto the “Background Output” stream.

```
(capi:contain (make-instance 'capi:collector-pane
                             :buffer-name "Background Output"))
```

See also

`with-random-typeout`
`map-typeout`
`unmap-typeout`

color-screen

Class

Package `capi`

Superclasses `screen`

Description This is a subclass of `screen` that gets created for color screens. It is primarily available as a means of discriminating on whether or not to use colors in an interface.

See also `element-screen`
`mono-screen`

column-layout

Class

Summary The `column-layout` lays its children out in a column.

Package `capi`

Superclasses `grid-layout`

Initargs

- `:ratios` The size ratios between the layout's children.
- `:adjust` The horizontal adjustment for each child.
- `:gap` The gap between each child.
- `:uniform-size-p`
If `t`, each child in the column has the same height.

Accessors `layout-ratios`

Description The `column-layout` lays its children out by inheriting the behavior from `grid-layout`. The *description* is a list of the layout's children, and the layout also translates the initargs *ratios*, *adjust*, *gap* and *uniform-size-p* into the `grid-layout`'s equivalent initargs *y-ratios*, *x-adjust*, *y-gap* and *y-uniform-size-p*.

description may also contain the keyword `:divider` which automatically creates a divider as a child of the `column-layout`. When specifying `:ratios` in a column with `:divider`, you should use `nil` to specify that the divider is given its minimum size, as in the example below.

Example

```
(capi:contain (make-instance
               'capi:column-layout
               :description
               (list
                (make-instance 'capi:push-button
                               :text "Press me")
                "Title"
                (make-instance 'capi:list-panel
                               :items '(1 2 3))))))
```

```

(setq column (capi:contain
              (make-instance
               'capi:column-layout
               :description
               (list
                (make-instance 'capi:push-button
                               :text "Press me")
                "Title:"
                (make-instance 'capi:list-panel
                               :items '(1 2 3)))
               :adjust :center)))

(capi:apply-in-pane-process
 column #'(setf capi:layout-x-adjust) :right column)

(capi:apply-in-pane-process
 column #'(setf capi:layout-x-adjust) :left column)

(capi:apply-in-pane-process
 column #'(setf capi:layout-x-adjust) :center column)

(flet ((make-list-panel (x y)
        (make-instance
         'capi:list-panel
         :items
         (loop for i below x
               collect i)
         :selection
         (loop for i below x by y
               collect i)
         :interaction
         :multiple-selection)))
       (capi:contain
        (make-instance
         'capi:column-layout
         :description
         (list
          (make-list-panel 100 5)
          :divider
          (make-list-panel 100 10))
         :ratios '(1 nil 2))))

```

Compatibility Note *layout-divider-default-size* and column-layout-divider are not supported in LispWorks 4.4 and later.

See also row-layout

component-name *Function*

Summary	Gets and sets the <i>component-name</i> of an <i>ole-control-pane</i> .
Package	<code>capi</code>
Signature	<code>component-name <i>pane</i> => <i>name</i></code> <code>(setf component-name) <i>name pane</i> => <i>name</i></code>
Description	<p>The function <code>component-name</code> accesses the <i>component-name</i> of an <i>ole-control-pane</i>.</p> <p>When the <i>ole-control-pane</i> is created, it automatically opens the component and inserts it.</p> <p>If <code>(setf component-name)</code> is called on a pane that is already created, any existing component is closed, and the new component is opened and inserted. <code>(setf component-name)</code> also sets the pane's <i>user-component</i> to <code>nil</code>.</p> <p>Note: <code>component-name</code> is implemented only in LispWorks for Windows. Load the functionality by <code>(require "embed")</code>.</p>
Example	See the example in <code>examples/com/ole/simple-container/doc-viewer-pair.lisp</code>
See also	<code>ole-control-pane</code>

confirm-quit *Function*

Summary	Quits the Lisp session, potentially after user confirmation.
Package	<code>capi</code>
Signature	<code>confirm-quit <i>application-name</i></code>
Arguments	<i>application-name</i> A string.

Description The function `confirm-quit` calls `quit`, potentially after confirmation from the user.

The behavior of `confirm-quit` when called within LispWorks is determined by a LispWorks user preference, which can be set by **Tools > Global Preferences... > Confirm Before Exiting**. This preference can also be set programmatically (for example in an application) by `set-confirm-quit-flag`.

If the value of the flag is `:check-editor-files` (the default), `confirm-quit` checks whether there are editor buffers which are associated with files and are modified. If there is at least one such modified buffer, `confirm-quit` prompts the user to decide between three options:

Save Changes Saves all modified buffers before quitting

Discard Changes Quits without saving

Cancel Does not save or quit

If there are no such modified buffers, `confirm-quit` simply calls `quit`.

If the flag is `nil` then `confirm-quit` simply calls `quit`.

If the flag is `t` then `confirm-quit` prompts the user. If there are unsaved buffers, the prompt is as described above, otherwise the prompt is a simple yes/no confirmer dialog.

application-name is used in the prompt to identify the application.

Note: The LispWorks IDE uses `confirm-quit`.

See also `set-confirm-quit-flag`

confirm-yes-or-no

Function

Summary The function `confirm-yes-or-no` pops up a dialog button containing a message and a **Yes** and **No** button.

Package	<code>capi</code>
Signature	<code>confirm-yes-or-no <i>format-string</i> &rest <i>format-args</i></code>
Description	<p>This pops up a dialog box containing a message and the buttons Yes and No, returns <code>t</code> when the Yes button is clicked, and <code>nil</code> when the No button is clicked. The message is obtained by applying the <i>format-string</i> and the <i>format-args</i> to the Common Lisp function <code>format</code>.</p> <p>This function is actually a convenient version of <code>prompt-for-confirmation</code>, but has the disadvantage that you cannot specify any customization arguments. For more flexibility, use <code>prompt-for-confirmation</code> itself.</p>
Example	<pre>(setq pane (capi:contain (make-instance 'capi:text-input-pane) :title "Test Interface")) (when (capi:confirm-yes-or-no "Close ~S?" pane) (capi:apply-in-pane-process pane 'capi:quit-interface pane))</pre>
See also	<p><code>prompt-for-confirmation</code> <code>display-dialog</code> <code>popup-confirmer</code></p>

confirmer-pane*Function*

Summary	Returns the pane associated with a confirmer interface.
Package	<code>capi</code>
Signature	<code>confirmer-pane <i>interface</i> => <i>pane</i></code>
Arguments	<i>interface</i> A confirmer interface displayed by <code>popup-confirmer</code> .

Values	<i>pane</i>	The <i>pane</i> argument passed to <code>popup-confirmer</code> .
Description	<p>The function <code>confirmer-pane</code> returns the pane associated with a confirmer interface that has been displayed by <code>popup-confirmer</code>.</p> <p>In most cases the programmer does not have access to this interface, but it can be passed to the confirmer's callbacks when extra buttons are added via the <i>buttons</i> argument.</p>	
See also	<code>popup-confirmer</code>	

contain

Function

Summary	Displays a window containing an element.	
Package	<code>capi</code>	
Signature	<code>contain <i>element</i> &rest <i>interface-args</i> &key <i>screen process title</i> &allow-other-keys => <i>element</i></code>	
Description	<p>The function <code>contain</code> creates and displays a container for the CAPI element <i>element</i>. <code>contain</code> returns <i>element</i> as its result.</p> <p><code>contain</code> is provided as a convenient way of testing CAPI functionality and is useful mainly during interactive development. Many of the CAPI examples use it.</p> <p>The container is created using <code>make-container</code>, which can make containers for any of the following classes:</p> <pre> simple-pane layout interface pinboard-object menu menu-item menu-component list </pre>	

In the case of a `list`, the CAPI tries to see what sort of objects they are and makes an appropriate container. For instance, if they were all `simple-panes` it would put them into a `column-layout`.

interface-args, after removing the arguments *screen* and *process*, are passed to `make-container` as the `initargs` to the interface. *title* is used as the title of the container.

The values of the arguments *screen* and *process* are passed to `display` when displaying the container.

Example

```
(capi:contain (make-instance 'capi:text-input-pane))

(capi:contain (make-instance
              'capi:column-layout
              :description `("Title:"
                            ,(make-instance
                               'capi:text-input-pane))))

(capi:contain (make-instance 'capi:menu-item)
              :title "Test")
```

See also

```
make-container
display
element
```

convert-relative-position

Function

Summary Converts a screen position from one coordinate system to another.

Package `capi`

Signature `convert-relative-position from to x y => to-x, to-y`

Arguments

<i>from</i>	A pane, interface or screen.
<i>to</i>	A pane, interface or screen.
<i>x</i>	An integer.

	<i>y</i>	An integer.
Values	<i>to-x</i>	An integer.
	<i>to-y</i>	An integer.
Description	The function <code>convert-relative-position</code> converts the position <i>x,y</i> in the coordinate system of <i>from</i> to that of <i>to</i> .	
Example	See the example file <code>examples/capi/elements/convert-relative-position.lisp</code> .	
See also	<code>top-level-interface-geometry</code> <code>with-geometry</code>	

convert-to-screen

Function

Summary	The <code>convert-to-screen</code> function finds the appropriate screen or container for a CAPI object.	
Package	<code>capi</code>	
Signature	<code>convert-to-screen &optional <i>object</i> => <i>result</i></code>	
Arguments	<i>object</i>	A CAPI object, a plist, or <code>nil</code> .
Values	<i>result</i>	A screen or a container.
Description	This finds the appropriate screen or container for the CAPI object <i>object</i> . If <i>object</i> is <code>nil</code> , <i>result</i> is the default screen. <i>object</i> defaults to <code>nil</code> .	

If *object* is a pane inside a MDI interface, then *result* is the `capi:container` of the interface, rather than the real screen, because this is more useful in most cases. To obtain the real screen, call `convert-to-screen` on the top level interface. See `document-frame` for a description of MDI interfaces.

object can be a plist. The keys below are supported on X11/Motif. Other libraries ignore them.

- `:display` The value is an X Window System display string describing the X display and screen to use. The default value is derived from the `DISPLAY` environment variable or the `-display` command-line option. If neither is supplied, the default is to use the default screen on the local host.
- `:host` The name of the host to use for the X Window System display. This key is valid only if no `:display` key/value is supplied. The default value is the local host.
- `:server-number` The number of the display server to use for the X Window System display. This key is valid only if no `:display` key/value is supplied. The default value is 0.
- `:screen-number` The number of the screen to use for the X Window System display. This key is valid only if no `:display` key/value is supplied. The default value is the default screen of the display.
- `:application-class`
The value is a string naming the application class used for X Window System resources. The default value is "Lispworks". When running a delivered LispWorks image, you

should specify the `:application-class` key if you want to provide application-specific resources.

`:command-line-args`

The value is a list of strings representing the set of command-line arguments to pass to `xtOpenDisplay`. Each string corresponds to a single argument. The default value is derived from the command line used to start Lisp.

`:fallback-resources`

The value is a list of strings representing the set of application context fallback resources to use (see `xtAppSetFallbackResources`). Each string corresponds to a single line of an X resource file.

The resources are used only when no other system resource files can be found. When running a non-delivered LispWorks image, the default value of the `:fallback-resources` key is read from the file whose name is the value of the `:application-class` key in the `app-defaults` directory of the current LispWorks library. When running a delivered LispWorks image, you should specify the `:fallback-resources` key if your application needs fallback resources.

Example `(capi:convert-to-screen)`

See also `document-frame`
 `screen`

count-collection-items

Generic Function

Summary Returns the number of items in a collection.

Package `capi`

Signature	<code>count-collection-items</code> <i>collection</i> &optional <i>representation</i>
Description	The <code>count-collection-items</code> generic function returns the number of items in <i>collection</i> by calling the <i>items-count-function</i> . <i>representation</i> defaults to <code>nil</code> . If it is non- <code>nil</code> , it is used instead of the <i>items</i> of <i>collection</i> .
Examples	The following example uses <code>count-collection-items</code> to return the number of items in a list panel. <pre>(setq list (make-instance 'capi:list-panel :items '(1 2 3 4 5))) (capi:count-collection-items list)</pre> The following example shows how to count the number of items in a specified list. <pre>(capi:count-collection-items list '(1 2))</pre>
See also	<code>collection</code> <code>get-collection-item</code> <code>search-for-item</code>

current-dialog-handle*Function*

Summary	Returns the underlying handle of the current dialog.
Package	<code>capi</code>
Signature	<code>current-dialog-handle</code> => <i>handle</i>
Values	<i>handle</i> A Microsoft Windows hwnd, a X windowid or a Cocoa window number.

Description The function `current-dialog-handle` returns the underlying handle of the current dialog. On Microsoft Windows, this is the `hwnd` of the current dialog. On X11/Motif this a windowid of the current dialog. On Mac OS X this is the value returned by the Cocoa `NSWindow`'s `windowNumber` method.

This value is useful if you want to perform some operation on the underlying handle that the CAPI does not supply.

If there is no current dialog, `current-dialog-handle` returns `nil`.

Example Press on "Get handle" to see the handle of the dialog.

```
(capi:popup- confirmer
 (make-instance
  'capi:push-button
  :text "Get handle"
  :callback-type :none
  :selection-callback
  #'(lambda ()
      (capi:display-message
       (format nil "current-dialog-handle ~a~%"
                (capi:current-dialog-handle))))))
nil
:title "A dialog")
```

See also `simple-pane-handle`

current-document

Generic Function

Summary Returns the current document of a MDI interface.

Package `capi`

Signature `current-document` *mdi-interface* => *child*

Arguments *mdi-interface* An instance of a subclass of `document-frame`.

Values	<i>child</i>	The current document of <i>mdi-interface</i> .
Description	The generic function <code>current-document</code> returns the top child interface of a MDI interface.	
See also	<code>document-frame</code>	

current-pointer-position

Function

Summary	Returns the current position of the pointer.	
Package	<code>capi</code>	
Signature	<code>current-pointer-position &key <i>relative-to</i> <i>pane-relative-p</i> => <i>x</i>, <i>y</i></code>	
Arguments	<i>relative-to</i>	A screen or a displayed interface or a CAPI pane.
	<i>pane-relative-p</i>	A boolean.
Results	<i>x</i>	An integer.
	<i>y</i>	An integer.
Description	<p>The function <code>current-pointer-position</code> returns the current <i>x,y</i> position of the pointer on the screen of <i>relative-to</i>, which defaults to the current screen.</p> <p>If <i>pane-relative-p</i> is true then the position is returned relative to <i>relative-to</i>, otherwise it is returned relative to the screen. The default value of <i>pane-relative-p</i> is <code>t</code>.</p>	
See also	<code>interface</code> <code>screen</code>	

current-printer

Function

Summary	Returns the currently selected printer object.
Package	<code>capi</code>
Signature	<code>current-printer &key <i>interactive</i> => <i>printer</i></code>
Arguments	<i>interactive</i> A boolean.
Values	<i>printer</i> A printer, or <code>nil</code> .
Description	The <code>current-printer</code> function returns the currently selected printer object for the default library. If <i>interactive</i> is non- <code>nil</code> and there is no current printer, a confirmer is displayed warning the user and <i>printer</i> is <code>nil</code> . The default value of <i>interactive</i> is <code>nil</code> .
See also	<code>page-setup-dialog</code> <code>set-printer-options</code>

default-editor-pane-line-wrap-marker

Variable

Summary	The default line wrap marker for editor panes.
Package	<code>capi</code>
Initial Value	<code>#\!</code>
Description	The variable <code>*default-editor-pane-line-wrap-marker*</code> provides the default value for the <i>line-wrap-marker</i> of an <code>editor-pane</code> . The value should be a <code>character</code> object, or <code>nil</code> .
See also	<code>editor-pane</code>

default-library	<i>Function</i>
Summary	Returns the default library.
Package	<code>capi</code>
Signature	<code>default-library => <i>library</i></code>
Values	<i>library</i> A library name.
Description	<p>The function <code>default-library</code> returns a keyword naming the the default library.</p> <p>On Unix/Linux platforms, currently the only library available is <code>:motif</code>, hence this is the default library.</p> <p>On Microsoft Windows platforms, currently the only library available is <code>:win32</code>, hence this is the default library.</p> <p>On Mac OS X platforms, currently the only library available is <code>:cocoa</code>, hence this is the default library.</p>
See also	<code>installed-libraries</code>

define-command	<i>Macro</i>
Summary	The <code>define-command</code> macro defines an alias for a mouse or keyboard gesture that can be used in the input model of an output pane.
Package	<code>capi</code>
Signature	<code>define-command <i>name gesture &key translator host</i></code>
Description	The macro <code>define-command</code> defines an alias for a mouse or keyboard gesture that can then be used in <code>output-pane</code> 's input models. The <i>name</i> is the name of the alias and the <i>gesture</i> is one of the gestures accepted by <code>output-pane</code> . The

translator is a function that gets passed the arguments that would be passed to the callback, and returns a list of arguments to be passed to the callback along with the output-pane (which will be the first argument). The *host* indicates which platforms this gesture should apply for (it defaults to all platforms).

For a full description of the gesture syntax, see `output-pane`.

Examples

Firstly, here is an example of defining a command which maps onto a gesture.

```
(defun gesture-callback (output-pane x y)
  (capi:display-message
   "Pressed ~S at (~S,~S)"
   output-pane x y))

(capi:define-command :select (:button-1 :press))

(capi:contain (make-instance
               'capi:output-pane
               :input-model '(:select
                              gesture-callback))))
```

Here is a more complicated example demonstrating the use of *translator* to affect the arguments passed to a callback.

```
(capi:define-command
 :select-object (:button-1 :press)
 :translator #'(lambda (output-pane x y)
                 (let ((object
                       (capi:pinboard-object-at-position
                        output-pane x y)))
                   (when object
                     (list object))))))

(defun object-select-callback (output-pane
                              &optional object)
  (when object (capi:display-message
                "Pressed on ~S in ~S"
                object output-pane)))
```

```
(setq pinboard
  (capi:contain (make-instance
                'capi:pinboard-layout
                :input-model '(:select-object
                              object-select-callback))))

(make-instance 'capi:item-pinboard-object
              :text "Press Me!"
              :parent pinboard
              :x 10 :y 20)

(make-instance 'capi:line-pinboard-object
              :parent pinboard
              :start-x 20 :start-y 50
              :end-x 120 :end-y 150)
```

There is a further example in the file
`capi/output-panes/commands.lisp`.

See also

```
output-pane
invoke-command
invoke-untranslated-command
```

define-interface

Macro

Summary	The <code>define-interface</code> macro defines subclasses of <code>interface</code> .
Package	<code>capi</code>
Signature	<code>define-interface</code> <i>name superclasses slots &rest options</i>
Description	The macro <code>define-interface</code> is used to define subclasses of <code>interface</code> , which when created with <code>make-instance</code> has the specified panes, layouts and menus created automatically. If non- <code>nil</code> , <i>superclasses</i> must include <code>interface</code> or a subclass of it. <code>define-interface</code> is essentially a version of <code>defclass</code> which accepts the following extra options:

<code>:panes</code>	Descriptions of the interface's panes.
<code>:layouts</code>	Descriptions of the interface's layouts.
<code>:menus</code>	Descriptions of the interface's menus.
<code>:menu-bar</code>	A list of menus for the interface's menu bar.
<code>:definition</code>	Options to alter <code>define-interface</code> .

The class options `:panes`, `:layouts` and `:menus` add extra slots to the class that will contain the CAPI object described in their description. Within the scope of the extra options, the slots themselves are available by referencing the name of the slot, and the interface itself is available with the variable `capl:interface`. Each of the slots can be made to have readers, writers or accessors by passing the appropriate `defclass` keyword as one of the optional arguments in the description. Therefore, if you need to find a pane within an interface instance, you can provide an accessor, or simply use `with-slots`.

The `:panes` option is a list of pane descriptions of the following form

```
(:panes
  (slot-name pane-class initargs)
  ...
  (slot-name pane-class initargs)
)
```

where *slot-name* is a name for the slot, *pane-class* is the class of the pane being included in the interface, and *initargs* are the initialization arguments for the pane - the allowed forms are described below.

The `:layouts` option is a list of layout descriptions of the following form

```
(:layouts
  (slot-name layout-class children initargs)
  ...
  (slot-name layout-class children initargs)
)
```

where *slot-name* is a name for the slot, *layout-class* specifies the type of layout, *children* is a list of children for the layout, and *initargs* are the initialization arguments for the layout - the allowed forms are described below. The primary layout for the interface defaults to the first layout described, but can be specified as the `:layout` initarg to the interface. If no layouts are specified, then the CAPI will place all of the defined panes into a column layout and make that the primary layout.

The `:menus` option is a list of menu and menu component descriptions of the following form

```
( :menus
  (slot-name title descriptions initargs)
  ...
  (slot-name title descriptions initargs)
)
```

slot-name is the slot name for each menu or menu component.

title is the menu's title, the keyword `:menu`, or the keyword `:component`.

descriptions is a list of menu item descriptions. Each menu item description is either a title, a slot name for a menu, or a list of items containing a title, descriptions, and a list of initialization arguments for the menu item.

initargs are the initialization arguments for the menu.

The values given in *initargs* under `:panes`, `:layouts` and `:menus` can be lists of the form

```
(:initarg keyword-name)
(:initarg key-spec)
(:initarg key-spec initarg-value)
```

key-spec := var | (var) | (var initform) | ((keyword-name var) | ((keyword-name var) initform)

keyword-name := any keyword

key-spec is interpreted as in the `&key` symbol of ordinary Common Lisp lambda lists. When this form of value is used, the specified *keyword-name* is added as an extra initarg to the class defined by the `define-interface` form.

If *key-spec* is followed by *initarg-value*, then its value is used as the initarg of the pane. Otherwise the value from *key-spec* is used.

Additionally *initargs* may contain the keyword argument `:make-instance-extra-apply-args` which is useful when you want to supply initargs to the pane *slot-name* when the interface is initialized. The value *make-instance-extra-apply-args* should be a keyword which becomes an extra initarg to the interface class *name*. The value of that initarg should be a list of pane initargs and values which is passed when the pane is initialized. For an example, see `examples/capi/applications/argument-passing.lisp`.

The `:menu-bar` option is a list of slot names, where each slot referred to contains a menu that should appear on the menu bar.

The `:definition` option is a property list of arguments which `define-interface` uses to change the way that it behaves. Currently there is only one definition option:

`:interface-variable`

The name of the variable containing the interface.

Examples Firstly, a couple of pane examples:

```

(capi:define-interface test1 ()
  ()
  (:panes
    (text capi:text-input-pane)
    (:default-initargs :title "Test1"))
(capi:display (make-instance 'test1))
(capi:define-interface test2 ()
  ()
  (:panes
    (text capi:text-input-pane)
    (buttons capi:button-panel :items '(1 2 3)
      :reader test2-buttons))
  (:layouts
    (main-layout capi:column-layout '(text buttons)))
  (:default-initargs :title "Test2"))

(test2-buttons
  (capi:display (make-instance 'test2)))

```

Here are a couple of menu examples:

```

(capi:define-interface test3 ()
  ()
  (:menus
    (color-menu "Colors" (:red :green :blue)
      :print-function 'string-capitalize))
  (:menu-bar color-menu)
  (:default-initargs :title "Test3"))

(capi:display (make-instance 'test3))

(capi:define-interface test4 ()
  ()
  (:menus
    (colors-menu "Colors"
      ((:component
        (:red :green :blue)
        :interaction :single-selection
        :print-function
        'string-capitalize)
      more-colors-menu))
    (more-colors-menu "More Colors"
      (:pink :yellow :cyan)
      :print-function
      'string-capitalize))
  (:menu-bar colors-menu)
  (:default-initargs :title "Test4"))

```

```
(capi:display (make-instance 'test4))
```

This example demonstrates inheritance amongst subclasses of interface:

```
(capi:define-interface test5 (test4 test1)
  ()
  (:default-initargs :title "Test5"))

(capi:display (make-instance 'test5))
```

The next three examples illustrate the use of `:initarg` in initarg specifications for `:panes`.

Here we initialize the `:selected-items` initarg of the pane `foo` to the value passed by `:select` when making the interface object, or `nil` otherwise:

```
(capi:define-interface init1 () ()
  (:panes
   (foo
    capi:list-panel
    :items '(0 1 2 3 4)
    :visible-min-height '(:character 5)
    :interaction :multiple-selection
    :selected-items (:initarg select))))

(capi:contain (make-instance 'init1
                           :select '(1 3)))

(capi:contain (make-instance 'init1))
```

Here we initialize the `:selected-items` initarg of pane `foo` to the value passed by `:select` initarg when making the interface object, or `(1 3)` otherwise:

```
(capi:define-interface init2 () ()
  (:panes
   (foo
    capi:list-panel
    :items '(0 1 2 3 4)
    :visible-min-height '(:character 5)
    :interaction :multiple-selection
    :selected-items
    (:initarg (select '(1 3))))))

(capi:contain (make-instance 'init2))
```

Here we increment the indices passed in the interface's `:select` `initarg` before passing them in the `:selected-items` `initarg` of pane `foo`:

```
(capi:define-interface init3 () ()
  (:panes
   (foo
    capi:list-panel
    :items '(0 1 2 3 4)
    :visible-min-height '(:character 5)
    :interaction :multiple-selection
    :selected-items
    (:initarg select
     (mapcar '1+ select))))))

(capi:contain (make-instance 'init3
                             :select '(1 3)))
```

There are many more examples in the directory `examples/capi/`.

See also `interface`
`layout`
`menu`

define-layout

Macro

Summary The macro `define-layout` creates new classes of `layout`.

Package `capi`

Signature `define-layout` *name superclasses slots &rest options*

Description The macro `define-layout` is used to create new classes of `layout`. The macro is essentially the same as `defclass` except that its default superclass is `layout`.

To implement a new class of `layout`, methods need to be provided for the following generic functions:

`interpret-description`

Translate the layout's child descriptions.

`calculate-constraints`

Calculate the constraints for the layout.

`calculate-layout`

Layout the children of the layout.

See also

`interpret-description`

`calculate-constraints`

`calculate-layout`

`layout`

define-ole-control-component

Macro

Summary Defines a class that implements the OLE Control protocol for a CAPI pane.

Package `capi`

Signature `define-ole-control-component class-name (superclass-name*) slots &rest class-options`

Description The macro `define-ole-control-component` defines an Automation component class *class-name* that also implements the OLE Control protocols and other named interfaces or a coclass. This allows a CAPI pane to be embedded in an OLE Control container implemented outside LispWorks.

Each *superclass-name* argument specifies a direct superclass of the new class, which can be any `standard-class` provided that certain standard classes are included somewhere in the overall class precedence list. These standard classes depend on the other options and provide the default superclass list if none is specified. The following standard classes are available:

`ole-control-component` is always needed and provides an implementation of the OLE Control protocol.

`com:standard-i-dispatch` is always needed and provides a complete implementation of the i-dispatch interface, based on the type information in a type library.

`com:standard-i-connection-point-container` is needed if there are any source interfaces specified (via the `:coclass` or `:source-interfaces` options). This provides a complete implementation of the Connection Point protocols, used to support events.

`slots` is a list of standard `defclass` slot definitions.

`class-options` are standard `defclass` options. In addition the following options are recognized:

```
(:coclass coclass-name)
```

```
(:interfaces interface-name*)
```

```
(:source-interfaces interface-name*)
```

See `com:define-automation-component` in the *LispWorks COM/Automation User Guide and Reference Manual* for details of these options.

Typically the `:pane-function` and `:create-callback` `initargs` are supplied using the `:default-initarg` option.

Implementations of the methods in the `:coclass` and `:interfaces` options should be defined using `com:define-com-method`, `com:define-dispinterface-method` or `com:com-object-dispinterface-invoke`.

Note: `define-ole-control-component` is implemented only in LispWorks for Windows. Load the functionality by

```
(require "embed")
```

See also

`ole-control-component`

define-menu

Macro

Summary	The <code>define-menu</code> macro defines a menu function.
Package	<code>capi</code>
Signature	<code>define-menu <i>function-name</i> (<i>self</i>) <i>title</i> <i>menu-body</i> &rest <i>menu-options</i></code>
Description	The macro <code>define-menu</code> defines a function called <i>function-name</i> with a single argument <i>self</i> that will make a menu. The parameters <i>title</i> , <i>menu-body</i> and <i>menu-options</i> take the same form as the <code>:menus</code> section of <code>define-interface</code> .
Example	<pre>(capi:define-menu make-test-menu (self) "Test" ("Item1" "Item2" (:component ("Item3" "Item4") :interaction :single-selection) (:menu ("Item5" "Item6") :title "More Items")))) (setq interface (make-instance 'capi:interface)) (setf (capi:interface-menu-bar-items interface) (list (make-test-menu interface))) (capi:display interface)</pre>
See also	<code>define-interface</code> <code>menu</code>

destroy

Generic Function

Summary	Closes a window and calls the <i>destroy-callback</i> .
---------	---

Package	<code>capi</code>
Signature	<code>destroy</code> <i>interface</i>
Description	<p>The generic function <code>destroy</code> closes the window associated with <i>interface</i>, and then calls the interface's <i>destroy-callback</i> if it has one.</p> <p>There is a complementary function <code>quit-interface</code> which calls the interface's <i>confirm-destroy-function</i> to confirm that the destroy should be done, and it is advisable to always use this unless you want to make sure that the interface's <i>confirm-destroy-function</i> is ignored.</p> <p>Note: <code>destroy</code> must only be called in the process of <i>interface</i>. Menu callbacks on <i>interface</i> will be called in that process, but otherwise you probably need to use <code>execute-with-interface</code> OR <code>apply-in-pane-process</code>.</p>
Example	<pre>(setq interface (capi:display (make-instance 'capi:interface :title "Test Interface" :destroy-callback #'(lambda (interface) (capi:display-message "Quitting ~S" interface)))))) (capi:apply-in-pane-process interface 'capi:destroy interface)</pre>
See also	<p><code>interface</code> <code>quit-interface</code> <code>*update-screen-interfaces-hooks*</code></p>

detach-simple-sink

Function

Summary Detaches a previously-attached simple sink object.

Package	<code>capi</code>
Signature	<code>detach-simple-sink <i>sink pane</i></code>
Arguments	<p><i>sink</i> A class instance.</p> <p><i>pane</i> An <code>ole-control-pane</code>.</p>
Description	<p>The function <code>detach-simple-sink</code> detaches a sink that was previously attached to the active component in the <code>ole-control-pane</code> <i>pane</i> by a call to <code>attach-simple-sink</code>.</p> <p><i>sink</i> is the value returned by <code>attach-simple-sink</code> when the sink was attached.</p> <p><i>pane</i> is an <code>ole-control-pane</code> which is the pane where the component is.</p> <p>Attached sinks are automatically disconnected when the object is closed.</p> <p>Note: this function is implemented only in LispWorks for Windows. Load the functionality by <code>(require "embed")</code>.</p>
See also	<p><code>attach-simple-sink</code></p> <p><code>ole-control-pane</code></p>

detach-sink

Function

Summary	Detaches a previously-attached sink.
Package	<code>capi</code>
Signature	<code>detach-sink <i>sink pane interface-name</i></code>
Arguments	<p><i>sink</i> A class instance.</p> <p><i>pane</i> An <code>ole-control-pane</code>.</p> <p><i>interface-name</i> A refguid or the symbol <code>:default</code>.</p>

Description The function `detach-sink` detaches a sink which was previously attached to the active component in the `ole-control-pane` *pane*.

sink is an instance of a class that implements the interface *interface-name*.

pane is an `ole-control-pane` which is the pane where the component is.

interface-name is either a string naming a source interface that the component in *pane* supports or `:default` to disconnect from the default source interface.

Attached sinks are automatically disconnected when the object is closed.

Note: this function is implemented only in LispWorks for Windows. Load the functionality by `(require "embed")`.

See also `attach-sink`
 `ole-control-pane`

display *Function*

Summary The `display` function displays a CAPI interface on a specified screen.

Package `capi`

Signature `display interface &key screen owner window-styles process => interface`

Arguments *interface* A CAPI interface.

screen A screen, or any argument accepted by `convert-to-screen`.

owner A CAPI interface.

window-styles A list of keywords.

	<i>process</i>	On Windows or Motif, a CAPI process, <code>t</code> or <code>nil</code> . On Cocoa, this argument is not supported.
Values	<i>interface</i>	A CAPI interface.
Description		<p>The function <code>display</code> displays the CAPI interface <i>interface</i> on the specified <i>screen</i> (or the current one if not supplied).</p> <p>If <i>process</i> is not supplied, then if <i>owner</i> is supplied <i>interface</i> runs in <i>owner</i>'s process, otherwise interface runs in the process of the parent of <i>interface</i> if it is a <code>document-container</code>, or in a new process created for <i>interface</i> if not.</p> <p>On Windows and Motif, if <i>process</i> is <code>t</code>, then <i>interface</i> runs in a newly-created process. If <i>process</i> is <code>nil</code>, <i>interface</i> runs in the current process. Otherwise <i>process</i> is expected to be a CAPI process, and <i>interface</i> runs in it. A CAPI process is a <code>mp:process</code> which was created by calling <code>display</code>. You can pass only a CAPI process as <i>process</i>, because it needs to handle messages using the LispWorks event loop. The default value of <i>process</i> is <code>t</code>.</p> <p>On Cocoa, all CAPI interfaces run in the Cocoa Event Loop process (which is the main thread of LispWorks) and therefore the <i>process</i> argument is not supported. If the value of <i>process</i> is any process other than the Cocoa Event Loop process an error is signalled.</p> <p><i>owner</i> specifies an owner for <i>interface</i>, which should be another CAPI interface. <i>interface</i> inherits a number of attributes from <i>owner</i>, including the default process, default screen and default display state.</p> <p><i>window-styles</i>, if supplied, sets the <i>window-styles</i> slot of <i>interface</i>. See <code>interface</code> for information about <i>window-styles</i>.</p> <p><code>display</code> returns its <i>interface</i> argument.</p> <p>Note: Use the function <code>contain</code> to display objects other than interfaces.</p>

Note: A generic function `interface-display` is called immediately after `display` displays an interface. You can add post-display code by defining your own `:after` method.

Example

```
(capi:display (make-instance 'capi:interface
                             :title "Test"))
```

See also `contain`
`convert-to-screen`
`display-dialog`
`document-container`
`execute-with-interface`
`interface`
`interface-display`
`quit-interface`
`*update-screen-interfaces-hooks*`

display-dialog

Function

Summary The `display-dialog` function displays a CAPI interface as a dialog box.

Package `capi`

Signature `display-dialog` *interface &key screen focus modal*
owner x y position-relative-to continuation =>
result, okp

Arguments *interface* A CAPI interface.
screen A screen.
focus A pane of *interface*.
modal A boolean.
owner A pane.

	<i>x, y</i>	Real numbers representing coordinates, or keywords or lists specifying an adjusted position.
	<i>position-relative-to</i>	:owner or nil.
	<i>continuation</i>	A function or nil.
Values	<i>result</i>	An object.
	<i>okp</i>	A boolean.
Description	<p>This is a complementary function that displays the CAPI <i>interface interface</i> as a dialog box.</p> <p><i>screen</i> is the <i>screen</i> for the dialog to be displayed on.</p> <p><i>focus</i> should be the pane within the interface that should be given the focus initially. If a focus is not supplied, then it lets the window system decide.</p> <p><i>modal</i> indicates whether or not the dialog takes over all input to the application. The default is <i>t</i>.</p> <p><i>owner</i> specifies an owner window for the dialog. See the "Prompting for Input" chapter in the <i>LispWorks CAPI User Guide</i> for details.</p> <p>If <i>x</i> and <i>y</i> are numbers they specify the coordinates of the dialog. Alternatively <i>x</i> and <i>y</i> can be keywords like <i>:left</i> and <i>:top</i>, or lists like (<i>:left 100</i>), (<i>:bottom 50</i>) and so on.. These values cause the dialog to be positioned relative to its owner in the same way as the <i>adjust</i> argument to <i>pane-adjusted-position</i>. The default location is at the center of the dialog's owner.</p> <p><i>position-relative-to</i> has a default value <i>:owner</i>, meaning that <i>x</i> and <i>y</i> are relative to dialog's owner. The value <i>nil</i> means that <i>x</i> and <i>y</i> are relative to the screen.</p>	

If *continuation* is non-`nil`, then it must be a function with a lambda list that accepts two arguments. The *continuation* function is called with the values that would normally be returned by `display-dialog`. On Cocoa, passing *continuation* causes the dialog to be made as a window-modal sheet and `display-dialog` returns immediately, leaving the dialog on the screen. The `with-dialog-results` macro provides a convenient way to create a *continuation* function.

The values returned depend on how the dialog is dismissed. Typically a user gesture will trigger a call to `abort-dialog`, causing the values `nil, nil` to be returned or to `exit-dialog` causing the values *result*, `t` to be returned, where *result* is the argument to `exit-dialog`. If *continuation* is non-`nil`, then the returned values are always `:continuation, nil`.

The CAPI also provides `popup-confirmer` which gives you the standard **OK** and **Cancel** button functionality.

Note: if you need to replace one dialog with another, you can use `display-replacable-dialog` and `replace-dialog`.

Example

```
(capi:display-dialog
  (capi:make-container
    (make-instance 'capi:push-button-panel
      :items '("OK" "Cancel")
      :callback-type :data
      :callbacks '(capi:exit-dialog
                  capi:abort-dialog))
    :title "Empty Dialog"))
```

There are further examples in the directory `examples/capi/dialogs/`.

See also

```
abort-dialog
display
display-replacable-dialog
exit-dialog
interface
```

```
popup-confirmer
with-dialog-results
*update-screen-interfaces-hooks*
```

display-errors

Macro

Summary	Displays a message if an error is signalled.
Package	<code>capi</code>
Signature	<code>display-errors &body <i>body</i></code>
Description	The macro <code>display-errors</code> executes the code of <i>body</i> inside a <code>handler-case</code> form. If an error is signalled inside <i>body</i> , a message is displayed and the debugger is not entered.

display-message

Function

Summary	The function <code>display-message</code> displays a message on the current CAPI screen.
Package	<code>capi</code>
Signature	<code>display-message <i>format-string</i> &rest <i>format-args</i></code>
Description	The function <code>display-message</code> creates a message from the arguments using <code>format</code> , and then displays it on the current CAPI screen. Note: If you need to make a window-modal sheet on Cocoa, then use the function <code>prompt-with-message</code> .
Example	<pre>(capi:display-message "Current screen = ~S" (capi:convert-to-screen))</pre>

See also `prompt-with-message`
`display-message-for-pane`
`display-dialog`

display-message-for-pane

Function

Summary The function `display-message-for-pane` displays a message on the same screen as a specified pane.

Package `capi`

Signature `display-message-for-pane` *pane* *format-string* &rest *format-args*

Description The function `display-message-for-pane` creates a message from the arguments using `format`, and then displays it on the same screen as *pane*.

Note: If you need to make a window-modal sheet on Cocoa, then use the function `prompt-with-message`.

Example

```
(setq pane (capi:contain (make-instance
                          'capi:text-input-pane)))
(capi:display-message-for-pane pane
                              "Just created ~S" pane)
```

Compatibility Note The function `display-message-on-screen` is retained for compatibility with previous versions of LispWorks. It is a synonym for `display-message-for-pane`.

See also `prompt-with-message`
`display-message`

display-pane

Class

Summary The class `display-pane` is a pane that displays several lines of text.

See also `display-pane`
`editor-pane`
`text-input-pane`
`title-pane`

display-popup-menu

Function

Summary Displays a popup menu.

Package `capi`

Signature `display-popup-menu menu &key owner x y button => result`

Arguments *menu* A menu.
 owner A pane.
 x The horizontal coordinate of *menu*'s position relative to *owner*.
 y The vertical coordinate of *menu*'s position relative to *owner*.
 button The mouse button that raises the menu.

Description The function `display-popup-menu` displays the menu *menu* at position *x,y*. `display-popup-menu` should be used in response to the user clicking a mouse button, and is typically used to implement contextual ("right button") menus.

The user may select an item in the menu, in which case the item's *selection-callback* is invoked, and `display-popup-menu` returns `t`.

Alternatively the user may cancel the menu, by clicking elsewhere or pressing the `Escape` key. In this case, `display-popup-menu` returns `nil`.

owner specifies the owner of the menu, that is, a pane that the menu is associated with. If *owner* is not supplied the system tries to find the appropriate owner, which usually suffices.

x and *y* default to the horizontal and vertical coordinates, relative to *owner*, of the location of the mouse pointer.

button defaults to `:button-3`.

Example

```
(defun popup-test-menu (pinboard x y &optional gspec)
  (capi:display-popup-menu
   (make-instance 'capi:menu :items '(1 2 3))
   :owner pinboard :x x :y y))

(capi:contain
 (make-instance 'capi:pinboard-layout
                :input-model
                '(:post-menu popup-test-menu))
 :visible-min-width 100
 :visible-min-height 100))
```

See also

`menu`
`pinboard-layout`

display-replacable-dialog

Function

Summary Displays a replacable dialog.

Package `capi`

Signature `display-replacable-dialog interface &rest args => result`

Arguments *interface* An interface.

args Other arguments as for `display-dialog`.

Values *result* The value returned by the dialog.

Description The function `display-replacable-dialog` displays a dialog that can be replaced by another dialog.

interface is a CAPI interface to be displayed as a dialog.

The arguments *args* are interpreted the same as the arguments to `display-dialog`, except that *modal* is ignored. `display-replacable-dialog` displays the dialog like `display-dialog`.

Within the scope of `display-replacable-dialog` (that is, inside the callbacks) the programmer can call `replace-dialog` which replaces the dialog by a new dialog and destroys the existing one. There can be many calls to `replace-dialog` inside the same scope of `display-replacable-dialog`.

`display-replacable-dialog` returns the last dialog that was displayed.

Inside `display-replacable-dialog`, the functions that use the current dialog, such as `exit-dialog` and `abort-dialog`, work in the same way that they work inside `display-dialog`, except that they don't affect the return value of `display-replacable-dialog`.

See also `abort-dialog`
`display-dialog`
`exit-dialog`
`replace-dialog`

display-tooltip

Generic Function

Package	<code>capi</code>
Signature	<code>display-tooltip <i>output-pane</i> &key <i>x y text</i> => <i>result</i></code>
Arguments	<p><i>output-pane</i> An instance of a subclass of <code>output-pane</code>.</p> <p><i>x</i> The horizontal coordinate of the tooltip position.</p>

y The vertical coordinate of the tooltip position.

text The help text.

Description The generic function `display-tooltip` displays *text* as tooltip help at position *x,y* in *output-pane*.

Example See the example file
`examples/capi/graphics/pinboard-help.lisp`

docking-layout

Class

Summary A class that implements docking of panes.

Package `capi`

Superclasses `simple-layout`

Initargs

- `:items` A list of pane specifications. The panes become the items in the layout.
- `:controller` A controller for the layout, which can make multiple `docking-layouts` work together.
- `:docking-test-function`
A function controlling whether a pane can be docked in a `docking-layout`.
- `:docking-callback`
A function called when a pane is docked or undocked.
- `:divider-p` A boolean allowing a visible edge around the layout.
- `:orientation` One of `:horizontal` or `:vertical`.

Accessors	<code>docking-layout-controller</code> <code>docking-layout-divider-p</code> <code>docking-layout-docking-test-function</code> <code>docking-layout-items</code>
Readers	<code>docking-layout-orientation</code>
Description	<p>The class <code>docking-layout</code> defines a region in which panes can be docked and undocked. The undocking functionality works only in LispWorks for Windows.</p> <p>If <i>controller</i> is non-<code>nil</code>, it must be a controller object as returned by a call to <code>make-docking-layout-controller</code>. In this case the <code>docking-layout</code> is one of a group of <code>docking-layouts</code> which share that same controller, known as the Docking Group. The panes that can be docked and undocked are shared between the members of the Docking Group. If <i>controller</i> is <code>nil</code> (the default value), the <code>docking-layout</code> is in a Docking Group of one.</p> <p>A pane <i>pane</i> is dockable in a Docking Group when it is an item of any member of the Docking Group. This is the case when it is one of the <i>items</i> passed to <code>make-instance</code> for some member of the group, or it has been set in some member by <code>(setf docking-layout-items)</code>. The user can dock and undock <i>pane</i> in any member of the Docking Group. You can change the dockable status of panes programmatically by <code>(setf docking-layout-items)</code>. You can query a pane's docked and visible status in a <code>docking-layout</code> by <code>docking-layout-pane-docked-p</code> and <code>docking-layout-pane-visible-p</code>. You can change a pane's docked and visible status in a <code>docking-layout</code> by <code>(setf docking-layout-pane-docked-p)</code> and <code>(setf docking-layout-pane-visible-p)</code>.</p> <p>By default, the context menu allows the user to alter the visibility status of each of the panes in the Docking Group.</p>

items is a list of pane specifications. Each specification in the list is either an atom denoting a pane, or a list wherein the car is an object denoting a pane and the cdr is a plist of options and values. The object denoting the pane can be:

- The pane itself
- A symbol naming a slot in the interface which contains the `docking-layout`. The value in that slot, which must be a pane, is used. Typically the slot name is defined in the `:panes` or `:layouts` class option in the `define-interface` form.
- A string, denoting a `title-pane` with that text.
- A list, wherein the car is the name of a pane class and the cdr is a list of initialization arguments for that class. This denotes the pane created by applying `make-instance` to the list. Note that in this case the list cannot be the item in the *items* list, because it would be wrongly interpreted as a list wherein the car denotes a pane directly and the cdr is a plist of options and values.

When an item in the *items* list is a list, the cdr is a plist of options and values, which can contain these options:

- | | |
|-------------------------|--|
| <code>:title</code> | A string which is title associated with the pane. This is used when the pane is presented to the user, for example in the default context menu. |
| <code>:docked-p</code> | A boolean specifying whether the pane should be docked. The default value is <code>t</code> . When a pane is not docked and is visible, it is displayed in its own window. |
| <code>:visible-p</code> | A boolean specifying whether the pane is visible. The default value is <code>t</code> . |

`:undocked-geometry`

A list of four integers specifying the geometry of the pane when undocked, as (*x y width height*).

`:start-new-line-p`

A boolean specifying whether to place the pane on a new line in the `docking-layout`. The default value is `nil`.

`docking-layout-items` always returns the items as lists, with the `cdr` containing the options and values.

docking-test-function is a function of two arguments with a boolean return value. When the user attempts to dock a pane *pane* in the `docking-layout`, *docking-test-function* is called with the `docking-layout` and *pane*. If it returns `nil`, *pane* is not docked. If it returns `true`, *pane* is docked. The default behavior is that all panes under the controller which is the *controller* in this `docking-layout`, and only these panes, can be docked.

docking-callback, if non-`nil`, is a function of three arguments: the `docking-layout`, the pane and a boolean. This third argument is `t` when the pane is docked, and `nil` when the pane is undocked. The default value of *docking-callback* is `nil`.

divider-p controls whether a visible edge is drawn around the border of the `docking-layout`. The default value is `nil`.

orientation specifies whether the items are laid out horizontally or vertically. The default value is `:horizontal`.

Examples

See the file `examples/capi/layouts/docking-layout.lisp`

See also

`docking-layout-pane-docked-p`
`docking-layout-pane-visible-p`

docking-layout-pane-docked-p

Function

Package	<code>capi</code>
Signature	<code>docking-layout-pane-docked-p</code> <i>docking-layout</i> <i>pane</i> &key <i>anywhere</i> => <i>dockedp</i>
Signature	<code>(setf docking-layout-pane-docked-p)</code> <i>dockedp</i> <i>docking-layout</i> <i>pane</i> => <i>dockedp</i>
Arguments	<i>docking-layout</i> An instance of <code>docking-layout</code> or a sub-class. <i>pane</i> A pane. <i>anywhere</i> A boolean.
Values	<i>dockedp</i> A boolean.
Description	<p>The function <code>docking-layout-pane-docked-p</code> returns a boolean indicating whether <i>pane</i> is currently docked.</p> <p>If <i>anywhere</i> is <code>t</code>, <i>dockedp</i> is true if <i>pane</i> is docked in any member of the Docking Group of <i>docking-layout</i>. If <i>anywhere</i> is <code>nil</code>, <i>dockedp</i> is true only if <i>pane</i> is docked in <i>docking-layout</i> itself. The default value of <i>anywhere</i> is <code>nil</code>.</p> <p><code>(setf docking-layout-pane-docked-p)</code> may be used to change the docking state of <i>pane</i> in <i>docking-layout</i> only when <i>pane</i> is dockable in the Docking Group of <i>docking-layout</i>.</p>
See also	<code>docking-layout</code>

docking-layout-pane-visible-p

Function

Package	<code>capi</code>
Signature	<code>docking-layout-pane-visible-p</code> <i>docking-layout</i> <i>pane</i> => <i>visiblep</i>

Signature	<code>(setf docking-layout-pane-visible-p) <i>visiblep</i> <i>docking-layout</i> <i>pane</i> => <i>visiblep</i></code>
Arguments	<i>docking-layout</i> An instance of <code>docking-layout</code> or a subclass. <i>pane</i> A pane.
Values	<i>visiblep</i> A boolean.
Description	The function <code>docking-layout-pane-visible-p</code> returns a boolean indicating whether <i>pane</i> is currently visible in the Docking Group of <i>docking-layout</i> . <i>pane</i> may be docked in any member of the Docking Group, or undocked. <code>(setf docking-layout-pane-visible-p)</code> may be used to change the visibility of <i>pane</i> in <i>docking-layout</i> only when <i>pane</i> is dockable in the Docking Group of <i>docking-layout</i> .
See also	<code>docking-layout</code>

document-container

Class

Package	<code>capi</code>
Superclasses	<code>capi-object</code>
Readers	<code>screen-interfaces</code>
Description	The class of the container in a <code>document-frame</code> . A document container has some screen-like functionality, responding to <code>screen-internal-geometry</code> and <code>screen-active-interface</code> . This works only in LispWorks for Windows.

See also `display`
`document-frame`
`screen-active-interface`
`screen-internal-geometry`

document-frame

Class

Summary The class `document-frame` is used to implement MDI.
This works only in LispWorks for Windows.

Package `capi`

Superclasses `interface`

Readers `document-frame-container`

Description The class `document-frame` is used to implement Multiple-Document Interface (MDI) which is a standard technique on Microsoft Windows (see the MSDN for documentation).

To use MDI in the CAPI, define an interface class that inherits from `document-frame`, and use the two special slots `capi:container` and `capi:windows-menu` as described below.

In your interface's layouts, use the symbol `capi:container` in the *description* to denote the pane inside the MDI interface in which child interfaces are added.

`document-frame-container` is a reader which returns the `document-container` of the `document-frame`.

Interfaces of any type other than subclasses of `document-frame` may be added as children. To add a child interface in your MDI interface, call `display` on the child interface and pass the MDI interface as the *screen* argument. This will display the child interface inside the container pane.

To obtain a list of the child interfaces, call the screen reader function `screen-interfaces`, passing the frame's `document-container` as the `screen` argument.

You can use most of the normal CAPI window operations such as `top-level-interface-geometry` and `activate-pane` on windows displayed as children of a `document-frame`.

The `capi:windows-menu` slot contains the Windows Menu, which allows the user to manipulate child interfaces. The standard functionality of the Windows Menu is handled by the system and normally you will not need to modify it. However, you will want to specify its position in the menu bar. Do this by adding the symbol `capi:windows-menu` in the `:menu-bar` option of your `define-interface` form.

Note: `capi:windows-menu` is a special slot in `document-frame` and this symbol should not appear elsewhere in the `define-interface` form.

By default the menu bar is made by effectively appending the menu bar of the `document-frame` interface with the menu bar of the current child. You can customize this behavior with `merge-menu-bars`.

Example

This example uses `document-frame` to create a primitive `apropos` browser.

Firstly we define an interface that lists symbols. There is nothing special about this in itself.

```
(capi:define-interface symbols-listing ()
  ((symbols :initarg :symbols))
  (:panes
   ( symbols-pane capi:list-panel
     :items symbols
     :print-function
     'symbol-name))
  (:default-initargs
   :best-width '(character 40)
   :best-height '(character 10)))
```

Next we define the MDI interface. Note:

1. It inherits from `document-frame`.
2. `capi:container` is used in the layout description.
3. `capi:windows-menu` is in the `:menu-bar` list.
4. When the interface showing the symbols is being displayed, the MDI interface is passed as the *screen* argument to `display`.

Otherwise, this example uses standard Common Lisp and CAPI functionality.

```
(capi:define-interface my-afropos-browser
  (capi:document-frame)
  ((string :initarg :string))
  (:panes
   (package-list
    capi:list-panel
    :items
    (loop for package in (list-all-packages)
          when
            (let ((al (afropos-list string package)))
              (when al
                (cons (package-name package) al)))
            collect it)
    :print-function 'car
    :action-callback
    #'(lambda (mdi-interface name-and-symbols)
        (capi:display
         (make-instance
          'symbols-listing
          :symbols (cdr name-and-symbols)
          :title (car name-and-symbols)
          :screen mdi-interface))
        :callback-type :interface-data)
    )
  (:menu-bar capi:windows-menu)
  (:layouts
   (main
    capi:row-layout
    '(package-list :divider capi:container)
    :ratios '(1 nil 4)))
  (:default-initargs
   :visible-min-height '(character 20)
   :visible-min-width '(character 100)))
```

To browse apropos of a specific string

```
(capi:display
 (make-instance 'my-apropos-browser
                :string "EDITOR"))
```

See also `current-document`
`merge-menu-bars`

double-headed-arrow-pinboard-object

Class

Summary A `pinboard-object` that draws itself as an arrow, which can switch dynamically from double-headed to single-headed.

Package `capi`

Superclasses `arrow-pinboard-object`

Initargs `:double-head-predicate`

A function determining whether a single or double arrowhead is drawn.

Description *double-head-predicate* should be a function of two arguments returning a boolean value. The first argument is the output pane on which the arrow pinboard object is drawn. The second argument is the arrow pinboard object itself.

double-head-predicate should return a true value if the arrow is to be double-headed, and `nil` if a single-headed arrow should be drawn. It is called each time the arrow object is redrawn.

Example

```
(defvar *doublep* t)

(let ((dhr
      (capi:contain
       (make-instance
        'capi:pinboard-layout
        :description
        (list
         (make-instance
          'capi:double-headed-arrow-pinboard-object
          :double-head-predicate
          #'(lambda (x y) *doublep*)
          :start-x 5 :start-y 5 :end-x 95 :end-y 95)
         (make-instance
          'capi:double-headed-arrow-pinboard-object
          :double-head-predicate
          #'(lambda (x y) *doublep*)
          :head-direction :backwards
          :start-x 5 :start-y 95 :end-x 95 :end-y 5)))
        :visible-min-width 100
        :visible-min-height 100)))
  (dotimes (x 10)
    (sleep 1)
    (setq *doublep* (not *doublep*))
    (mapcar 'capi:redraw-pinboard-object
            (capi:layout-description dhr))))
```

double-list-panel

Class

Summary A choice which displays its selected items and its unselected items in disjoint lists, and facilitates easy movement of items between these lists.

Package capi

Superclasses choice
 interface

Description The class `double-list-panel` is a choice which displays its *items* in two `list-panels`. One list contains the selected items and the other contains the unselected items. There is a pair of arrow buttons which move highlighted items between the lists.

The default *interaction* of `double-list-panel` is `:extended-selection`.

The *selection-callback*, *extend-callback* or *retract-callback* is called as appropriate when items are moved between the lists. There is no *action-callback* for `double-list-panel`.

The user selects and de-selects items in the `double-list-panel` by moving them between the two lists. There are three ways to move the items:

1. Highlight the items to move by normal `list-panel` selection gestures, then press an arrow button.
2. Highlight a single item to move by normal `list-panel` selection gestures, then press `Return`.
3. Double click on an item to move it.

Example

```
(capi:display
 (make-instance
  'capi:double-list-panel
  :items '("John" "Geoff" "chicken" "blue" "water")
  :selection-callback
  #'(lambda (item choice)
      (capi:display-message "selecting ~a" item))
  :extend-callback
  #'(lambda (item choice)
      (capi:display-message "extending ~a" item))
  :retract-callback
  #'(lambda (item choice)
      (capi:display-message "deselecting ~a" item))))
```

See also `list-panel`

drag-pane-object

Function

Summary	Initiates a dragging operation	
Package	capi	
Signature	drag-pane-object <i>pane value &key string plist image-function operations => operation</i>	
Arguments	<i>pane</i>	A pane
	<i>value</i>	An object to be dragged
	<i>string</i>	A string to be dragged or <code>nil</code>
	<i>plist</i>	A plist of formats and objects to be dragged
	<i>image-function</i>	A function or <code>nil</code>
	<i>operations</i>	A list of operation keywords allowed for the dragged objects
Values	<i>operation</i>	One of the operation keywords
Description	<p>The function <code>drag-pane-object</code> initiates a dragging operation from within the pane <i>pane</i>. It can only be called from within the button <code>:press</code> or button <code>:motion</code> callbacks of the <i>input-model</i> of an <i>output-pane</i>.</p> <p>The <i>value</i>, <i>string</i> and <i>plist</i> arguments are combined to provide an object to be dragged in various formats.</p> <p><i>value</i> can be any Lisp object (not necessarily a string) to make available for dropping into a pane within the local Lisp image.</p> <p><i>string</i> can be a string representation of <i>value</i> to make available, or <code>nil</code>. If <i>string</i> is <code>nil</code> and <i>value</i> is a string, then that will be made available as the string.</p>	

plist is a property list of additional format/value pairs to make available. The currently supported formats are as described for `set-drop-object-supported-formats`. You can make more than one format available simultaneously.

image-function provides a graphical image for use during the dragging operation on Cocoa. If *image-function* is supplied, then it should be a function of one argument. It might be called to provide an image for use during the dragging operation. The function *image-function* should return three values: a `image` object, an x offset and a y offset. The x and y offsets are the position within the image where the mouse should be located. If the image is `nil` or *image-function* is not supplied then a default image is generated. If the x or y offsets are `nil` or not returned then the image is positioned with the mouse at its center point.

Note: *image-function* is only called on Cocoa. There is no way to specify an image when dragging on Windows.

operations should be a list of operation keywords that the pane will allow the target application to perform. The operation keywords are `:copy`, `:move` and `:link` as described for the effect in `drop-object-drop-effect`. If certain platform-specific modifier keys are pressed, then some of the operations will be ignored.

The return value *operation* indicates which operation was performed by the application where the dragged object was dropped. The value will be `:none` if the object was not dropped anywhere or dragging was abandoned (for example, by the user hitting the `Escape` key). If *operation* is `:move`, then you should update the data structures in your application to remove the object that was dragged.

Note: `drag-pane-object` is not supported on X11/Motif. See `simple-pane` for information about drop callbacks.

Example

See `examples/capi/output-panes/drag-and-drop.lisp`

See also `simple-pane`

draw-metafile

Function

Summary Draws a metafile to a pane.

Package `capi`

Signature `draw-metafile pane metafile x y width height`

Arguments *pane* An output-pane.
metafile A metafile, as described in `with-internal-metafile`.
x,y Integers.
width,height Non-negative integers.

Description The function `draw-metafile` draws the metafile *metafile* to the pane *pane* at position *x,y* with size *width, height*.
metafile should be a metafile as returned by `with-internal-metafile`.
`draw-metafile` is not implemented on X11/Motif.

See also `clipboard`
`draw-metafile-to-image`
`free-metafile`
`with-internal-metafile`

draw-metafile-to-image

Function

Summary Draws a metafile as an image.

Package `capi`

Signature	<code>draw-metafile-to-image pane metafile &key width height max-width max-height background alpha => image</code>	
Arguments	<code>pane</code>	An output-pane.
	<code>metafile</code>	A metafile.
	<code>width,height</code>	Non-negative integers, or <code>nil</code> .
	<code>max-width,max-height</code>	Non-negative integers, or <code>nil</code> .
	<code>background</code>	A color specification.
	<code>alpha</code>	A generalized boolean.
Values	<code>image</code>	An image.
Description	<p>The function <code>draw-metafile-to-image</code> returns a new <code>image</code> object for <code>pane</code>, with <code>metafile</code> drawn into the image.</p> <p><code>metafile</code> should be a metafile as returned by <code>with-internal-metafile</code>.</p> <p>If <code>width</code> and <code>height</code> are both <code>nil</code> then the size of the image is computed from the metafile. If both <code>width</code> and <code>height</code> are integers, then they specify the size of the image and the metafile is scaled to fit. If one of <code>width</code> or <code>height</code> is <code>nil</code>, then it is computed from the other dimension, preserving the aspect ratio of the metafile. The default values of <code>width</code> and <code>height</code> are both <code>nil</code>.</p> <p>The <code>max-width</code> and <code>max-height</code> arguments, if non-<code>nil</code>, constrain the computed or specified values of <code>width</code> and <code>height</code> respectively. The aspect ratio is retained when the size is constrained, so specifying a <code>max-width</code> can also reduce the actual height of the image. The default values of <code>max-width</code> and <code>max-height</code> are both <code>nil</code>.</p>	

background should be a color spec, which controls the non-drawn parts of the image. (A color spec can be obtained by `get-color-spec`, `make-rgb` and so on.) If *background* is omitted, then the background color of *pane* is used.

If *alpha* is non-`nil`, then the image will have an alpha component. The default value of *alpha* is `nil`.

`draw-metafile-to-image` is not implemented on X11/Motif.

See also `clipboard`
`draw-metafile`
`free-metafile`
`with-internal-metafile`

drawn-pinboard-object

Class

Summary The class `drawn-pinboard-object` is a subclass of `pinboard-object` which is drawn by a supplied function, and is provided as a means of the user creating their own pinboard objects.

Package `capi`

Superclasses `pinboard-object`

Initargs `:display-callback`
Called to display the object.

Accessors `drawn-pinboard-object-display-callback`

Description The *display-callback* is called with the output pane to draw on, the `drawn-pinboard-object` itself, and the *x*, *y*, *width* and *height* of the object, and it is expected to redraw that section.

An alternative way of doing this is to create a subclass of `pinboard-object` and to provide a method for `draw-pinboard-object`.

Example

```
(defun draw-an-ellipse
  (output-pane self x y width height)
  (let ((x-radius (floor width 2))
        (y-radius (floor height 2)))
    (gp:draw-ellipse output-pane
      (+ x x-radius) (+ y y-radius)
      x-radius y-radius
      :foreground :red
      :filled t)))

(capi:contain (make-instance
  'capi:drawn-pinboard-object
  :visible-min-width 200
  :visible-min-height 100
  :display-callback 'draw-an-ellipse))
```

See also `pinboard-layout`

draw-pinboard-object

Generic Function

Package `capi`

Signature `draw-pinboard-object` *pinboard object* &key *x y*
width height &allow-other-keys

Description This generic function is called whenever a pinboard object needs to be drawn. The *x*, *y*, *width* and *height* arguments indicate the region that needs to be redrawn, but a method is free to ignore these and just draw the complete object.

Example See the example in the file `examples/capi/graphics/circled-graph-nodes.lisp`

See also `pinboard-layout`
`pinboard-object`

draw-pinboard-object-highlighted

Generic Function

Summary	The generic function <code>draw-pinboard-object-highlighted</code> draws highlighting on a pre-drawn pinboard object.
Package	<code>capi</code>
Signature	<code>draw-pinboard-object-highlighted</code> <i>pinboard object</i> <code>&key &allow-other-keys</code>
Description	This generic function draws the highlighting onto a pinboard object that has already been drawn. The default highlighting method draws a box around the object, and should be sufficient for most purposes.
Example	See the example in the file <code>examples/capi/graphics/circled-graph-nodes.lisp</code>
See also	<code>draw-pinboard-object-unhighlighted</code> <code>highlight-pinboard-object</code>

draw-pinboard-object-unhighlighted

Generic Function

Summary	The generic function <code>draw-pinboard-object-unhighlighted</code> removes the highlighting from a pinboard object.
Package	<code>capi</code>
Signature	<code>draw-pinboard-object-unhighlighted</code> <i>pinboard object</i> <code>&key &allow-other-keys</code>
Description	This generic function removes the highlighting from a pinboard object.
Example	See the example in the file <code>examples/capi/graphics/circled-graph-nodes.lisp</code>

See also `draw-pinboard-object-highlighted`
`highlight-pinboard-object`

drop-object-allows-drop-effect-p *Function*

Summary Queries whether a dropping operation can be performed with a given effect.

Package `capi`

Signature `drop-object-allows-drop-effect-p` *drop-object effect => result*

Arguments *drop-object* A *drop-object*, as passed to the *drop-callback*.
effect An effect keyword

Values *result* A boolean

Description The function `drop-object-allows-drop-effect-p` returns `non-nil` if the dropping operation can be performed with the given effect *effect*. It returns `nil` if the dropping operation cannot be performed. See `drop-object-drop-effect` for information on drop effect keywords.

Note: `drop-object-allows-drop-effect-p` should only be called within a *drop-callback*. It is not supported on X11/Motif. See `simple-pane` for information about drop callbacks.

See also `drop-object-drop-effect`
`simple-pane`

drop-object-drop-effect *Function*

Summary Reads or sets the current effect of a dropping operation.

Package	<code>capi</code>
Signature	<code>drop-object-drop-effect</code> <i>drop-object</i> => <i>effect</i>
Signature	<code>(setf drop-object-drop-effect)</code> <i>effect</i> <i>drop-object</i> => <i>effect</i>
Arguments	<i>drop-object</i> A <i>drop-object</i> , as passed to the <i>drop-callback</i> .
Values	<i>effect</i> An effect keyword
Description	<p>The function <code>drop-object-drop-effect</code> gets or sets the current effect of the dropping operation. <i>effect</i> can be one of:</p> <ul style="list-style-type: none"> <code>:copy</code> The object will be copied. This is the most common value for operations between applications. <code>:move</code> The object will be moved. This is usually triggered by the user dragging with a platform-specific modifier key pressed. <code>:link</code> A link to the object will be created. This is usually triggered by the user dragging with a platform-specific modifier key pressed. <code>:none</code> No dragging is possible. <p>Note: <code>drop-object-drop-effect</code> should only be called within a <i>drop-callback</i>. It is not supported on X11/Motif. See <code>simple-pane</code> for information about drop callbacks.</p>
Example	See <code>examples/capi/output-panes/drag-and-drop.lisp</code>
See also	<code>simple-pane</code>

drop-object-get-object

Function

Summary Returns a dropped object in a given format

Package	<code>capi</code>	
Signature	<code>drop-object-get-object</code>	<i>drop-object format => object</i>
Arguments	<i>drop-object</i>	A <i>drop-object</i> , as passed to the <i>drop-callback</i> .
	<i>format</i>	A format keyword
Values	<i>object</i>	An object in the given format
Description	<p>The function <code>drop-object-get-object</code> returns the dropped object in the given format. See <code>set-drop-object-supported-formats</code> for information on format keywords.</p> <p>Note: <code>drop-object-get-object</code> should only be called within a <i>drop-callback</i>. It is not supported on X11/Motif. See <code>simple-pane</code> for information about drop callbacks.</p>	
Example	See <code>examples/capi/output-panes/drag-and-drop.lisp</code>	
See also	<code>set-drop-object-supported-formats</code> <code>simple-pane</code>	

drop-object-pane-x

Function

Summary	Gets the x coordinate in the pane that the object is being dropped over.	
Package	<code>capi</code>	
Signature	<code>drop-object-pane-x</code>	<i>drop-object => coord</i>
Arguments	<i>drop-object</i>	A <i>drop-object</i> , as passed to the <i>drop-callback</i> .
Values	<i>coord</i>	An integer

Description The function `drop-object-pane-x` returns the x coordinate within the pane that the object is being dropped over. This information is only meaningful when the pane is an instance of `output-pane` or one of its subclasses.

Note: `drop-object-pane-x` should only be called within a *drop-callback*. It is not supported on X11/Motif. See `simple-pane` for information about drop callbacks.

See also `drop-object-pane-y`
`simple-pane`

drop-object-pane-y

Function

Summary Gets the y coordinate in the pane that the object is being dropped over.

Package `capi`

Signature `drop-object-pane-y drop-object => coord`

Arguments *drop-object* A *drop-object*, as passed to the *drop-callback*.

Values *coord* An integer

Description The function `drop-object-pane-y` returns the y coordinate within the pane that the object is being dropped over. This information is only meaningful when the pane is an instance of `output-pane` or one of its subclasses.

Note: `drop-object-pane-y` should only be called within a *drop-callback*. It is not supported on X11/Motif. See `simple-pane` for information about drop callbacks.

See also `drop-object-pane-x`
`simple-pane`

drop-object-provides-format*Function*

Summary	Queries whether a dropping operation can provide an object in a given format.	
Package	<code>capi</code>	
Signature	<code>drop-object-provides-format</code> <i>drop-object</i> <i>format</i> => <i>result</i>	
Arguments	<i>drop-object</i>	A <i>drop-object</i> , as passed to the <i>drop-callback</i> .
	<i>format</i>	A format keyword
Values	<i>result</i>	A boolean
Description	<p>The function <code>drop-object-provides-format</code> returns <code>non-nil</code> if the dropping operation can provide an object in the given format. It returns <code>nil</code> if it cannot provide that format.</p> <p>See <code>set-drop-object-supported-formats</code> for information on format keywords.</p> <p>Note: <code>drop-object-provides-format</code> should only be called within a <i>drop-callback</i>. It is not supported on X11/Motif. See <code>simple-pane</code> for information about drop callbacks.</p>	
Example	See <code>examples/capi/output-panes/drag-and-drop.lisp</code>	
See also	<code>set-drop-object-supported-formats</code> <code>simple-pane</code>	

echo-area-cursor-inactive-style*Variable*

Summary	The drawing style of the Echo Area cursor when the window is inactive.	
Package	<code>capi</code>	

Initial Value	<code>:invisible</code>
Description	The drawing style of the cursor in the Echo Area of an inactive Common LispWorks window. The allowed values are <code>:inverse</code> , <code>:outline</code> , <code>:underline</code> and <code>:invisible</code> .

echo-area-pane

Class

Summary	The class of the Editor's echo area.
Package	<code>capi</code>
Superclasses	<code>editor-pane</code>
Description	The class <code>echo-area-pane</code> is used to implement the small window for user interaction, known as the Echo Area, which is at the bottom of Editor windows in the Common LispWorks development environment. You should not normally need to work with this class directly. To add an Echo Area, pass <code>:echo-area t</code> when making the <code>editor-pane</code> .

editor-cursor-color

Variable

Summary	The background color of the cursor.
Package	<code>capi</code>
Initial Value	<code>nil</code>
Description	When non- <code>nil</code> , the value is a color spec or color alias determining the background color of the <code>editor-pane</code> cursor. See the <i>LispWorks User Guide</i> for details of the color package.

The value `nil` means that the cursor background color is the same as the foreground color of the editor pane.

Example `(setf capi:*editor-cursor-color* :red)`

editor-cursor-active-style

Variable

Summary The drawing style of the editor's cursor when the window is active.

Package `capi`

Initial Value `:inverse`

Description The drawing style of an `editor-pane` cursor when the window is active.

The allowed values are `:inverse`, `:outline`, `:underline`, `:left-bar` and `:caret`.

See also `editor-pane-blink-rate`

editor-cursor-drag-style

Variable

Summary The drawing style of the editor's cursor during a selection drag.

Package `capi`

Initial Value `:left-bar`

Description The drawing style of an `editor-pane` cursor during a selection drag.

The allowed values are `:inverse`, `:outline`, `:underline`, `:left-bar` and `:caret`.

editor-cursor-inactive-style

Variable

Summary	The drawing style of the editor's cursor when the window is inactive.
Package	<code>capi</code>
Initial value	<code>:outline</code>
Description	The drawing style of an <code>editor-pane</code> cursor when the window is inactive. The allowed values are <code>:inverse</code> , <code>:outline</code> , <code>:underline</code> or <code>:invisible</code> .

editor-pane

Class

Summary	An editor pane is an editor that has all of the functionality described in the <i>LispWorks Guide To The Editor</i> .	
Package	<code>capi</code>	
Superclasses	<code>output-pane</code>	
Subclasses	<code>interactive-pane</code> <code>collector-pane</code>	
Initargs	<code>:text</code>	The text in the editor pane.
	<code>:enabled</code>	If <code>t</code> the editor pane will accept input from the mouse and keyboard.
	<code>:buffer-modes</code>	A list specifying the modes of the editor buffer.
	<code>:buffer-name</code>	The name of the editor buffer.
	<code>:change-callback</code>	A function designator, or <code>nil</code> .

<code>:before-input-callback</code>	A function designator, or <code>nil</code> .
<code>:after-input-callback</code>	A function designator, or <code>nil</code> .
<code>:echo-area</code>	A flag determining whether the editor pane has an Echo Area.
<code>:fixed-fill</code>	An integer specifying the fill length, or <code>nil</code> .
<code>:line-wrap-marker</code>	A character, or <code>nil</code> .
<code>:line-wrap-face</code>	An <code>editor:face</code> object, or a symbol naming a face, or <code>nil</code> .
<code>:wrap-style</code>	An integer specifying the fill length, or <code>nil</code> .

Accessors

`editor-pane-text`
`editor-pane-change-callback`
`editor-pane-enabled`
`editor-pane-fixed-fill`
`editor-pane-line-wrap-marker`
`editor-pane-line-wrap-face`
`editor-pane-wrap-style`

Description

The accessor `editor-pane-text` is provided to read and write the text in the editor buffer. The accessor `editor-pane-enabled` is used to enable and disable the editor (when it is disabled, it ignores all input from the mouse and keyboard).

The `editor-pane` stores text in buffers which are uniquely named, and so to create an `editor-pane` using an existing buffer you should pass the *buffer-name*. To create an `editor-pane` with a new buffer, pass a *buffer-name* that does not match any existing buffer. If *buffer-name* is not passed, then the `editor-pane` uses some existing buffer.

buffer-modes allows you to specify the initial major mode and minor modes of the `editor-pane`'s buffer. It should be a list of the form (*major-mode-name* . *minor-mode-names*). See the *LispWorks Editor User Guide* for a description of major and minor modes in the LispWorks editor.

Note: *buffer-modes* is used only when the CAPI creates the buffer, and not when it reuses a buffer.

If *echo-area* is non-`nil`, then an Echo Area is added. *echo-area* defaults to `nil`.

If *fixed-fill* is non-`nil`, the editor pane tries to form lines of length close to, but no more than, *fixed-fill*. It does this by forcing line breaks at spaces between words. *fixed-fill* defaults to `nil`.

The cursor in an `editor-pane` blinks on and off by the mechanism described in `editor-pane-blink-rate`.

change-callback, if non-`nil`, should be a function which is called whenever the editor buffer under the `editor-pane` changes. The value *change-callback* can be set either by:

```
(make-instance 'capi:editor-pane :change-callback ...)
```

or

```
(setf capi:editor-pane-change-callback)
```

The current value can be queried by the accessor `editor-pane-change-callback`.

The *change-callback* function must have signature:

```
change-callback pane point old-length new-length
```

pane is the `editor-pane` itself.

point is an `editor:point` object where the modification to the underlying buffer starts. *point* is a temporary point, and is not valid outside the scope of the change callback. For more information about `editor:point` objects, see "Points" in the *LispWorks Editor User Guide*.

old-length is the length of the affected text following *point*, prior to the modification.

new-length is the length of the affected text following *point*, after the modification has occurred.

Typical calls to the *change-callback* occur on insertion of text (when *old-length* is 0) and on deletion of text (when *new-length* is 0). There can be other combinations, for example, after executing the `UPPERCASE REGION` editor command, *change-callback* be called with both *old-length* and *new-length* being the length of the region. The same is true for changing editor text properties.

The *change-callback* is always executed in the process of *pane* (as if by `apply-in-pane-process`).

The *change-callback* is permitted to modify the buffer of *pane*, and other editor buffers. The callback is disabled inside the dynamic scope of the call, so there are no recursive calls to the *change-callback* of *pane*. However, changes done by the callback may trigger *change-callback* calls on other `editor-panes`, whether in the same process or in another process.

There is an example illustrating the use of *change-callback* in the file `examples/capi/editor/change-callback.lisp`.

You can use the initargs `:before-input-callback` and `:after-input-callback` to add input callbacks which are called when `call-editor` is called. Note that the default *input-model* also generates calls to `call-editor`, so unless you override the default *input-model* the input callbacks are called for all keyboard and mouse gestures (other than gestures that are processed by a non-focus completer window).

In both cases (*before* and *after*) the argument is a function that takes two arguments: the editor pane itself and the input gesture (the second argument to `call-editor`).

`call-editor` may redirect gestures to another pane. For example, gestures to an `editor-pane` are redirected to the echo area while it is used. In this case the *before* callback is

called more than once for the same gesture. The *after* callback is called only once for each gesture, on the pane that actually processed the gesture.

line-wrap-marker specifies the marker to display at the end of a line that is wrapped to the next line, or truncated if *wrap-style* is `nil`. The value must be a `character`, or `nil` (which is interpreted as `#\space`). The default value is the value of `*default-editor-pane-line-wrap-marker*`. The value can be read by `editor-pane-line-wrap-marker`.

line-wrap-face specifies a face to use when displaying the *line-wrap-marker*. The argument can be `nil`, an `editor:face` object (the result of a call to `editor:make-face`), or a symbol naming a face (that is, the first argument to `editor:make-face`).

The default value of *line-wrap-face* is an internal symbol naming a face. The value can be accessed by `editor-pane-line-wrap-face`. The default face can be modified in the IDE via the Editor tool's **Preferences...** dialog (**Styles** tab, style name **Line Wrap Marker**).

wrap-style defines the wrapping of text lines that cannot be displayed in one line of the `editor-pane`. The argument can be one of:

- `t` Normal wrapping. Display as many characters as possible in the `editor-pane` line.
- `nil` Do not wrap. Text lines that are too long are truncated.
- `:split-on-space` Wrapping, but attempts to split lines on spaces. When the text reaches the end of a line, the code looks backwards for space, and wraps before it.

The default value of *wrap-style* is `t` and the value can be accessed by `editor-pane-wrap-style`.

Note: editor panes support GNU Emacs keys on all platforms. Additionally on Microsoft Windows they support Windows editor keys, on Motif they support KDE/Gnome keys, and on Cocoa they support Mac OS X editor keys. Exactly one style of emulation is active at any one time for each editor pane. By default, editor panes in the Common LispWorks development environment use Emacs emulation on all platforms. By default, editor panes in delivered applications use Windows emulation on Microsoft Windows, Mac OS X editor emulation on Cocoa, and Emacs emulation on Motif. To alter the choice of emulation, see `interface-keys-style` or the `deliver` keyword `:editor-style`, described in the *LispWorks Delivery User Guide*.

Compatibility Note In LispWorks 4.4 and previous versions, `editor-pane` supports only fixed-width fonts. In LispWorks 5.0 and later, variable-width fonts can also be used on Microsoft Windows and Motif. Specify the font via the `:font` initarg (see `simple-pane`).

Compatibility Note The `:wrap-style` initarg supersedes `editor:set-window-split-on-space`, which is deprecated.

Example

```
(capi:contain (make-instance 'capi:editor-pane
                           :text "Hello world"))

(setq ed (capi:contain
         (make-instance 'capi:editor-pane
                       :text "Hello world"
                       :enabled nil)))
```

Note that you cannot type into the editor pane.

```
(capi:apply-in-pane-process
 ed #'(setf capi:editor-pane-enabled) t ed)
```

Now you can enter text into the editor pane interactively.

You can also change the text programmatically:

```
(capi:apply-in-pane-process
  ed #'(setf capi:editor-pane-text) "New text" ed)
```

In this example the callback modifies the buffer in the correct editor context so you that see the editor update immediately:

```
(capi:define-interface updating-editor ()
  ()
  (:panes
    (numbers capi:list-panel
      :items '(1 2 3)
      :selection-callback 'update-editor
      :callback-type :interface
      :visible-min-height '(:character 3))
    (editor capi:editor-pane
      :text
      "Select numbers in the list above."
      :visible-min-width
      (list :character 35))))

(defun update-editor (interface)
  (with-slots (numbers editor) interface
    (editor:process-character
      (list #'(setf capi:editor-pane-text)
            (format nil "~R"
                    (capi:choice-selected-item numbers))
            editor)
      (capi:editor-window editor))))

(capi:display (make-instance 'updating-editor))
```

This example illustrates the use of *buffer-modes* to specify a major mode:

```

(defclass my-lisp-editor (capi:editor-pane) ()
  (:default-initargs
   :buffer-modes '("Lisp")
   :echo-area t
   :text
   ";; Lisp mode functionality such as command bindings
and
;; parenthesis balancing work in this window.

(list 1 2 3)
"
   :visible-min-width '(:character 60)
   :name "My Lisp Editor Pane"))

(capi:define-interface my-lisp-editor-interface ()
  ()
  (:panes
   (ed
    my-lisp-editor
   ))
  (:default-initargs
   :title "My Lisp Editor Interface"))

;; Ensure Emacs-like bindings regardless of platform
(defmethod capi:interface-keys-style
  ((self my-lisp-editor-interface))
  :emacs)

(capi:display
 (make-instance 'my-lisp-editor-interface))

```

Also see the examples in the directory
[examples/capi/editor/](#).

See also

```

call-editor
*default-editor-pane-line-wrap-marker*
editor-pane-blink-rate
*editor-cursor-active-style*
*editor-cursor-color*
*editor-cursor-drag-style*
*editor-cursor-inactive-style*
interface-keys-style
modify-editor-pane-buffer

```

editor-pane-blink-rate

Generic Function

Summary	Returns the cursor blinking rate for an editor pane.
Package	<code>capi</code>
Signature	<code>editor-pane-blink-rate self => blink-rate</code>
Arguments	<i>self</i> An editor pane.
Values	<i>blink-rate</i> A non-negative real number, or <code>nil</code> .
Description	<p>The system calls the function <code>editor-pane-blink-rate</code> to determine the cursor blinking rate in milliseconds. The pane uses the value <i>blink-rate</i> each time it gets the focus.</p> <p>If <i>blink-rate</i> is a positive real number, then it is the blinking rate in milliseconds. If <i>blink-rate</i> is 0, then there is no blinking. If <i>blink-rate</i> is <code>nil</code>, then the default blinking rate is used.</p> <p>The default method on <code>editor-pane-blink-rate</code> returns <i>nil</i>, which means use the default blinking rate. <code>set-default-editor-pane-blink-rate</code>.</p> <p>You can define your own methods on <code>editor-pane-blink-rate</code> for <code>editor-pane</code> and subclasses thereof.</p>
See also	<code>*editor-cursor-active-style*</code> <code>editor-pane</code> <code>editor-pane-native-blink-rate</code> <code>set-default-editor-pane-blink-rate</code>

editor-pane-buffer

Function

Summary	Returns the editor buffer associated with an editor pane.
Package	<code>capi</code>

Signature	<code>editor-pane-buffer</code> <i>pane</i>
Description	The function <code>editor-pane-buffer</code> returns the editor buffer associated with an editor pane, which can be manipulated in the standard ways with the routines in the editor package.
Example	<pre>(setq editor-pane (capi:contain (make-instance 'capi:editor-pane :text "Hello world"))) (setq buffer (capi:editor-pane-buffer editor-pane)) (editor:insert-string (editor:buffers-end buffer) (format nil "~%Here's some more text..."))</pre>
See also	<code>editor-pane</code>

editor-pane-native-blink-rate

Function

Summary	Returns the native cursor blinking rate for an <code>editor-pane</code> .
Package	<code>capi</code>
Signature	<code>editor-pane-native-blink-rate</code> <i>pane</i> => <i>blink-rate</i>
Arguments	<i>pane</i> An <code>editor-pane</code> .
Values	<i>blink-rate</i> A non-negative real number, or <code>nil</code> .
Description	<p>The function <code>editor-pane-native-blink-rate</code> returns the native cursor blinking rate for the <code>editor-pane</code> <i>pane</i>, that is the rate that the GUI library (Motif, Microsoft Windows, Cocoa) uses.</p> <p>The value <i>blink-rate</i> is interpreted as a blinking rate as described in <code>editor-pane-blink-rate</code>.</p>

See also `editor-pane-blink-rate`
`set-default-editor-pane-blink-rate`

editor-pane-selected-text

Function

Summary Returns the selected text in an `editor-pane`.

Package `capi`

Signature `editor-pane-selected-text editor-pane => result`

Arguments *editor-pane* An `editor-pane`.

Values *result* A string or `nil`.

Description The function `editor-pane-selected-text` takes an instance of `editor-pane` as its argument and returns the selected text in *editor-pane*, or `nil` if there is no selection.

See also `editor-pane`
`editor-pane-selected-text-p`

editor-pane-selected-text-p

Function

Summary The predicate for a current selection in an `editor-pane`.

Package `capi`

Signature `editor-pane-selected-text-p editor-pane => result`

Arguments *editor-pane* An `editor-pane`.

Values *result* A boolean.

Description The function `editor-pane-selected-text-p` takes an instance of `editor-pane` as its argument and returns `t` if there is text currently selected in `editor-pane`, or `nil` if there is no selection.

See also `editor-pane`
`editor-pane-selected-text`

editor-pane-stream

Function

Summary Returns the output stream associated with an editor pane.

Package `capi`

Signature `editor-pane-stream editor-pane => stream`

Arguments `editor-pane` An `editor-pane`.

Values `stream` An output stream.

Description The function `editor-pane-stream` returns the stream where the results of evaluation in the editor buffer currently associated with `pane` are printed to.

See also `editor-pane`

editor-window

Generic Function

Summary Returns the editor window object.

Package `capi`

Signature `editor-window editor => editor-window`

Arguments	<code>editor</code>	An <code>editor-pane</code> or an Editor interface in the Common LispWorks IDE.
Values	<code>editor-window</code>	An editor window object.
Description	<p>The generic function <code>editor-window</code> returns the editor window object associated with <code>editor</code>.</p> <p>The functionality of editor windows is documented in the <i>LispWorks Editor User Guide</i>.</p>	
See also	<code>editor-pane</code>	

element

Class

Summary	The class <code>element</code> is the superclass of all CAPI objects that appear in a window.	
Package	<code>capi</code>	
Superclasses	<code>capi-object</code>	
Subclasses	<code>simple-pane</code> <code>menu</code>	
Initargs	<code>:parent</code>	The element containing this element.
	<code>:interface</code>	The interface containing this element.
	<code>:accepts-focus-p</code>	Specifies that the element should accept input.
	<code>:help-key</code>	An object used for lookup of help. Default value <code>t</code> .

The following initargs are geometry hints, influencing the initial size and position of an element and constraining its size:

<code>:x</code>	The x position of the element in a pinboard.
<code>:y</code>	The y position of the element in a pinboard.
<code>:external-min-width</code>	The minimum width of the element in its parent.
<code>:external-min-height</code>	The minimum height of the element in its parent.
<code>:external-max-width</code>	The maximum width of the element in its parent.
<code>:external-max-height</code>	The maximum height of the element in its parent.
<code>:visible-min-width</code>	The minimum visible width of the element.
<code>:visible-min-height</code>	The minimum visible height of the element.
<code>:visible-max-width</code>	The maximum visible width of the element.
<code>:visible-max-height</code>	The maximum height of the element.
<code>:internal-min-width</code>	The minimum width of the display region.
<code>:internal-min-height</code>	The minimum height of the display region.
<code>:internal-max-width</code>	The maximum width of the display region.
<code>:internal-max-height</code>	The maximum height of the display region.

Accessors	<code>element-parent</code>
Readers	<code>element-interface</code> <code>help-key</code>
Description	<p>The class <code>element</code> contains the slots <i>parent</i> and <i>interface</i> which contain the element and the interface that the element is contained in respectively. The writer method <code>element-parent</code> can be used to re-parent an element into another parent (or to remove it from a container entirely by setting its parent to <code>nil</code>). Note that an element should not be used in more than one place at a time.</p> <p>The initarg <i>accepts-focus-p</i> specifies that the element can accept input. The default value is <code>t</code>. In some subclasses including <code>display-pane</code> and <code>title-pane</code> the default value of <i>accepts-focus-p</i> is <code>nil</code>. A pane accepts the input focus if and only if the function <code>accepts-focus-p</code> returns true.</p> <p><i>accepts-focus-p</i> also influences whether a pane is a tabstop on Microsoft Windows, where a pane acts as a tabstop if and only if the function <code>accepts-focus-p</code> returns true and the <code>:accepts-focus-p</code> initarg value is <code>:force</code>. On Motif and Cocoa, a pane acts as a tabstop if and only if the function <code>accepts-focus-p</code> returns true.</p> <p><i>help-key</i> is used to determine how help is displayed for the pane. The value <code>nil</code> means that no help is displayed. Otherwise, <i>help-key</i> is passed to the <i>interface-help-callback</i>, except when <i>help-key</i> is <code>t</code>, when the name of the pane is passed to the <i>interface-help-callback</i>. For details of <i>interface-help-callback</i>, see <code>interface</code>.</p> <p>All elements accept initargs (listed above) representing hints as to the initial size and position of the element. By default elements have a minimum pixel size of one by one, and a maximum size of <code>nil</code> (meaning no maximum), but the hints can be specified to change these values. The possible values for these hints are as follows:</p>

<i>integer</i>	The size in pixels.
<code>t</code>	For <code>:visible-max-width</code> , <code>t</code> means use the value of <code>:visible-min-width</code> . For <code>:visible-max-height</code> , <code>t</code> means use the value of <code>:visible-min-height</code> .
<code>:text-width</code>	The width of any text in the element.
<code>:text-height</code>	The height of any text in the element.
<code>:screen-width</code>	The width of the screen.
<code>:screen-height</code>	The height of the screen.

Also, hints can be a list starting with any of the following operators, followed by one or more hints.

<code>max</code>	The maximum size of the hints.
<code>min</code>	The minimum size of the hints.
<code>+</code>	The sum of the hints.
<code>-</code>	The subtraction of hints from the first.
<code>*</code>	The multiplication of the hints.
<code>/</code>	The division of hints from the first.

Also, a hint can be a two element list specifying the size of a certain amount of text when drawn in the element:

`(:character integer)`

`(character integer)`

The size of *integer* characters.

`(:string string)`

`(string string)`

The size of *string*.

A hint can be a two-element list interpreted as the value of a symbol:

```
(symbol-value foo)
```

The size of the `symbol-value` of `foo`.

Finally, you can choose to `apply` or `funcall` an arbitrary function, by passing a list starting with `funcall` or `apply`, followed by the function and then the arguments.

The hints of an element can be changed dynamically using `set-hint-table`: such a call might change the geometry.

Note: If the `visible-max-width` is the same as the `visible-min-width`, then the element is not horizontally resizable. If the `visible-max-height` is the same as the `visible-min-height`, then the element is not vertically resizable.

Note: Some classes have default `initargs` providing useful hints. For example, `display-pane` has `:text-height` as the default value of `:visible-min-height`, ensuring that the text is visible.

Note: The `ratios`, `x-ratios` and `y-ratios` settings in some layouts (for example `grid-layout`) also control the actual size of the pane when the constraints are not specified. In particular, if `nil` is used in the ratios then the associated pane(s) will be fixed at their minimum size.

Compatibility Note

The `:min-width`, `:max-width`, `:min-height`, and `:max-height` `initargs` are still accepted for compatibility with LispWorks 3.2, but their use is discouraged.

In LispWorks 4, `:visible-min-width` means the same as `:min-width`, but takes precedence if both are specified. The use of `:min-width` can lead to confusion because some CAPI classes have default values for `:visible-min-width` which will override `:min-width`. Similarly for `:min-height`, `:max-width`, and `:max-height`. Therefore, your code should use `:visible-min-width` and friends.

Examples

```
(capi:display (make-instance 'capi:interface
                             :title "Test"
                             :visible-min-width 300))
```

```
(capi:display (make-instance 'capi:interface
                            :title "Test"
                            :visible-min-width 300
                            :visible-max-height 200))
```

Here is a simple example that demonstrates the use of the `element-parent` accessor to place elements.

```
(setq pinboard (capi:contain
                (make-instance
                 'capi:pinboard-layout)
                :visible-min-width 520
                :visible-min-height 395))

(setq object
  (make-instance
   'capi:image-pinboard-object
   :x 10 :y 10
   :image
   (sys:lispworks-file
    "examples/capi/graphics/lwsplash.bmp")
   :parent pinboard))

(capi:apply-in-pane-process
 pinboard #'(setf capi:element-parent) nil object)

(capi:apply-in-pane-process
 pinboard #'(setf capi:element-parent) pinboard object)
```

See also `set-hint-table`

element-container

Function

Summary	Returns the container of an element.	
Package	<code>capi</code>	
Signature	<code>element-container</code> <i>element</i> => <i>container</i>	
Arguments	<i>element</i>	An element.
Values	<i>container</i>	A screen or a document-frame.

Description	<p>The function <code>element-container</code> returns the container of the element <i>element</i>.</p> <p>If <i>element</i> is inside a standalone interface, then <i>container</i> is the <code>screen</code> object.</p> <p>If <i>element</i> is inside an interface that is inside a MDI interface, then <i>container</i> is the <code>capi:container</code> object of that MDI interface. See <code>document-frame</code> for details.</p>
See also	<p><code>document-frame</code></p> <p><code>element</code></p>

element-interface-for-callback

Function

Summary	Returns the interface that is used in an element's callbacks.
Package	<code>capi</code>
Signature	<code>element-interface-for-callback element => interface</code>
Description	<p>The function <code>element-interface-for-callback</code> returns the interface that is passed to callbacks in <i>element</i>. Normally this is the interface that <i>element</i> is in, but that can be changed by <code>attach-interface-for-callback</code>.</p>
See also	<p><code>attach-interface-for-callback</code></p> <p><code>element</code></p>

element-screen

Function

Summary	Returns the screen that an element is associated with.
Package	<code>capi</code>
Signature	<code>element-screen element => screen</code>

Description The function `element-screen` returns the screen that the element *element* is associated with.

See also `element`

ellipse

Class

Summary A pinboard object that draws itself as an ellipse.

Package `capi`

Superclasses `pinboard-object`

Accessors `filled`

Initargs `:filled` A boolean.

Description The class `ellipse` is a `pinboard-object` that draws itself as an ellipse.

If *filled* is true, then the ellipse is filled with the foreground color. *filled* defaults to `nil`.

ensure-area-visible

Generic Function

Summary Ensures an area is visible in a scrollable pane.

Package `capi`

Signature `ensure-area-visible self x y width height`

Arguments *self* A `simple-pane` with internal scrolling.

x,y The coordinates of the origin of the area to make visible.

width, height The dimensions of the area to make visible

Description The generic function `ensure-area-visible` ensures that the area specified by `x`, `y`, `width` and `height`, or at least part of it, is visible.

This function works only for subclasses of `simple-pane` that do internal scrolling (such as `editor-pane`). An error is signalled if it is called with other classes.

ensure-interface-screen

Function

Summary The `ensure-interface-screen` function ensures that a top level interface is displayed on a given screen.

Package `capi`

Signature `ensure-interface-screen self &key screen`

Description This ensures that the top level interface is displayed on the given `screen` (or the default) if `display` is called later without a `screen` argument. This allows the querying of font and color information associated with a particular screen. It returns the screen that is used.

See also `screen`
 `display`
 `interface`

execute-with-interface

Function

Summary Allows functions to be executed in the event process of a given interface.

Package `capi`

Signature `execute-with-interface interface function &rest args`

Arguments	<i>interface</i>	An <i>interface</i>
	<i>function</i>	A function designator
	<i>args</i>	Arguments passed to <i>function</i>
Description	<p>The function <code>execute-with-interface</code> is a useful way of operating on an <i>interface</i> owned by another process. It takes a top-level interface, a function and some arguments and queues the function to be run by that process when it next enters its event loop (for an interface owned by the current process, it calls the function immediately).</p> <p>Note: <code>execute-with-interface</code> applies <i>function</i> even if <i>interface</i> does not have a screen representation, for example when it is destroyed. To call <i>function</i> only if <i>interface</i> has a representation, use <code>execute-with-interface-if-alive</code>.</p> <p>Note: All accesses (reads as well as writes) on a CAPI interface and its sub-elements should be performed in the interface process. Within a callback on the interface this happens automatically, but <code>execute-with-interface</code> is a useful utility in other circumstances.</p>	
Example	<pre>(setq a (capi:display (make-instance 'capi:interface))) (capi:execute-with-interface a 'break "Break inside the interface process")</pre>	
See also	<pre>apply-in-pane-process execute-with-interface-if-alive</pre>	

execute-with-interface-if-alive*Function*

Summary	Executes a function in the event process of a given interface if it is alive.
Package	<code>capi</code>

Signature	<code>execute-with-interface-if-alive</code> <i>interface</i> <i>function</i> &rest <i>args</i> => nil
Description	<p>The function <code>execute-with-interface-if-alive</code> applies the function <i>function</i> to the arguments <i>args</i> in the process of the interface <i>interface</i>, if the interface is "alive". An interface is alive if it has a representation on the screen.</p> <p>If <i>interface</i> is not alive, <i>function</i> is not applied. This is in contrast to <code>execute-with-interface</code>, which in this case applies the function in the current process.</p> <p><code>execute-with-interface-if-alive</code> is useful for automatic updating of interfaces that may be destroyed by the user, where the update is redundant if the interface is not alive.</p> <p>Note: All accesses (reads as well as writes) on a CAPI interface and its sub-elements should be performed in the interface process.</p>
See also	<code>execute-with-interface</code>

exit-confirmer

Function

Summary	The <code>exit-confirmer</code> function is called by the OK button on a dialog created with <code>popup-confirmer</code> .
Package	<code>capi</code>
Signature	<code>exit-confirmer</code> &rest <i>dummy-args</i>
Description	This is the function that is called by the OK button on a dialog created using <code>popup-confirmer</code> , and it is provided as an entry point so that other callbacks can behave in the same way. There is a full description of the OK button in <code>popup-confirmer</code> .

Example This example demonstrates the use of `exit-confirmer` to make the dialog exit when pressing `Return` in the text input pane. It also demonstrates the use of *value-function* as a means of deciding the return value from `popup-confirmer`.

```
(capi:popup-confirmer (make-instance
                      'capi:text-input-pane
                      :callback 'capi:exit-confirmer)
                      "Enter some text:"
                      :value-function
                      'capi:text-input-pane-text)
```

See also `popup-confirmer`
`display-dialog`
`interface`

exit-dialog

Function

Summary The `exit-dialog` function exits the current dialog.

Package `capi`

Signature `exit-dialog` *value*

Description This function is the means to successfully return a value from the current dialog. Hence, it might be called from an **OK** button so that pressing the button would cause the dialog to return successfully, whilst the **Cancel** button would call the counterpart function `abort-dialog`.

If there is no current dialog then `exit-dialog` does nothing and returns `nil`. If there is a current dialog then `exit-dialog` either returns non-`nil` or does a non-local exit. Therefore code that depends on `exit-dialog` returning must be written carefully - see the discussion under `abort-dialog` for details.

Example

```
(capi:display-dialog
 (capi:make-container
  (make-instance 'capi:text-input-pane
                 :callback-type :data
                 :callback 'capi:exit-dialog)
  :title "Test Dialog"))
```

There is another example in the file
`examples/capi/dialogs/simple-dialog.lisp`.

See also

- `abort-dialog`
- `display-dialog`
- `popup-confirmer`
- `interface`

expandable-item-pinboard-object

Class

Summary A class used to implement nodes in `graph-pane`.

Package `capi`

Superclasses `item-pinboard-object`

Description The class `expandable-item-pinboard-object` is a `pinboard-object` that `graph-pane` uses by default to implement nodes in a graph.

`expandable-item-pinboard-object` draws itself with a small circle to indicate that the node has children.

See also `graph-pane`

extended-selection-tree-view

Class

Summary A pane that displays a heirarchical list of items which (unlike `tree-view`) allows extended selection.

Package	<code>capi</code>
Superclasses	<code>tree-view</code>
Description	The class <code>extended-selection-tree-view</code> is like <code>tree-view</code> but allows more than one item to be selected at once.
See also	<code>tree-view</code>

filtering-layout

Class

Summary	A layout that can be used for filtering.
Package	<code>capi</code>
Superclasses	<code>row-layout</code>
Initargs	<p><code>:callback-object</code></p> <p>The argument for the callbacks. If it is <code>nil</code> the top-level-interface of the layout is used.</p> <p><code>:change-callback</code></p> <p>A function of one argument (the <i>callback-object</i>). It is called whenever the text in the filter changes. Also if <i>callback</i> is not supplied, <i>change-callback</i> is called instead.</p> <p><code>:callback</code></p> <p>A function of one argument (the <i>callback-object</i>). It is called when the user presses <code>Return</code>, makes a selection from the menu, or clicks the Confirm button. If <i>callback</i> is not supplied, <i>change-callback</i> is called instead.</p> <p><code>:text</code></p> <p>A string specifying the initial text of the filter, or <code>nil</code>.</p>

`:matches-title`

Controls whether the `filtering-layout` contains a `display-pane` (the "matches pane") showing the number of matches. If `matches-title` is a string, it provides the title of the matches pane, otherwise the title is **Matches:**. Note that the actual text in the matches pane must be set by the caller by `(setf capi:filtering-layout-matches-text)`.

Accessors `filtering-layout-state`
`filtering-layout-matches-text`

Description The main part of a filtering layout is a `text-input-pane` which allows the user to enter a string. The string is used for filtering. The user can control how it is used by a menu that allows her to specify whether:

- the string is used as a regular expression or plain string
- the filter excludes matches or includes matches
- filtering is case-sensitive or case-insensitive

The filtering layout defines the parameters to use, and calls the callbacks to perform the filtering. It does not do any filtering itself.

To actually do the filtering, the using code needs to call `filtering-layout-match-object-and-exclude-p`, which returns as multiple values a precompiled regexp and a flag specifying whether to exclude matches. The regexp should be used to perform the filtering, typically by using `lisp-works:find-regexp-in-string`. Note that `filtering-layout-match-object-and-exclude-p` returns `nil` when there is no string in the `text-input-pane`, and that even when the filter is set to plain match it is returns a regexp (which matches a plain string).

You supply a `filtering-layout` amongst the *panes* of your interface definition (not its *layouts*). The description of a `filtering-layout` is set by the `initialize-instance` method of the class, and therefore the description cannot be passed as an `initarg` and should not be manipulated.

`filtering-layout-state` returns a "state" object which can be used later to set the state of any `filtering-layout` by `(setf capi:filtering-layout-state)`. When setting the state, the value can also be a string or `nil`. A string means setting the filter string to it and making the filtering state be plain string, includes matches, and case-insensitive. `nil` means the same as the empty string.

Example

```
(defvar *things* (list "Foo" "Bar" "Baz" 'car 'cdr))

(capi:define-interface my-interface ()
  ((things :reader my-things
           :initform *things*))
  (:panes
   (my-things-list-panel
    capi:list-panel
    :reader my-interface-list-panel
    :items things
    :visible-min-height `(:character ,(length
*things*)))
   (my-filtering
    capi:filtering-layout
    :change-callback 'update-my-interface
    :reader my-interface-filtering))
  (:layouts
   (a-layout
    capi:column-layout
    '(my-filtering my-things-list-panel)))
  (:default-initargs :title "Filtering example")
  )

(defun update-my-interface (my-interface)
  (let* ((things (my-things my-interface))
         (filtered-things
          (multiple-value-bind (regexp excludep)
            (capi:filtering-layout-match-object-and-
exclude-p
             (my-interface-filtering my-interface)
             nil)
            (if regexp
                (loop for thing in things
                      when (if (find-regexp-in-string
                               regexp
                               (string thing))
                               (not excludep)
                               excludep)
                        collect thing)
                things))))
    (setf (capi:collection-items
           (my-interface-list-panel my-interface))
          filtered-things)))
```

See also

`filtering-layout-match-object-and-exclude-p`

filtering-layout-match-object-and-exclude-p *Function*

Summary	Returns filtering parameters for a <code>filtering-layout</code> .	
Package	<code>capi</code>	
Signature	<code>filtering-layout-match-object-and-exclude-p</code> <i>filtering-layout display-message</i> => <i>regexp, excludep</i>	
Arguments	<i>filtering-layout</i>	A <code>filtering-layout</code>
	<i>display-message</i>	A generalized boolean
Values	<i>regexp</i>	A precompiled regular expression
	<i>excludep</i>	A boolean
Description	<p>The function <code>filtering-layout-match-object-and-exclude-p</code> returns a <code>regexp</code> to use for filtering in the <i>filtering-layout</i>. The second returned value <i>excludep</i> specifies whether the filter should be used to exclude or include matches.</p> <p><i>display-message</i> is a generalised boolean controlling whether a message is displayed to the user if there is an error when compiling the <code>regexp</code>.</p> <p>See <code>filtering-layout</code> for details.</p>	
See also	<code>filtering-layout</code>	

find-graph-edge *Generic Function*

Summary	Finds and returns an edge in a graph given two items.	
Package	<code>capi</code>	
Signature	<code>find-graph-edge</code> <i>graph from to</i> => <i>edge</i>	
Arguments	<i>graph</i>	A <code>graph-pane</code> .

	<i>from</i>	An item in <i>graph</i> .
	<i>to</i>	An item in <i>graph</i> .
Values	<i>edge</i>	A graph edge, or <code>nil</code> .
Description	<p>The generic function <code>find-graph-edge</code> finds the edge that goes from the node corresponding to <i>from</i> to the node corresponding to <i>to</i>.</p> <p>If there is no such edge, <code>find-graph-edge</code> returns <code>nil</code>.</p>	
See also	<code>find-graph-node</code> <code>graph-pane</code>	

find-graph-node

Generic Function

Summary	Finds and returns a node in a graph corresponding to an item.	
Package	<code>capi</code>	
Signature	<code>find-graph-node graph object => node</code>	
Arguments	<i>graph</i>	A <code>graph-pane</code> .
	<i>object</i>	An item in <i>graph</i> .
Values	<i>node</i>	A node of <i>graph</i> , or <code>nil</code> .
Description	<p>The generic function <code>find-graph-node</code> finds the node that corresponds to the item <i>object</i>.</p> <p>If there is no such node, <code>find-graph-node</code> returns <code>nil</code>.</p>	
See also	<code>find-graph-edge</code> <code>graph-pane</code>	

find-interface*Generic Function*

Summary	Displays an interface of a given class, making it if necessary.	
Package	<code>capi</code>	
Signature	<code>find-interface</code> <i>class-name</i> &rest <i>initargs</i> &key <i>screen</i> &allow-other-keys => <i>interface</i>	
Arguments	<i>class-name</i>	A specifier for a subclass of <code>interface</code> .
	<i>initargs</i>	Initialization arguments for <i>class-name</i> .
	<i>screen</i>	A <code>screen</code> or <code>nil</code> .
Values	<i>interface</i>	An interface of class <i>class-name</i> .
Description	<p>The generic function <code>find-interface</code> finds and displays an interface of the given class <i>class-name</i> that matches <i>initargs</i> and <i>screen</i>.</p> <p><i>class-name</i> can be the name of a suitable class, the class itself, or an instance of the class.</p> <p><i>screen</i> can be a CAPI object as accepted by <code>convert-to-screen</code>. <i>screen</i> defaults to the default screen.</p> <p><code>find-interface</code> calls <code>locate-interface</code> to locate an existing interface:</p> <ol style="list-style-type: none"> 1. If an interface of the class specified by <i>class-name</i> matching <i>initargs</i> exists already on <i>screen</i>, then this interface is activated and returned. 2. Otherwise, if an interface of the class specified by <i>class-name</i> exists already on <i>screen</i>, then <code>reinitialize-interface</code> is applied to this interface which is then activated and returned. <p>If no instance of class <i>class-name</i> exists on <i>screen</i>, then <code>find-interface</code> creates one by passing <i>class-name</i> and <i>initargs</i> to <code>make-instance</code>, and displays the result on <i>screen</i>.</p>	

Note: The Common LispWorks development environment uses `find-interface` in many places.

See also `locate-interface`
`reinitialize-interface`

find-string-in-collection

Generic Function

Summary The `find-string-in-collection` generic function returns the next item whose printed representation matches a given string.

Package `capi`

Signature `find-string-in-collection self string &optional set`

Description The `find-string-in-collection` generic function returns the next item whose printed representation matches *string*. If *set* is non-`nil`, the choice selection is set to this item. The search is started from the previous search point. If the choice selection is set, the next search will start from the first selected item.

See also `collection-search`
`collection`

force-screen-update

Function

Summary Ensures a screen is up to date.

Package `capi`

Signature `force-screen-update &key screen`

Description The function `force-screen-update` makes sure that the screen specified by `screen` is up to date.

`screen` can be a CAPI object as accepted by `convert-to-screen`. The default for `screen` is `nil`.

See also `force-update-all-screens`

force-update-all-screens

Function

Summary Ensures a screen is up to date.

Package `capi`

Signature `force-update-all-screens`

Description The function `force-update-all-screens` makes sure that all screens are up to date.

See also `force-screen-update`

foreign-owned-interface

Class

Package `capi`

Superclasses `interface`

Description The class `foreign-owned-interface` allows another application's window to be the owner of a CAPI dialog. Instances should be created by calling `make-foreign-owned-interface`.

`foreign-owned-interface` is implemented only on Microsoft Windows.

See also `make-foreign-owned-interface`

form-layout

Class

Summary	The class <code>form-layout</code> lays its children out in a form.
Package	<code>capi</code>
Superclasses	<code>layout</code>
Initargs	<code>:vertical-gap</code> The gap between rows in the form. <code>:vertical-adjust</code> The adjustment made to the rows. <code>:title-gap</code> The gap between the two columns. <code>:title-adjust</code> The adjustment made to the left column.
Accessors	<code>form-vertical-gap</code> <code>form-vertical-adjust</code> <code>form-title-gap</code> <code>form-title-adjust</code>
Description	The form layout lays its children out in two columns, where the children in the left column (which are usually titles) are right adjusted whilst the children in the right column are left adjusted.
Compatibility Note	This class has been superseded by <code>grid-layout</code> , and will probably be removed at some point in the future. The examples below demonstrate the use of grid layouts as an alternative to forms.
Examples	<pre>(setq children (list "Button:" (make-instance 'capi:push-button :text "Press Me") "Enter Text:" (make-instance 'capi:text-input-pane) "List:" (make-instance 'capi:list-panel :items '(1 2 3))))</pre>

```
(capi:contain (make-instance
               'capi:grid-layout
               :description children
               :x-adjust '(:right :left)
               :y-adjust :center))
```

See also `grid-layout`
`layout`

free-metatile

Function

Summary Frees a metatile.

Package `capi`

Signature `free-metatile metatile`

Arguments `metatile` A metatile.

Description The function `free-metatile` releases the window system storage used by the metatile.

`free-metatile` must be called when the metatile is no longer needed, to avoid memory leaks.

`free-metatile` is not implemented on X11/Motif.

See also `clipboard`
`draw-metatile`
`draw-metatile-to-image`

free-sound

Function

Summary Frees a loaded sound object.

Package `capi`

Signature	<code>free-sound</code> <i>sound</i>
Arguments	<i>sound</i> An array returned by <code>load-sound</code> .
Description	The function <code>free-sound</code> unloads (frees) the loaded sound object <i>sound</i> .
See also	<code>load-sound</code> <code>read-sound-file</code>

get-collection-item

Generic Function

Summary	Returns the item at a specified position in a collection.
Package	<code>capi</code>
Signature	<code>get-collection-item</code> <i>self</i> <i>index</i>
Description	The generic function <code>get-collection-item</code> returns the item at position <i>index</i> from the collection <i>self</i> . It achieves this by calling the <i>items-get-function</i> of the collection. There is also a complementary function, <code>search-for-item</code> which finds the index for a given item in a collection.
See also	<code>collection</code> <code>search-for-item</code>

get-constraints

Function

Summary	Returns a list of the constraints for an element.
Package	<code>capi</code>
Signature	<code>get-constraints</code> <i>element</i>

The valid *keys* are:

`:min-range` The minimum data coordinate.

`:max-range` The maximum data coordinate.

`:slug-position`
 The current scroll position.

`:slug-size` The length of the scroll bar slug.

`:page-size` The scroll page size.

`:step-size` The scroll step size.

Note: For the other pane classes, such as `list-pane1`, the underlying widget decides what the scroll range and units are.

Example See the following CAPI example files:
`output-panes/scroll-test.lisp`
`output-panes/scrolling-without-bar.lisp`

See also `get-scroll-position`
 `scroll`
 `set-horizontal-scroll-parameters`
 `simple-pane`

get-page-area

Function

Summary Calculates the dimensions of suitable rectangles for use with `with-page-transform`.

Package `capi`

Signature `get-page-area printer &key scale dpi screen`

Description The `get-page-area` function is provided to simplify the calculation of suitable rectangles for use with `with-page-transform`. It calculates and returns the width and height of

the rectangle in the user's coordinate space that corresponds to one printable page, based on the logical resolution of the user's coordinate space in dpi.

For example, if a logical resolution of 72 dpi was specified, this means that each unit in user space would map onto 1/72 of an inch on the printed page, assuming that no *scale* is specified.

If *dpi* is `nil` or unspecified, the logical resolution of the specified screen is used, or the logical resolution of the default screen if no screen is specified. The *dpi* argument can be a number, or a list of two elements representing the logical resolution of the coordinate spaces in the x and y directions respectively.

If *scale* is specified the rectangle is calculated so that the image is scaled by this factor when printed. It defaults to 1.0.

See also `printer-metrics`
`with-page-transform`

get-printer-metrics

Function

Summary Returns the metrics for a printer.

Package `capi`

Signature `get-printer-metrics printer`

Description The `get-printer-metrics` function takes a *printer* as its argument and returns a `printer-metrics` object.

The metrics values in this object should be accessed by the `printer-metrics` readers.

See also `set-printer-metrics`
`printer-metrics`
`with-page-transform`

get-scroll-position

Function

Summary Returns the current scroll position of a pane such as `list-panel`, `display-pane` or `tree-view`.

Package `capi`

Signature `get-scroll-position pane dimension => position`

Arguments *pane* A pane with built-in scrolling.
dimension A keyword, either `:horizontal` or `:vertical`.

Values *position* An integer.

Description The function `get-scroll-position` returns the scroll position of the pane *pane* in the given *dimension*.
pane should be an instance of a pane class that has built-in scrolling. That is, the scrolling is implemented by the underlying widget. Examples include `list-panel`, `display-pane` and `tree-view`.

In general, the units in the returned value *position* are unspecified, but they can be passed to the generic function `scroll` with *operation* `:move` to restore the position.

For a `list-panel`, the vertical units are items.

See also `get-horizontal-scroll-parameters`
`get-vertical-scroll-parameters`
`scroll`

get-vertical-scroll-parameters*Generic Function*

Summary	Queries the scroll parameters of a vertical scroll bar.	
Package	<code>capi</code>	
Signature	<code>get-vertical-scroll-parameters self &rest keys</code> <code>=> parameter, parameter,...</code>	
Arguments	<i>self</i>	A displayed <code>output-pane</code> or <code>layout</code> .
	<i>keys</i>	Keywords as below.
Values	<i>parameter</i>	The parameters are returned as multiple values, one for each key passed in <i>keys</i> and in the same order as the arguments.
Description	<p>The function <code>get-vertical-scroll-parameters</code> retrieves the specified parameters of the vertical scroll bar of <i>self</i>, which should be a displayed instance of a subclass of <code>output-pane</code> (such as <code>editor-pane</code>) or <code>layout</code>.</p> <p>The valid <i>keys</i> are:</p> <ul style="list-style-type: none"> <code>:min-range</code> The minimum data coordinate. <code>:max-range</code> The maximum data coordinate. <code>:slug-position</code> The current scroll position. <code>:slug-size</code> The length of the scroll bar slug. <code>:page-size</code> The scroll page size. <code>:step-size</code> The scroll step size. <p>Note: For the other pane classes, such as <code>list-pane1</code>, the underlying widget decides what the scroll range and units are.</p>	
Example	See the following CAPI example files:	

`output-panes/scroll-test.lisp`
`output-panes/scrolling-without-bar.lisp`

See also `get-scroll-position`
`scroll`
`get-horizontal-scroll-parameters`
`simple-pane`

graph-edge

Class

Summary The class of objects that represent edges in a graph.

Package `capi`

Superclasses `graph-object`

Initargs `:from` The node where the edge starts.
 `:to` The node where the edge ends.

Accessors `graph-edge-from`
 `graph-edge-to`

Description The class of objects that represent edges in a `graph-pane`.
from and *to* are the nodes that the edge connects.

See also `graph-pane`

graph-node

Class

Summary The class of objects that represent nodes in a graph.

Package `capi`

Superclasses `graph-object`

Readers	<code>graph-node-x</code> <code>graph-node-y</code> <code>graph-node-width</code> <code>graph-node-height</code> <code>graph-node-in-edges</code> <code>graph-node-out-edges</code>
Description	The default class of nodes in a <code>graph-pane</code> . The <code>graph-pane</code> generates a graph of <code>graph-node</code> and <code>graph-edge</code> objects.
See also	<code>graph-edge</code> <code>graph-pane</code>

graph-node-children

Generic Function

Summary	Returns the children of a graph node.
Package	<code>capi</code>
Signature	<code>graph-node-children node => result</code>
Arguments	<i>node</i> A <code>graph-node</code> .
Values	<i>result</i> A list.
Description	The generic function <code>graph-node-children</code> returns a list of all the 'children' of the node <i>node</i> . These children are the nodes which are at the other end of some edge in the <code>graph-node-out-edges</code> of the <code>graph-node node</code> .
See also	<code>graph-node</code>

graph-object

Class

Summary	The superclass of node and edge objects.
---------	--

Package	<code>capi</code>
Subclasses	<code>graph-edge</code> <code>graph-node</code>
Readers	<code>graph-object-element</code> <code>graph-object-object</code>
Description	<p>The class <code>graph-object</code> is the superclass of <code>graph-edge</code> and <code>graph-node</code>.</p> <p>The reader <code>graph-object-element</code> returns the CAPI object that is displayed.</p> <p>The reader <code>graph-object-object</code> returns the user object associated with the graph object.</p>

graph-pane

Class

Summary	A graph pane is a pane that displays a hierarchy of items in a graph.	
Package	<code>capi</code>	
Superclasses	<code>simple-pinboard-layout</code> <code>choice</code>	
Subclasses	<code>simple-network-pane</code>	
Initargs	<code>:roots</code>	The roots of the graph.
	<code>:children-function</code>	Returns the children of a node.
	<code>:layout-function</code>	A keyword denoting how to layout the nodes.

`:layout-x-adjust`

The adjust value for the x direction.

`:layout-y-adjust`

The adjust value for the y direction.

`:node-pinboard-class`

The class of pane to represent nodes.

`:edge-pinboard-class`

The class of pane to represent edges.

`:node-pane-function`

A function to return a pane for each node.

Accessors

`graph-pane-layout-function`

`graph-pane-roots`

Description

A graph pane calculates the items of the graph by calling the *children-function* on each of its *roots*, and then calling it again on each of the children recursively until no more children are found. The *children-function* gets called with an item of the graph and should return a list of the children of that item.

Each item is represented by a node in the graph.

The *layout-function* tells the graph pane how to lay out its nodes. It can be one these values:

`:left-right` Lay the graph out from the left to the right.

`:top-down` Lay the graph out from the top down.

`:right-left` Lay the graph out from the right to the left.

`:bottom-up` Lay the graph out from the bottom up.

layout-x-adjust and *layout-y-adjust* act on the underlying layout to decide where to place the nodes. The values should be a keyword or a list of the form (*keyword n*) where *n* is an

integer. These values of *adjust* are interpreted as by `pane-adjusted-position`. `:top` is the default for *layout-y-adjust* and `:left` is the default for *layout-x-adjust*.

When a graph pane wants to display nodes and edges, it creates instances of *node-pinboard-class* and *edge-pinboard-class* which default to `item-pinboard-object` and `line-pinboard-object` respectively. These classes must be subclasses of `simple-pane` or `pinboard-object`, and there are some examples of the use of these keywords below.

The *node-pane-function* is called to create a pane for each node, and by default it creates an instance of *node-pinboard-class*. It gets passed the graph pane and the item corresponding to the node, and should return an instance of a subclass of `simple-pane` or `pinboard-object`.

To expand or contract a node, the user clicks on the circle next to the node. An expandable node has a unfilled circle and a collapsable node has a filled circle.

`graph-pane` is a subclass of `choice`, so for details of its selection handling, see `choice`.

The highlighting of the children is controlled as described for `pinboard-layout`, but for `graph-pane` the default value of *highlight-style* is `:standard`.

Compatibility Note

In LispWorks 4.3 the double click gesture on a `graph-pane` node always calls the *action-callback*, and the user gesture to expand or collapse a node is to click on the circle drawn alongside the node.

In LispWorks 4.2 and previous versions, the double click gesture was used for expansion and contraction of nodes and the *action-callback* was not always called.

Examples

```
(defun node-children (node)
  (when (< node 16)
    (list (* node 2)
          (1+ (* node 2))))))
```

```

(setq graph
  (capi:contain
    (make-instance 'capi:graph-pane
      :roots '(1)
      :children-function
      'node-children)
    :best-width 300 :best-height 400))

(capi:apply-in-pane-process
 graph #'(setf capi:graph-pane-roots) '(2 6) graph)

(capi:contain
 (make-instance 'capi:graph-pane
  :roots '(1)
  :children-function
  'node-children
  :layout-function :top-down)
 :best-width 300 :best-height 400)

(capi:contain
 (make-instance 'capi:graph-pane
  :roots '(1)
  :children-function
  'node-children
  :layout-function :top-down
  :layout-x-adjust :left)
 :best-width 300 :best-height 400)

```

This example demonstrates a different style of graph output with right-angle edges and parent nodes being adjusted towards the top instead of at the center.

```

(capi:contain
 (make-instance
  'capi:graph-pane
  :roots '(1)
  :children-function 'node-children
  :layout-y-adjust '(:top 10)
  :edge-pinboard-class
  'capi:right-angle-line-pinboard-object)
 :best-width 300
 :best-height 400)

```

This example demonstrates the use of `:node-pinboard-class` to specify that the nodes are drawn as push buttons.

```
(capi:contain
 (make-instance
  'capi:graph-pane
  :roots '(1)
  :children-function 'node-children
  :node-pinboard-class 'capi:push-button)
 :best-width 300
 :best-height 400)
```

There are more examples in the directory
examples/capi/graphics/.

See also `item-pinboard-object`
`line-pinboard-object`

graph-pane-add-graph-node

Generic Function

Summary Adds a node to a graph.

Package `capi`

Signature `graph-pane-add-graph-node` *graph-pane object*
parent-node => new-node

Arguments *graph-pane* A `graph-pane`.

object An object.

parent-node A `graph-node`.

Values *new-node* A `graph-node`.

Description The generic function `graph-pane-add-graph-node` adds a new node in the graph *graph-pane* corresponding to *object*, and links it as a child of *parent-node*.

See also `graph-node`
`graph-pane`

graph-pane-delete-object*Generic Function*

Summary	Removes a node from a graph.
Package	<code>capi</code>
Signature	<code>graph-pane-delete-object</code> <i>graph-pane</i> <i>object</i>
Arguments	<i>graph-pane</i> A graph-pane. <i>object</i> An object.
Description	The generic function <code>graph-pane-delete-object</code> deletes the node corresponding to <i>object</i> in the graph <i>graph-pane</i> .
See also	<code>graph-node</code> <code>graph-pane</code> <code>graph-pane-add-graph-node</code> <code>graph-pane-delete-objects</code>

graph-pane-delete-objects*Generic Function*

Summary	Removes nodes from a graph.
Package	<code>capi</code>
Signature	<code>graph-pane-delete-objects</code> <i>graph-pane</i> <i>objects</i>
Arguments	<i>graph-pane</i> A graph-pane. <i>objects</i> A list of objects.
Description	The generic function <code>graph-pane-delete-objects</code> deletes the node in the graph <i>graph-pane</i> corresponding to each object in the list <i>objects</i> .

See also `graph-node`
`graph-pane`
`graph-pane-delete-object`

graph-pane-delete-selected-objects

Generic Function

Summary Removes selected nodes from a graph.

Package `capi`

Signature `graph-pane-delete-selected-objects` *graph-pane*

Arguments *graph-pane* A `graph-pane`.

Description The generic function `graph-pane-delete-selected-objects` deletes the currently selected nodes in the graph *graph-pane*.

See also `graph-node`
`graph-pane`
`graph-pane-delete-object`

graph-pane-direction

Generic Function

Summary Returns or sets the direction of a graph.

Package `capi`

Signature `graph-pane-direction` *graph-pane* => *direction*

Signature `(setf graph-pane-direction)` *direction* *graph-pane* => *direction*

Arguments *graph-pane* A `graph-pane`.

Values *direction* One of `:forwards` or `:backwards`.

Description	<p>The generic function <code>graph-pane-direction</code> returns the direction of the graph <i>graph-pane</i>. If the <i>layout-function</i> of <i>graph-pane</i> is <code>:top-down</code> or <code>:left-right</code> then <i>direction</i> is <code>:forwards</code>. Otherwise <i>direction</i> is <code>:backwards</code>.</p> <p>The generic function <code>(setf graph-pane-direction)</code> maintains the dimension of the <i>layout-function</i> but potentially reverses its direction.</p>
Example	<pre>(setf gp (make-instance 'capi:graph-pane :layout-function :top-down)) => #<CAPI:GRAPH-PANE [0 items] 20603294> (setf (capi:graph-pane-direction gp) :backwards) => NIL (capi:graph-pane-layout-function gp) => :TOP-DOWN</pre>
See also	<code>graph-pane</code>

graph-pane-edges*Function*

Summary	Returns the edges of a graph.
Package	<code>capi</code>
Signature	<code>graph-pane-edges <i>graph-pane</i> => <i>edges</i></code>
Arguments	<i>graph-pane</i> A <code>graph-pane</code> .
Values	<i>edges</i> A list.

Description The function `graph-pane-edges` returns a list of all the `graph-edge` objects in the graph *graph-pane*.

See also `graph-edge`
 `graph-pane`

graph-pane-nodes

Function

Summary Returns the nodes of a graph.

Package `capi`

Signature `graph-pane-nodes graph-pane => nodes`

Arguments *graph-pane* A `graph-pane`.

Values *nodes* A list.

Description The function `graph-pane-nodes` returns a list of all the `graph-node` objects in the graph *graph-pane*.

See also `graph-node`
 `graph-pane`

graph-pane-object-at-position

Function

Summary Returns the graph object at a given position in a graph.

Package `capi`

Signature `graph-pane-object-at-position graph-pane x y => object`

Arguments *graph-pane* A `graph-pane`.

Values *object* A `graph-object`, or `nil`.

`x, y` Non-negative numbers.

Description The function `graph-pane-object-at-position` returns the `graph-object` (either a `graph-edge` or a `graph-node`) at the coordinates `x, y` in the graph *graph-pane*.

If there is no `graph-object` at position `x,y` then `graph-pane-object-at-position` returns `nil`.

See also `graph-pane`

graph-pane-select-graph-nodes

Generic Function

Summary Selects nodes in a graph according to a predicate.

Package `capi`

Signature `graph-pane-select-graph-nodes` *graph-pane predicate*

Arguments *graph-pane* A `graph-pane`.

predicate A function of one argument with boolean result.

Description The generic function `graph-pane-select-graph-nodes` applies *predicate* to all of the `graph-nodes` in *graph-pane*, and sets the *selected-items* to be the objects corresponding to those nodes for which *predicate* returns a true value.

See also `choice-selected-items`
`graph-node`
`graph-pane`

graph-pane-update-moved-objects

Generic Function

Summary Updates a graph after the user moves objects.

Package	<code>capi</code>
Signature	<code>graph-pane-update-moved-objects</code> <i>graph-pane objects</i>
Arguments	<i>graph-pane</i> A <code>graph-pane</code> . <i>objects</i> A list.
Description	The generic function <code>graph-pane-update-moved-objects</code> is called after some objects in the graph <i>graph-pane</i> were moved by a user gesture. <i>objects</i> is a list containing the objects that were moved. The primary method updates the geometry of edges connected to the moved objects. You can add non-primary methods to perform other operations at that point.
See also	<code>graph-pane</code>

grid-layout

Class

Summary	The <code>grid-layout</code> is a layout which positions its children on a two dimensional grid.
Package	<code>capi</code>
Superclasses	<code>x-y-adjustable-layout</code>
Subclasses	<code>row-layout</code> <code>column-layout</code>
Initargs	<code>:columns</code> The number of columns in the grid. <code>:has-title-column-p</code> A boolean specifying whether the first column is a title column. <code>:orientation</code> The orientation of the children.

<code>:rows</code>	The number of rows in the grid.
<code>:x-ratios</code>	The ratios between the columns.
<code>:y-ratios</code>	The ratios between the rows.
<code>:x-gap</code>	The gap between each column.
<code>:y-gap</code>	The gap between each row.
<code>:x-uniform-size-p</code>	If <code>t</code> , make each of the columns the same size.
<code>:y-uniform-size-p</code>	If <code>t</code> , make each of the rows the same size.

Accessors

`layout-x-ratios`
`layout-y-ratios`
`layout-x-gap`
`layout-y-gap`

Description

The row and column sizes are controlled by the constraints on their children. For example, the *visible-min-width* of any column is the maximum of the *visible-min-width* in of the children in the column. The size of the layout is controlled by the constraints on the rows and columns.

For `grid-layout` *description* is either a two dimensional array or a list in the order specified by *orientation* (which defaults to `:row`). In the case of a list, one of *columns* or *rows* can be supplied to specify the dimensions (the default is two columns). As well as panes, slot names and strings, *description* may contain the element `nil`, which is interpreted as a special dummy pane with suitable geometry for resizable gaps. This special interpretation of `nil` in the *description* is specific to `grid-layout` and its subclasses.

The *x-ratios* and *y-ratios* slots control the sizes of the elements in a grid layout in the following manner:

The elements of *x-ratios* (or *y-ratios*) control the size of each child relative to the others. If an element in *x-ratios* (or *y-ratios*) is `nil` the child is fixed at its minimum size. Otherwise the size is calculated as follows

```
(round (* total ratio) ratio-sum)
```

where *ratio-sum* is the sum of the non-`nil` elements of *x-ratios* (or *y-ratios*) and *ratio* is the element of ratios corresponding to the child. If this ideal ratio size does not fit the maximum or minimum constraints on the child size, and the constraint means that changing the ratio size would not assist the sum of the child sizes fitting the total space available, then the child is fixed at its constrained size, the child is removed from the ratio calculation, and the calculation is performed again. If *x-ratios* (or *y-ratios*) has fewer elements than the number of children, 1 is used for each of the missing ratios. Leaving *x-ratios* (or *y-ratios*) `nil` causes all of the children to be the same size.

The positions of each pane in the layout can be specified using *x-adjust* and *y-adjust* like every other `x-y-adjustable-layout`, except that if there is one value then it is used for all of the panes, whereas if it is a list then each value in the list refers to one row or column. If the list does not contain a value for every row or column then the last value is taken to refer to all of the remaining panes.

If *has-title-column-p* is true, then the items in the description which correspond to the first column are treated specially:

A string Equivalent to specifying `(:title string)`

A list of the form `(:title string . options)`

Make a title using the given list as initargs.
options is a plist of options, which can include the keys `:title-font`, `:title-args`, `:mnemonic` or `:mnemonic-escape`. See `titled-object` for how these are processed.

A list of the form (`:mnemonic-title string . options`)

Make a title using the given list as initargs. *string* can contain the mnemonic escape. *options* is a plist of options, which can include the keys `:title-font`, `:title-args`, or `:mnemonic-escape`. See `titled-object` for how these are processed.

Note: mnemonics are not supported on all platforms.

Example

```
(capi:contain (make-instance
               'capi:grid-layout
               :description '("1" "2" "3"
                             "4" "5" "6"
                             "7" "8" "9")
               :columns 3))

(capi:contain (make-instance
               'capi:grid-layout
               :description (list "List:"
                                  (make-instance
                                   'capi:list-panel
                                   :items '(1 2 3))
                                  "Buttons:"
                                  (make-instance
                                   'capi:button-panel
                                   :items '(1 2 3))))))

(capi:contain (make-instance
               'capi:grid-layout
               :description (list "List:"
                                  (make-instance
                                   'capi:list-panel
                                   :items '(1 2 3))
                                  "Buttons:"
                                  (make-instance
                                   'capi:button-panel
                                   :items '(1 2 3))))
               :x-adjust '(:right :left)
               :y-adjust '(:center :bottom)))
```

```
(capi:contain (make-instance
               'capi:grid-layout
               :description (list "List:"
                                  (make-instance
                                   'capi:list-panel
                                   :items '(1 2 3))
                                  "Buttons:"
                                  (make-instance
                                   'capi:button-panel
                                   :items '(1 2 3)))
               :orientation :column))
```

This example illustrates the special interpretation of `nil` in the *description*:

```
(capi:contain
 (make-instance
  'capi:grid-layout
  :description
  (cdr
   (loop for i below 5
         appending
         (list
          nil
          (make-instance 'capi:simple-pane
                        :background :red
                        :visible-min-width 50
                        :visible-max-width t
                        :visible-min-height 50
                        :visible-max-height t))))
  :columns 3)
 :height 150 :width 150 :title "Resize Me")
```

There are more examples in the directory `examples/capi/applications/`.

This example is a grid with `:has-title-column-p t`: `examples/capi/layouts/titles-in-grid.lisp`

See also `layout`

hide-interface*Function*

Summary	The function <code>hide-interface</code> hides the interface containing a specified pane.
Package	<code>capi</code>
Signature	<code>hide-interface pane &optional iconify</code>
Description	The function <code>hide-interface</code> hides the interface containing <i>pane</i> from the screen. If <i>iconify</i> is non- <code>nil</code> then it will iconify it, else it will just remove it from the screen. To show it again, use <code>show-interface</code> . The default value of <i>iconify</i> is <code>t</code> .
See also	<code>interface</code> <code>show-interface</code> <code>quit-interface</code>

hide-pane*Generic Function*

Summary	Hides the specified pane.
Package	<code>capi</code>
Signature	<code>hide-pane pane => pane</code>
Arguments	<i>pane</i> An instance of <code>simple-pane</code> or a subclass.
Description	The function <code>hide-pane</code> hides the pane <i>pane</i> , removing it from the screen. <i>pane</i> 's children, if any, are hidden too. To restore <i>pane</i> to the screen, use <code>show-pane</code> .
See also	<code>hide-interface</code> <code>show-pane</code>

highlight-pinboard-object

Generic Function

Summary	Highlights a specified pinboard object.
Package	<code>capi</code>
Signature	<code>highlight-pinboard-object</code> <i>pinboard object</i> &key <i>redisplay</i>
Arguments	<i>pinboard</i> A <code>pinboard-layout</code> . <i>object</i> A <code>pinboard-object</code> . <i>redisplay</i> A generalised boolean.
Description	<p>The generic function <code>highlight-pinboard-object</code> causes the pinboard object <i>object</i> to become highlighted until <code>unhighlight-pinboard-object</code> is called on it.</p> <p>The pinboard object highlighting is drawn according to the <i>highlight-style</i> of the <code>pinboard-layout</code> <i>pinboard</i>.</p> <p>If <i>redisplay</i> is non-<code>nil</code> the highlighting is drawn immediately. The default value for <i>redisplay</i> is <code>t</code>.</p>
See also	<code>unhighlight-pinboard-object</code> <code>draw-pinboard-object-highlighted</code> <code>pinboard-object</code> <code>pinboard-layout</code>

image-list

Class

Summary	An object used to manage the images displayed by tree views and list views.
Package	<code>capi</code>
Superclasses	<code>capi-object</code>

Initargs	<p><code>:image-width</code> The width of the images in this image list.</p> <p><code>:image-height</code> The height of the images in this image list.</p> <p><code>:image-sets</code> A list of images or image sets.</p>
Description	<p>The <code>:image-sets</code> initarg specifies a list. Each item in the list <i>image-sets</i> may be one of the following.</p> <p>A pathname or string</p> <p style="padding-left: 40px;">This specifies the filename of a file suitable for loading with <code>load-image</code>.</p> <p>A symbol</p> <p style="padding-left: 40px;">The symbol must be a predefined image identifier, or have been registered by means of a call to <code>register-image-translation</code>.</p> <p>An image object, as returned by <code>load-image</code>.</p> <p>An image-set object</p> <p style="padding-left: 40px;">See <code>image-set</code> for further details.</p> <p>Note that image sets are added in their entirety; it is not possible to use image-locators to extract a single image from an image set.</p> <p>The images added to the image list are numbered in order, starting from zero. An image-set containing <i>n</i> images contributes <i>n</i> images to the image list, and hence consumes <i>n</i> consecutive integer indices.</p>
Examples	<p>See the files</p> <p><code>examples/capi/choice/tree-view.lisp</code></p> <p><code>examples/capi/choice/extended-selection-tree-view.lisp</code></p>
See also	<p><code>image-set</code></p> <p><code>load-image</code></p> <p><code>register-image-translation</code></p>

image-pinboard-object

Class

Summary	An image pinboard object is a pinboard object that displays itself as an image.
Package	<code>capi</code>
Superclasses	<code>pinboard-object</code> <code>titled-object</code>
Initargs	<code>:image</code> The image to be displayed.
Accessors	<code>image-pinboard-object-image</code>
Description	The <i>image</i> initarg for an <code>image-pinboard-object</code> should either be an <code>external-image</code> or any other object accepted by <code>load-image</code> . The image displayed in the object can be changed dynamically using the writer function <code>(setf image-pinboard-object-image)</code>

Example

```
(cd (sys:lispworks-dir "examples/capi/"))

(setf image
  (capi:contain
    (make-instance
      'capi:image-pinboard-object
      :image "applications/images/info.bmp")))

(capi:apply-in-pane-process
  (capi:element-parent image)
  #'(setf capi:image-pinboard-object-image)
  "graphics/lwsplash.bmp" image)

(capi:apply-in-pane-process
  (capi:element-parent image)
  #'(setf capi:image-pinboard-object-image)
  "applications/images/info.bmp" image)

(capi:contain
  (make-instance
    'capi:image-pinboard-object
    :image "graphics/lwsplash.bmp"
    :title "LispWorks Splashscreen"
    :title-adjust :right
    :title-position :bottom))
```

See also

pinboard-layout

image-set*Class*

Package

capi

Description

An image set is an object that identifies the location of an image. The image is typically a large image to be broken down into sub-images. The sub-images must all have the same size and be positioned side by side.

The following functions are available to create image set objects:

See also `make-general-image-set`
`make-icon-resource-image-set`
`make-scaled-image-set`
`make-scaled-general-image-set`
`make-resource-image-set`

install-postscript-printer

Function

Summary Installs or modifies a Postscript printer definition.

Package `capi`

Signature `install-postscript-printer` *name* &key *if-exists* *default* *savep*
ppd-file *description* *use-jcl* *command*
use-file *always-print-to-file*
orientation *installed-options*

Arguments

<i>name</i>	A string.
<i>if-exists</i>	One of <code>:supersede</code> , <code>:error</code> or <code>nil</code> .
<i>default</i>	One of <code>t</code> , <code>nil</code> or <code>:when-none</code> .
<i>savep</i>	A boolean.
<i>ppd-file</i>	A string or pathname.
<i>description</i>	A string, or <code>:preserve</code> .
<i>use-jcl</i>	A boolean, or <code>:preserve</code> .
<i>command</i>	A string, or <code>:preserve</code> .
<i>use-file</i>	A boolean, or <code>:preserve</code> .
<i>always-print-to-file</i>	A boolean, or <code>:preserve</code> .
<i>orientation</i>	One of <code>:landscape</code> , <code>:portrait</code> or <code>:preserve</code> .
<i>installed-options</i>	An association list, or <code>:preserve</code> .

Description The function `install-postscript-printer` installs or modifies a Postscript printer definition for the given printer name.

This applies only on Unix.

name is a string naming the printer.

if-exists controls what happens if the named printer is already known. The default value is `:supersede`.

default controls whether the default printer is set. The value `t` forces the default printer to be set. The value `:when-none` causes the default printer to be set if there is currently no default. The default value of *default* is `nil`.

savep, if true, causes the printer to be saved for subsequent sessions, by writing a file to the path specified by the first item of `*printer-search-path*`.

ppd-file, if non-`nil`, should be a pathname or string specifying the name of a PPD file (PostScript Printer Description File) which comes with the printer and specifies the printer properties. *ppd-file* must be supplied when installing a new printer. The default value is `nil`.

All the other arguments provide optional printer information. Each defaults to the value `:preserve`, which means that appropriate defaults are used. These correspond to the settings on the dialog displayed by `printer-configuration-dialog`. Non-default values are as follows:

description is a string describing the printer.

use-jcl controls whether to use Job Control Language (JCL).

command is the command to execute to print with the printer.

use-file controls how to pass data to the printer. A true value means a file is used, `nil` means a pipe is used.

always-print-to-file controls whether printing always goes to a file.

orientation controls the orientation of the output.

installed-options is an association list, with pairs of strings where the *car* is an option name and the *cdr* is its value. Which options are available and their potential values is defined by the *OpenUI/*CloseUI and *JCLOpenUI/*JCLCloseUI entries in the PPD file.

See also `printer-configuration-dialog`
`*ppd-directory*`
`*printer-search-path*`
`uninstall-postscript-printer`

installed-libraries

Function

Summary	Returns the installed libraries.
Package	<code>capi</code>
Signature	<code>installed-libraries => <i>libraries</i></code>
Values	<i>libraries</i> A list of library names.
Description	<p>The function <code>installed-libraries</code> returns the list of installed CAPI libraries.</p> <p>A library name is a keyword naming a library.</p> <p>On UNIX and Linux platforms, currently <i>libraries</i> is always <code>(:motif)</code>.</p> <p>On Microsoft Windows platforms, currently <i>libraries</i> is always <code>(:win32)</code>.</p> <p>On Mac OS X platforms, currently <i>libraries</i> is always <code>(:cocoa)</code>.</p>
See also	<code>default-library</code>

interactive-pane*Class*

Summary	An <code>interactive-pane</code> is an editor with a process reading and processing input, and that collects any output into itself. The class <code>listener-pane</code> is built upon this, and adds functionality for handling Lisp forms.
Package	<code>capi</code>
Superclasses	<code>editor-pane</code>
Subclasses	<code>listener-pane</code> <code>shell-pane</code>
Initargs	<code>:top-level-function</code> The input processing function.
Readers	<code>interactive-pane-stream</code> <code>interactive-pane-top-level-function</code>
Description	<p>An <code>interactive-pane</code> contains its own GUI stream. The <i>top-level-function</i> is called once, when the interactive pane is created: it needs to repeatedly take input from the GUI stream and write output to it.</p> <p>The first argument to <i>top-level-function</i> is the interface containing the interactive pane. The second argument is the interactive pane itself. The third argument is the GUI stream. The default for <i>top-level-function</i> is a function which runs a Lisp listener top-loop.</p>
Compatibility Note	This class was named <code>interactive-stream</code> in LispWorks 3.2 but has been renamed to avoid confusion (this class is not a stream but a pane that contains a stream). The class <code>interactive-stream</code> and its accessors <code>interactive-stream-top-level-function</code> and <code>interactive-stream-stream</code> have been kept for compatibility but may be dropped in future versions of LispWorks.

Example This example assumes there is just one line of output from each command sent to the pipe

```
(capi:contain
  (make-instance
    'capi:interactive-pane
    :top-level-function
    #'(lambda (interface pane stream)
        (declare (ignore interface pane))
        (with-open-stream (s (sys:open-pipe
                              '("/usr/local/bin/bash")
                              :direction :io))
          (loop
            (progn
              (format stream "primitive xterm$ ")
              (let ((input (read-line stream nil nil)))
                (if input
                  (progn
                     (write-line input s)
                     (force-output s))
                  (return))))
              (let ((output (read-line s nil nil)))
                (if output
                  (progn
                     (write-line output stream)
                     (force-output stream))
                  (return))))))))
    :best-height 300
    :best-width 300)
```

See also `collector-pane`

interactive-pane-execute-command

Generic Function

Summary Simulates user entry of commands in an `interactive-pane`.

Package `capi`

Signature `interactive-pane-execute-command` *interactive-pane* *command* &key *command-modification-function* *editp* &allow-other-keys

Arguments *interactive-pane* An `interactive-pane`.

command A Lisp form.
command-modification-function
 A function or `nil`.
editp A generalized boolean.

Description The generic function `interactive-pane-execute-command` has the same effect as the user typing the Lisp form *command* into the `interactive-pane` *interactive-pane*, and pressing **Return**.

`interactive-pane-execute-command` may be called from any process.

If *command-modification-function* is non-`nil`, it is a function of one argument. It is called with argument *command* in the process in which *interactive-pane* runs. The result of this call is used as the command to enter. The default value of *command-modification-function* is `nil`.

If *editp* is true then the command is left at the end of the pane for the user to edit before pressing **Return**. If *editp* is `nil` then `interactive-pane-execute-command` simulates the user pressing **Return**. The default value of *editp* is `nil`.

See also `interactive-pane`
 `listener-pane-insert-value`

interface *Class*

Summary The class `interface` is the top level window class, which contains both menus and a hierarchy of panes and layouts. Interfaces can also themselves be contained within a layout, in which case they appear without their menu bar.

Package `capi`

Superclasses	<code>simple-pane</code> <code>titled-object</code>	
Initargs	<code>:title</code>	The title of the interface.
	<code>:layout</code>	The layout of the interface.
	<code>:menu-bar-items</code>	The items on the menu bar.
	<code>:auto-menus</code>	A flag controlling the automatic addition of system menu objects.
	<code>:create-callback</code>	A callback done on creating the window.
	<code>:destroy-callback</code>	A callback done on closing the window.
	<code>:confirm-destroy-function</code>	A function to verify closing of the window.
	<code>:best-x</code>	The best x position for the interface.
	<code>:best-y</code>	The best y position for the interface.
	<code>:best-width</code>	The best width of the interface.
	<code>:best-height</code>	The best height of the interface.
	<code>:geometry-change-callback</code>	A function called when the interface geometry changes.
	<code>:activate-callback</code>	A function called when the interface is activated or deactivated.
	<code>:iconify-callback</code>	A function called when the interface is iconified or restored.

- :override-cursor**
A cursor that takes precedence over the cursors of panes inside the interface.
- :message-area** A boolean determining whether the interface has a message area.
- :enable-pointer-documentation**
A boolean determining whether Pointer Documentation is enabled.
- :enable-tooltips**
A boolean determining whether Tooltip Help is enabled.
- :help-callback**
A function called when a user gesture requests help.
- :top-level-hook**
A function called around the top level event handler.
- :external-border**
An integer or `nil`.
- :initial-focus**
A pane, a symbol naming a pane, or `nil`.
- :display-state**
One of the keywords `:normal`, `:maximized`, `:iconic` and `:hidden`.
- :transparency**
A real number in the inclusive range `[0,1]`, used on Cocoa and later versions of Microsoft Windows.
- :window-styles**
A list of keywords, or `nil`.

Accessors	<pre> interface-title pane-layout interface-menu-bar-items interface-create-callback interface-destroy-callback interface-confirm-destroy-function interface-geometry-change-callback interface-activate-callback interface-iconify-callback interface-override-cursor interface-message-area interface-pointer-documentation-enabled interface-tooltips-enabled interface-help-callback top-level-interface-external-border top-level-interface-transparency </pre>
Readers	<pre> interface-window-styles </pre>
Description	<p>Every interface can have a title <i>title</i> which when it is a top level interface is shown as a title on its window, and when it is contained within another layout is displayed as a decoration (see the class <code>titled-object</code> for more details).</p> <p>The argument <i>layout</i> specifies a layout object that contains the children of the interface. To change this layout you can either use the writer <code>pane-layout</code>, or you can use the layout <code>switchable-layout</code> which allows you to easily switch the currently visible child.</p> <p>The argument <i>menu-bar-items</i> specifies a list of menus to appear on the interface's menu bar.</p> <p><i>auto-menus</i> defaults to <code>t</code>, which means that an interface may have some automatic menus created by the environment in which it is running (for example the Works menu in the Common LispWorks environment). To switch these automatic menus off, pass <code>:auto-menus nil</code>.</p> <p>When you have an instance of an interface, you can display it either as an ordinary window or as a dialog using respectively <code>display</code> and <code>display-dialog</code>. The CAPI calls <i>create-callback</i> (if supplied) with the interface as its single argument,</p>

after all the widgets have been created but before the interface appears on screen. Then to remove the interface from the display, you use `quit-interface` and either `exit-dialog` or `abort-dialog` respectively. When the interface is about to be closed, the CAPI calls the *confirm-destroy-function* (if there is one) with the interface, and if this function returns non-`nil` the interface is closed. Once the interface is closed, the *destroy-callback* is called with the interface.

Note: as well as *create-callback*, you can also add code to run just before or just after displaying the interface as an ordinary window by adding appropriate methods on `interface-display`.

The interface also accepts a number of hints as to the size and position of the interface for when it is first displayed. The arguments *best-x* and *best-y* must be the position as an integer or `nil` (meaning anywhere), while the arguments *best-width* and *best-height* can be any hints accepted by `:visible-max-width` and `:visible-max-height` for elements.

Whether or not an interface window is resizable is indicated as allowed by the window system. For non-resizable windows on Cocoa the interface window's maximize button is disabled and the resize indicator is not shown, and on Microsoft Windows the maximize box is disabled.

geometry-change-callback may be `nil`, meaning there is no callback. This is the default value. Otherwise *geometry-change-callback* is a function of five arguments: the interface and the geometry. Its signature is:

`geometry-change-callback` *interface* *x* *y* *width* *height*

activate-callback may be `nil`, meaning there is no callback. This is the default value. Otherwise *activate-callback* is a function of two arguments: the interface and a boolean *activatep* which is true on activation and false on deactivation. Its signature is:

`activate-callback` *interface* *activatep*

iconify-callback may be `nil`, meaning there is no callback. This is the default value. Otherwise *iconify-callback* is a function of two arguments: the interface and a boolean *iconify* which is true when *interface* is iconified and false when it is restored. Its signature is:

`iconify-callback` *interface* *iconifyp*

override-cursor, if non-`nil`, specifies a cursor that is used instead of the cursor of each pane inside the interface. The default value of *override-cursor* is `nil`. See below for an example of setting and unsetting the override cursor.

If *message-area* is true, then the interface has a message area at the bottom. The text of the message area can be accessed using the `titled-object` accessor `titled-object-message`. The default value of *message-area* is `nil`.

enable-pointer-documentation is a boolean controlling whether Pointer Documentation is enabled. The default value is `t`. The actual action is done by the *help-callback*.

enable-tooltips is a boolean controlling whether Tooltip Help is enabled. The default value is `t`. The actual action is done by the *help-callback*.

help-callback may be `nil`, meaning there is no callback. This is the default value. Otherwise *help-callback* is a function of four arguments: the interface, the pane inside interface where help is requested, the type of help requested, and the help key of the pane. Its signature is:

`help-callback` *interface* *pane* *type* *help-key*

Here *type* can be one of:

`:pointer-documentation-enter`

The cursor entered the pane. The function should set the pointer documentation.

<code>:pointer-documentation-leave</code>	The cursor left the pane. The function needs to reset the pointer documentation.
<code>:tooltip</code>	A tooltip is requested. The function needs to return a string to display in the tooltip, or <code>nil</code> if no tooltip should be displayed.
<code>:help</code>	The function should display a detailed, asynchronous help. This value is passed when the user presses the <code>F1</code> key (not implemented on Cocoa). <code>:help</code> is also passed when the user clicks the '?' box in the title bar of a Microsoft Windows dialog with window style <code>:contexthelp</code> (see <i>window-styles</i> below).

help-key is the *help-key* of *pane*, as described in `element`. There is an example illustrating *help-callback* in `examples/capi/elements/help.lisp` and there is another example below.

top-level-hook can be used on Microsoft Windows and Motif to specify a hook function that is called around the interface's top level event handler. The hook is passed two arguments: a continuation function (with no arguments) and the interface. The hook must call the continuation, which normally does not return. *top-level-hook* is designed especially for error handling (see below for an example). It can also be used for other purposes, for instance to bind special variables around the top level function. `:top-level-hook` is not supported on Cocoa.

external-border controls how close to the edge of the screen the interface can be placed with explicit positioning using the *best-x*, *best-y*, *best-height* and *best-width* initargs or implicit positioning when a dialog is centered within its owner. The value `nil` allows the window to be anywhere, on or off the screen. The value `0` allows the window can be anywhere on the screen. If *external-border* is a positive integer then the

window can be anywhere within *external-border* pixels from the edge of the screen. If *external-border* is a negative integer then the window be anywhere on the screen or up to *external-border* pixels off the edge of the screen. This does not affect whether the user can move the window after it has been displayed. It also does not affect the default positioning of interfaces, where the window system chooses the position. The default value of *external-border* is 0.

initial-focus specifies a pane which has the input focus when the interface is first displayed. See `pane-initial-focus` for more information about the initial focus pane.

display-state controls the initial display of the interface window, as described for `top-level-interface-display-state`.

transparency is the overall transparency of the whole interface, where 0 is fully transparent and 1 is fully opaque. This has no effect on whether the user can click on the window. This is implemented for Cocoa and for Microsoft Windows, excluding Windows 98, Millennium Edition and NT 4.0.

`:transparency` should only be used for top-level interfaces.

window-styles is a list of keywords controlling various aspects of the top level window's appearance and behavior. Each keyword is supported only on the Window systems explicitly mentioned below.

The following keywords apply to ordinary windows:

`:no-geometry-animation`

Cocoa: Programmatic changes to window geometry happen without animation.

`:hides-on-deactivate-window`

Cocoa: The window is only visible when the application is the current application.

Microsoft Windows: The window is only visible when it is the active window.

:toolbox

Cocoa and Microsoft Windows: A window with a small title bar. This window style is used in `docking-layout`.

:borderless

Cocoa, Microsoft Windows and Motif: A window with no external decoration or frame.

:internal-borderless

Cocoa and Motif: Remove the default border between the window's edge and its contents.

:never-iconic

Cocoa, Microsoft Windows and Motif: The window cannot be minimized.

:movable-by-window-background

Cocoa and Microsoft Windows: The user can move the window by grabbing at any point not in an inner pane.

:shadowed

Cocoa: Force a shadow on windows with window style `:borderless`. (Other windows have a shadow by default.)

Windows XP (and later): The window has a shadow.

:shadowless

Cocoa: The window has no shadow.

:textured-background

Cocoa: The window has a textured background (like the Finder).

`:always-on-top`

Cocoa and Microsoft Windows: The window is always above all other windows. Such a window is also known as a windoid.

`:ignores-keyboard-input`

Cocoa: The window cannot be given the focus for keyboard input.

`:no-character-palette`

Cocoa: The **Special Characters...** menu item is not inserted automatically. (This menu item is added to the **Edit** menu by default.)

`:motion-events-without-focus`

Cocoa: `output-panes` in the window will see `:motion` input model events even if the output pane does not have the focus. This is the same behavior as on Microsoft Windows.

The following keywords are supported in *window-styles* when the interface is displayed as a dialog:

`:resizable`

Microsoft Windows: The dialog has a border to allow resizing. (Generally Windows dialogs do not allow resizing.)

`:contexthelp`

Microsoft Windows: A '?' box appears in the window's title bar that sends *help-callback* type `:help`.

Note: even though `interface` is a subclass of `titled-object`, the accessor `titled-object-message-font` cannot be used to get and set the font of the interface's message.

Compatibility
Note

`interface-iconize-callback` is deprecated. Use the synonym `interface-iconify-callback` instead.

Example

```

(capi:display (make-instance 'capi:interface
                             :title "Test Interface"))

(capi:display (make-instance
               'capi:interface
               :title "Test Interface"
               :destroy-callback
               #'(lambda (interface)
                   (capi:display-message
                    "Quitting ~S"
                    interface))))

(capi:display (make-instance
               'capi:interface
               :title "Test Interface"
               :confirm-destroy-function
               #'(lambda (interface)
                   (capi:confirm-yes-or-no
                    "Really quit ~S"
                    interface))))

(capi:display (make-instance
               'capi:interface
               :menu-bar-items
               (list
                (make-instance 'capi:menu
                               :title "Menu"
                               :items '(1 2 3)))
               :title "Menu Test"))

```

```

(setq interface
  (capi:display
    (make-instance
      'capi:interface
      :title "Test Interface"
      :layout
      (make-instance 'capi:simple-layout
        :description
        (list (make-instance
          'capi:text-input-pane
          :text "Text Pane"))))))))

(capi:execute-with-interface interface
  #'(setf capi:pane-layout) (make-instance
    'capi:simple-layout
    :description
    (list (make-instance
      'capi:editor-pane
      :text "Editor Pane"))))

interface)

(capi:display
  (make-instance
    'capi:interface
    :title "Test"
    :best-x 200
    :best-y 200
    :best-width '(/ :screen-width 2)
    :best-height 300))

```

The following forms illustrate the use of *help-callback*:

```

(capi:define-interface my-interface ()
  ()
  (:panes
   (a-pane
    capi:text-input-pane
    :help-key 'input)
   (another-pane
    capi:display-pane
    :help-key 'output
    :text "some text"))
  (:menu-bar a-menu)
  (:menus
   (A-menu
    "A menu"
    (("An item" :help-key "item 1")
     ("Another item" :help-key "item 2"))
    :help-key "a menu"))
  (:layouts
   (main-layout
    capi:column-layout
    '(a-pane another-pane)))

  (:default-initargs
   :help-callback 'my-help-callback
   :message-area t))

(defun do-detailed-help (interface)
  (capi:contain
   (make-instance
    'capi:display-pane
    :text "Detailed help for my interface")
   :title
   (format nil "Help for ~a"
            (capi:capi-object-name interface))))

(defun my-help-callback (interface pane type key)
  (declare (ignore pane))
  (case type
    (:tooltip (if (eq key 'input)
                  "enter something"
                  (when (stringp key)
                    (when (stringp key)
                      (setf (capi:titled-object-message interface)
                            key))))
    (:pointer-documentation-enter
     (when (stringp key)
       (setf (capi:titled-object-message interface)
             key)))
    (:pointer-documentation-leave
     (setf (capi:titled-object-message interface)
           key))))

```

```

        "Something else"))
      (:help (do-detailed-help interface ))))

(capi:display
 (make-instance 'my-interface :name "Helpful"))

```

The following forms illustrate the use of *override-cursor* to set and then remove an override cursor.

Create an interface with panes that have various different cursors. Move the pointer across each pane.

```

(setf interface
  (capi:element-interface
   (car
    (capi:contain
     (loop for cursor
          in '(:crosshair :hand :v-double-arrow)
          collect
           (make-instance 'capi:editor-pane
                         :cursor cursor
                         :text
                         (format nil "~A CURSOR"
                                cursor)))))))

```

Override the pane cursors by setting the override cursor on the interface, and move the pointer across each pane again.

```

(setf (capi:interface-override-cursor interface)
      :i-beam)

```

Remove the override cursor.

```

(setf (capi:interface-override-cursor interface)
      :default)

```

This example illustrates *top-level-hook*. Evaluate this form and then get an error by **Meta+Control+C** (on Motif) or **Control+Break** (on Microsoft Windows) in the editor pane. Then select the Destroy Interface restart.

```
(capi:display
(capi:make-container
(make-instance
'capi:editor-pane)
:top-level-hook
#'(lambda (func interface)
(restart-case (funcall func)
(nil ()
:report
(list "Destroy Interface ~a" interface)
(capi:destroy interface))))))
```

This example illustrates the use of `:create-callback`:

```
(defun get-children (self)
(let (children)
(capi:map-pane-descendant-children
self #'(lambda (x)
(push x children)))
(with-slots (lp) self
(setf (capi:collection-items lp) children))))

(defun get-children-data (x)
(list (class-name (class-of x))
(format nil "~X" (sys:object-address x))))

(capi:define-interface created-data () ()
(:panes
(title
capi:title-pane
:text "A list populated via :CREATE-CALLBACK")
(lp
capi:multi-column-list-panel
:visible-min-height '(:character 3)
:column-function 'get-children-data))
(:layouts
(main
capi:column-layout
'(title lp)))
(:default-initargs
:create-callback 'get-children
:title ":CREATE-CALLBACK Example Interface"
:width 300))

(capi:display (make-instance 'created-data))
```

See also `layout`
`switchable-layout`
`menu`
`display`
`display-dialog`
`interface-display`
`quit-interface`
`define-interface`
`activate-pane`
`titled-object`

interface-display

Generic Function

Summary The function called to display an interface on screen.

Package `capi`

Signature `interface-display` *interface*

Arguments *interface* An instance of a subclass of `interface`.

Description The generic function `interface-display` is called by `display` to display an interface on screen.

The primary method for `interface` actually does the work. You can add `:before` methods on your own interface classes for code that needs to be executed just before the interface appears, and `:after` methods for code that needs to be executed just after the interface appears.

Note: `interface-display` is called in the process of *interface*.

Note: `interface-display` is not called when *interface* is displayed as a dialog. Another way to run code before it appears on screen is to supply a *create-callback* for *interface*.

Examples This example shows how `interface-display` can be used to set the initial selection in a choice whose items are computed at display-time:

```
(capi:define-interface my-tree ()
  ((favorite-color :initform :blue))
  (:panes
   (tree
    capi:tree-view
    :roots '(:red :blue :green)
    :print-function
    'string-capitalize))
  (:default-initargs
   :width 200
   :height 200))

(defmethod capi:interface-display :after
  ((self my-tree))
  (with-slots (tree favorite-color) self
    (setf (capi:choice-selected-item tree)
          favorite-color)))

(capi:display (make-instance 'my-tree))
```

See also `display`
`interface`

interface-display-title

Function

Summary Returns the interface title to use on screen.

Package `capi`

Signature `interface-display-title interface => string`

Arguments `interface` A CAPI interface.

Values `string` A string.

Description The function `interface-display-title` returns the title to use when displaying the interface *interface* on screen.

This is equivalent to:

```
(capi:interface-extend-title
 interface
 (capi:interface-title interface))
```

See also `interface-extend-title`
 `set-default-interface-prefix-suffix`

interface-editor-pane

Generic Function

Summary Finds an `editor-pane` in an interface.

Package `capi`

Signature `interface-editor-pane interface => pane`

Arguments *interface* An instance of a subclass of `interface`.

Values *pane* An `editor-pane` or `nil`.

Description The generic function `interface-editor-pane` finds the first pane of `interface` that is an `editor-pane`, and returns it.

If there is no `editor-pane`, then `interface-editor-pane` returns `nil`.

See also `editor-pane`
 `interface`

interface-extend-title

Generic Function

Summary Calculates the complete interface title.

Package	<code>capi</code>
Signature	<code>interface-extend-title</code> <i>interface</i> <i>title</i> => <i>string</i>
Arguments	<i>interface</i> A CAPI interface. <i>title</i> A string.
Description	<p>The generic function <code>interface-extend-title</code> is called by the system with an interface and its title before actually displaying the title on the screen. The result must be a string, which is actually displayed. There is no requirement for any relation between the title argument and the result.</p> <p>The return value <i>string</i> is the title to display on the screen.</p> <p>The default method uses the values set by <code>set-default-interface-prefix-suffix</code>. You can specialize <code>interface-extend-title</code> to get other effects.</p>
See also	<code>interface-display-title</code> <code>set-default-interface-prefix-suffix</code>

interface-geometry*Generic Function*

Summary	Returns the geometry of an interface.
Package	<code>capi</code>
Signature	<code>interface-geometry</code> <i>interface</i> => <i>geometry</i>
Arguments	<i>interface</i> An instance of a subclass of <code>interface</code> .
Values	<i>geometry</i> A list.
Description	The generic function <code>interface-geometry</code> returns a list representing the geometry of interface in pixel values.

geometry is of the form (*x y width height*).

See also `interface`

interface-iconified-p

Function

Summary The predicate for whether an interface is iconified.

Package `capi`

Signature `interface-iconified-p pane => iconifiedp`

Arguments *pane* A CAPI element.

Values *iconifiedp* A boolean.

Description The function `interface-iconified-p` returns `t` if the top level interface containing *pane* is iconified. This means that the window is visible as an icon, also referred to as minimized.

If the top level interface is not iconified, then `interface-iconified-p` returns `nil`.

See also `hide-interface`
 `top-level-interface`
 `top-level-interface-display-state`

interface-keys-style

Generic Function

Summary Determines the emulation for an interface.

Package `capi`

Signature `interface-keys-style interface => keys-style`

Arguments	<i>interface</i>	An instance of a subclass of <i>interface</i> .
Values	<i>keys-style</i>	A keyword, <code>:pc</code> , <code>:emacs</code> or <code>:mac</code> .
Description		<p>The generic function <code>interface-keys-style</code> returns a keyword indicating a keys style, or <i>emulation</i>. It is called when <i>interface</i> starts running in a new process, and <i>keys-style</i> determines how user input is interpreted by output panes (including <code>editor-pane</code>) in <i>interface</i>.</p> <p>The editor (that is, instances of <code>editor-pane</code> and its subclasses) responds to user input gestures according to one of three basic models.</p> <p>When <i>keys-style</i> is <code>:emacs</code>, the editor emulates GNU Emacs. This value is allowed on all platforms.</p> <p>When <i>keys-style</i> is <code>:pc</code>, the editor emulates standard Microsoft Windows keys on Windows, and KDE/Gnome keys on Motif. This value is allowed in the Windows and the X11/Motif implementations.</p> <p>When <i>keys-style</i> is <code>:mac</code>, the editor emulates Mac OS X editor keys. This value is allowed only in the Mac OS X Cocoa implementation.</p> <p>The most important differences between the styles are in the handling of the <code>⌘</code> key on Microsoft Windows, selected text, and accelerators:</p> <p><code>:emacs</code> <code>⌘</code> is interpreted on Microsoft Windows as the Meta key (used to access many Emacs commands).</p> <p>The <code>:meta</code> modifier is used in an <code>output-pane input-model</code> gesture specification.</p> <p>Control characters such as <code>ctrl+s</code> are not interpreted as accelerators.</p> <p>The selection is not deleted on input.</p>

`:pc` `Alt` is interpreted as `Alt` on Microsoft Windows and can be used for shortcuts.

The `:meta` modifier is not used in an `output-pane` *input-model* gesture specification.

`Control` keystrokes are interpreted as accelerators. Standard accelerators are added for standard menu commands, for example `Ctrl+S` for **File > Save**.

The selection is deleted on input, and movement keys behave like a typical Microsoft Windows or KDE/Gnome editor.

`:mac` Emacs `control` keys are available, since they do not clash with the Macintosh `Command` key.

The selection is deleted on input, and movement keys behave like a typical Mac OS X editor.

By default *keys-style* is `:pc` on Microsoft Windows platforms and `:emacs` on Unix/Linux and Mac OS X platforms. You can supply methods for `interface-keys-style` on your own interface classes that override the default methods.

In the Cocoa implementation, `Command` keystrokes such as `Command+x` are available if there is a suitable **Edit** menu, regardless of the Editor emulation.

See the chapter "Emulation" in the *LispWorks Editor User Guide* for more detail about the different styles.

See also `editor-pane`

interface-match-p*Generic Function*

Summary	Determines whether an interface is suitable for displaying <i>initargs</i> .	
Package	<code>capi</code>	
Signature	<code>interface-match-p <i>interface</i> &rest <i>initargs</i> &key &allow-other-keys => <i>matchp</i></code>	
Arguments	<i>interface</i>	An instance of a subclass of <i>interface</i> .
	<i>initargs</i>	Initargs for <i>interface</i> .
Values	<i>matchp</i>	A boolean
Description	<p>The generic function <code>interface-match-p</code> returns a true value if <i>interface</i> is suitable for displaying the <i>initargs</i>.</p> <p><code>interface-match-p</code> is used by <code>locate-interface</code>. When there is an existing interface for which <code>interface-match-p</code> returns true, then <code>locate-interface</code> returns it.</p> <p>The default method for <code>interface-match-p</code> always returns <code>nil</code>. You can add methods for your own interface classes.</p>	
See also	<code>locate-interface</code>	

interface-menu-groups*Generic Function*

Summary	Used when an embedded document sets the <i>menu-bar-items</i> to its menus.	
Package	<code>capi</code>	
Signature	<code>interface-menu-groups <i>interface</i> => <i>result</i></code>	
Arguments	<i>interface</i>	A CAPI <i>interface</i> .

Values	<i>result</i> A list.
Description	<p>The generic function <code>interface-menu-groups</code> is called when an embedded document sets the menu bar of its containing interface.</p> <p>Then, the menu bar for the embedded document includes three groups of menus that are supplied by the container (<code>file-group</code>, <code>view-group</code>, <code>windows-group</code>). <code>interface-menu-groups</code> is used to define these groups of menus.</p> <p><code>interface-menu-groups</code> should return a list of length 3. Each element is a list of menus. In this list, each item is either a menu object, or a cons. When it is a cons, the car is a menu object and the cdr is a string, which overrides the the title of the menu.</p> <p>The default method, on interface, simply returns <code>(nil nil nil)</code>.</p> <p>Note: this function is implemented only in LispWorks for Windows. Load the functionality by <code>(require "embed")</code>.</p>
Example	See the example in <code>examples/com/ole/simple-container/doc-viewer-pair.lisp</code>
See also	<code>ole-control-pane</code>

interface-reuse-p

Generic Function

Summary	Determines whether an interface is suitable for re-use.
Package	<code>capi</code>
Signature	<code>interface-reuse-p</code> <i>interface</i> &rest <i>initargs</i> &key &allow-other-keys => <i>reusep</i>

Arguments	<i>interface</i>	An instance of a subclass of <i>interface</i> .
	<i>initargs</i>	Initargs for <i>interface</i> .
Values	<i>reusep</i>	A boolean
Description	<p>The generic function <code>interface-reuse-p</code> returns a true value if <i>interface</i> is suitable for reuse with <i>initargs</i>.</p> <p><code>interface-reuse-p</code> is used by <code>locate-interface</code> if no matching interface is found first by <code>interface-match-p</code>. In this case, when there is an interface for which <code>interface-reuse-p</code> returns true, then <code>locate-interface</code> reinitializes it by <code>reinitialize-interface</code> and returns it.</p> <p>Note: <code>interface-reuse-p</code> should not be confused with <code>reuse-interfaces-p</code>, which determines the global re-use state.</p>	
See also	<p><code>interface-match-p</code> <code>locate-interface</code></p>	

interface-visible-p

Function

Summary	The predicate for whether the interface containing a pane is visible.	
Package	<code>capi</code>	
Signature	<code>interface-visible-p pane => visiblep</code>	
Arguments	<i>pane</i>	A CAPI pane.
Values	<i>visiblep</i>	A boolean.
Description	<p>The function <code>interface-visible-p</code> returns <code>nil</code> if</p> <ol style="list-style-type: none"> <i>pane</i> is not associated with any interface, or 	

2. *pane* is associated with an interface which is not displayed, or
3. *pane* is associated with an interface which is minimized or iconified, or
4. *pane* is known to be fully obscured by other windows. This can happen on Motif, but is not detected on Microsoft Windows.

An error is signalled if *pane* is not a CAPI pane (that is, it is not an instance of a subclass of `element`, `collection` or `pin-board-object`).

Otherwise `interface-visible-p` returns `t`.

Note: On Microsoft Windows, `interface-visible-p` may return `t` even though the interface is entirely obscured by another window.

interpret-description

Generic Function

Summary	Converts an abstract description of a layout's children into a list of the children's geometry objects.
Package	<code>capi</code>
Signature	<code>interpret-description layout description interface</code>
Description	<p>The generic function <code>interpret-description</code> translates an abstract description of the <i>layout</i>'s children into a list of those children's geometry objects.</p> <p>For example, <code>column-layout</code> expects as its description a list of items where each item in the list is either the slot-name of the child or a string which should be turned into a title pane. This is the default handling of a layout's description, which is done by calling the generic function <code>parse-layout-descriptor</code> to do the translation for each item.</p>

Examples See the examples in the directory `examples/capi/layouts/`.

See also `parse-layout-descriptor`
`define-layout`
`layout`
`interface`

invalidate-pane-constraints

Function

Summary Causes the resizing of a pane if its minimum and maximum size constraints have changed. It returns `⊔` if resizing was necessary.

Package `capi`

Signature `invalidate-pane-constraints` *pane*

Description This function informs the CAPI that *pane*'s constraints (its minimum and maximum size) may have changed. The CAPI then checks this, and if the pane is no longer within its constraints it resizes it so that it is and then makes the pane's parent layout lay its children out and display them again at their new positions and sizes. If the pane is resized, then `invalidate-pane-constraints` returns `⊔`.

See also `get-constraints`
`layout`
`element`
`define-layout`

invoke-command

Function

Summary Invokes a command in the input model for a specified output pane.

item	<i>Class</i>	
Summary	The class <code>item</code> groups together a title, some data and some callbacks into a single object for use in collections and choices.	
Package	<code>capi</code>	
Superclasses	<code>callbacks</code> <code>capi-object</code>	
Subclasses	<code>menu-item</code> <code>button</code> <code>item-pinboard-object</code> <code>popup-menu-button</code> <code>toolbar-button</code>	
Initargs	<code>:collection</code>	The collection in which <code>item</code> is displayed
	<code>:data</code>	The data associated with the item.
	<code>:text</code>	The text to appear in the item (or <code>nil</code>).
	<code>:print-function</code>	If <code>text</code> is <code>nil</code> , this is called to print the data.
	<code>:selected</code>	If <code>t</code> the item is selected.
Accessors	<code>item-collection</code> <code>item-data</code> <code>item-text</code> <code>item-print-function</code> <code>item-selected</code>	
Description	An item can provide its own callbacks to override those specified in its enclosing <i>collection</i> , and can also provide some data to get passed to those callbacks.	

An item is printed in the collection by `print-collection-item`. By default this returns a string using *item's text* if specified, or else calls a print function on the item's *data*. The *print-function* will either be the one specified in the item, or else the *print-function* for its parent collection.

The *selected* slot in an item is non-`nil` if the item is currently selected. The accessor `item-selected` is provided to access and to set this value.

Example

```
(defun main-callback (data interface)
  (capi:display-message "Main callback: ~S"
                        data))

(defun item-callback (data interface)
  (capi:display-message "Item callback: ~S"
                        data))

(capi:contain (make-instance
              'capi:list-panel
              :items (list
                      (make-instance
                       'capi:item
                       :text "Item"
                       :data '(some data)
                       :selection-callback
                       'item-callback)
                      "Non-Item 1"
                      "Non-Item 2")
              :selection-callback 'main-callback))
```

See also

```
itemp
collection
choice
print-collection-item
```

itemp

Generic Function

Package

```
capi
```

Signature

```
itemp object
```

Description This is equivalent to
`(typep object 'capi:item)`

See also `item`
`collection`

item-pinboard-object

Class

Summary An `item-pinboard-object` is a `pinboard-object` that displays a single piece of text.

Package `capi`

Superclasses `pinboard-object`
`item`

Description The `item-pinboard-object` displays an item on a pinboard layout. It displays the text specified by the item in the usual way (either by the text field, or through printing the data with the print function).

Example

```
(capi:contain (make-instance
              'capi:item-pinboard-object
              :text "Hello World"))

(capi:contain (make-instance 'capi:item-pinboard-object
                             :data :red
                             :print-function
                             'string-capitalize))
```

See also `image-pinboard-object`
`pinboard-layout`

labelled-arrow-pinboard-object

Class

Package `capi`

Superclasses	<code>arrow-pinboard-object</code> <code>labelled-line-pinboard-object</code>
Description	A subclass of <code>pinboard-object</code> which displays an arrow and draws a label on it.
Example	See <code>labelled-line-pinboard-object</code> .
See also	<code>pinboard-layout</code>

labelled-line-pinboard-object

Class

Summary	A subclass of <code>pinboard-object</code> which draws a labelled line.
Package	<code>capi</code>
Superclasses	<code>item-pinboard-object</code> <code>line-pinboard-object</code>
Subclasses	<code>labelled-arrow-pinboard-object</code>
Initargs	<code>:text-foreground</code> The color of the label text.
Accessors	<code>labelled-line-text-foreground</code>
Description	A subclass of <code>pinboard-object</code> which displays a line and draws a label in the middle of it. Note that the label text is inherited from <code>item</code> .

Example

```
(capi:contain
 (make-instance
  'capi:pinboard-layout
  :description
  (list (make-instance
        'capi:labelled-line-pinboard-object
        :text "Labelled Line"
        :start-x 10 :start-y 10
        :end-x 80 :end-y 60)
        (make-instance
        'capi:labelled-arrow-pinboard-object
        :text "Labelled Arrow"
        :start-x 10 :start-y 70
        :end-x 80 :end-y 120
        :head-direction :both))))
```

See also `pinboard-layout`

layout

Class

Summary A `layout` is a simple pane that positions one or more child panes within itself according to a layout policy.

Package `capi`

Superclasses `titled-object`
`simple-pane`

Subclasses `simple-layout`
`grid-layout`
`pinboard-layout`
`switchable-layout`

Initargs `:default` A flag to mark the default layout for an interface.

`:description` The list of the layout's children.

`:initial-focus`

A child of the layout, or its *name*, specifying where the input focus should be, or `nil`.

Accessors `layout-description`

Description The layout's *description* is an abstract description of the children of the layout, and each layout defines its format. Generally, *description* is a list, each element of which is one of:

- a pane
- a slot name, where the name refers to a slot in the layout's interface containing a pane
- a string, where the string gets converted to a `title-pane`

For `grid-layout` and its subclasses, elements of *description* can also be `nil`. See `grid-layout` for the interpretation of this value.

Setting the layout description causes the layout to translate it, and then to layout the new children, adjusting the size of its parent if necessary.

A number of default layouts are provided which provide the majority of layout functionality that is needed. They are as follows:

`simple-layout` A layout for one child.

`row-layout` Lays its children out in a row.

`column-layout` Lays its children out in a column.

`grid-layout` Lays its children out in an n by m grid.

`pinboard-layout`

Places its children where the user specifies.

`switchable-layout`

Keeps only one of its children visible.

initial-focus specifies which child of the layout has the input focus when the layout is first displayed. Panes are compared by `eq` or `cap-object-name`.

Note: for a `pinboard-layout`, the order of the objects in *description* defines the Z-order, with the first object in the list being at the bottom. That is,

```
(setf (capi:layout-description pinboard-layout)
      (cons object
            (capi:layout-description pinboard-layout)))
```

is equivalent to

```
(capi:manipulate-pinboard pinboard-layout object
                          :add-bottom)
```

See also `define-layout`
`manipulate-pinboard`

line-pinboard-object

Class

Summary A subclass of `pinboard-object` which displays a line drawn between two corners of the area enclosed by the pinboard object.

Package `capi`

Superclasses `pinboard-object`

Subclasses `arrow-pinboard-object`
`right-angle-line-pinboard-object`

Initargs

<code>:start-x</code>	The x coordinate of the start of the line.
<code>:start-y</code>	The y coordinate of the start of the line.
<code>:end-x</code>	The x coordinate of the end of the line.
<code>:end-y</code>	The y coordinate of the end of the line.

Description `start-x`, `start-y`, `end-x` and `end-y` default to values computed from the `x`, `y`, `width` and `height`. They are used to compute the size of the object, and the proper value of `x` and `y`. Note that

width and *height* may be larger, for example to accommodate the label in a `labelled-line-pinboard-object`, and the *x* and *y* are adjusted for that.

To change the end points of the line, call `move-line`.

A complementary class `right-angle-line-pinboard-object` is provided which draws a line around the edge of the pinboard object.

Example

```
(capi:contain
 (make-instance
  'capi:line-pinboard-object
  :start-x 0 :end-x 100
  :start-y 100 :end-y 0))
```

See also

```
move-line
pinboard-layout
```

line-pinboard-object-coordinates

Function

Summary	Returns the coordinates of a <code>line-pinboard-object</code> .	
Package	<code>capi</code>	
Signature	<code>line-pinboard-object-coordinates object => start-x, start-y, end-x, end-y</code>	
Arguments	<i>object</i>	A <code>line-pinboard-object</code> .
Values	<i>start-x</i>	An integer.
	<i>start-y</i>	An integer.
	<i>end-x</i>	An integer.
	<i>end-y</i>	An integer.

The `list-panel` class does not support the `:no-selection` interaction style. For a non-interactive list use a `display-pane`.

To scroll a `list-panel`, call `scroll` with `scroll-operation :move`. `mnemonic-title` is interpreted as for `menu`.

`right-click-selection-behavior` can take the following values:

`nil` Corresponds to the behavior in LispWorks 4.4 and earlier. The data is not passed.

All non-`nil` values pass the clicked item as data to the pane menu:

`:existing-or-clicked/restore/discard`

If the clicked item is not already selected, make it be the entire selection while the menu is displayed. If the clicked item is already selected, do not change the selection. If the menu is cancelled, the original selection is restored. If the user chooses an item from the menu, the selection is not restored.

`:temporary-selection`

A synonym for `:existing-or-clicked/restore/discard`.

`:existing-or-clicked/restore/restore`

If the clicked item is not already selected, make it be the entire selection while the menu is displayed. If the clicked item is already selected, do not change the selection. If the user chooses an item from the menu and the item's callback does not set the selection then the original selection is restored after the callback. If the callback

sets the selection, then this selection remains. The original selection is restored if the user cancels the menu.

:temporary-restore

A synonym for **:existing-or-clicked/restore/restore**.

:clicked/restore/discard

Make the clicked item be the entire selection while the menu is displayed. If the menu is cancelled, the original selection is restored. If the user chooses an item from the menu, the selection is not restored.

:temporary-always

A synonym for **:clicked/restore/discard**.

:clicked/restore/restore

Make the clicked item be the entire selection while the menu is displayed. If the user chooses an item from the menu and the item's callback does not set the selection then the original selection is restored after the callback. If the callback sets the selection, then this selection remains. The original selection is restored if the user cancels the menu.

:existing-or-clicked/discard/discard

If the clicked item is not already selected, make it be the entire selection while the menu is displayed. If the clicked item is already selected, do not change the selection. The original selection is never restored, regardless of whether the user chooses an item from the menu or cancels the menu.

:discard-selection

A synonym for **:existing-or-clicked/discard/discard**.

:clicked/discard/discard

Make the clicked item be the entire selection. The original selection is never restored, regardless of whether the user chooses an item from the menu or cancels the menu.

:discard-always

A synonym for **:clicked/discard/discard**.

:no-change

Does not affect the selection, but the clicked item is nonetheless passed as the data.

The default value of *right-click-selection-behavior* is

:no-change.

color-function allows you to control the text colors on Microsoft Windows. If *color-function* is non-`nil`, then it is a function used to compute the text color of each item, with signature

`color-function list-panel item state => result`

state must be a keyword representing the state of the item. It can be one of `:normal`, `:selected` or `:disabled`. The value *result* should be a value suitable for the function `convert-color`. The pane uses the converted color as the foreground color for the item *item*. *color-function* is called while *list-panel* is being drawn, so it should not do heavyweight computations.

Example

```

(setq list (capi:contain
            (make-instance 'capi:list-panel
                          :items '(:red :blue :green)
                          :selected-item :blue
                          :print-function
                          'string-capitalize)))

(capi:apply-in-pane-process
 list #'(setf capi:choice-selected-item) :red list)

(capi:apply-in-pane-process
 list #'(setf capi:choice-selected-item) :green list)

(capi:contain (make-instance
              'capi:list-panel
              :items '(:red :blue :green)
              :print-function 'string-capitalize
              :selection-callback
              #'(lambda (data interface)
                  (capi:display-message
                   "~S" data))))

```

This example illustrates the use of `:right-click-selection-behavior`:

```

(capi:define-interface click ()
  ((keyword :initarg :right-click-selection-behavior))
  (:panes
   (list-panel
    capi:list-panel
    :items '("foo" "bar" "baz" "quux")
    :visible-min-height '(:character 4)
    :pane-menu 'my-menu
    :interaction :multiple-selection
    :right-click-selection-behavior keyword)))

(defun my-menu (pane data x y)
  (declare (ignore pane x y))
  (make-instance 'capi:menu
                 :items (list "Hi There"
                              ""
                              "Here's the data:"
                              data)))

(capi:display
 (make-instance 'click
                :right-click-selection-behavior
                :clicked/restore/restore))

```

See also the example in `examples/capi/choice/list-pane-pane-menu.lisp`.

There are further examples in the directory `examples/capi/choice/`.

See also `button-panel`

list-panel-enabled

Generic Function

Summary Gets or sets the enabled state of a `list-panel`.

Package `capi`

Signature `list-panel-enabled list-panel => enabledp`

Signature `(setf list-panel-enabled) enabledp list-panel => enabledp`

Arguments	<i>list-panel</i>	A <code>list-panel</code> .
Values	<i>enabledp</i>	A boolean.
Description	<p>The generic function <code>list-panel-enabled</code> determines whether <code>list-panel</code> is currently enabled. It is equivalent to the <code>simple-pane</code> accessor <code>simple-pane-enabled</code>.</p> <p>The generic function <code>(setf list-panel-enabled)</code> enables <i>list-panel</i> when <i>enabledp</i> is true, and disables it otherwise. It is equivalent to <code>(setf simple-pane-enabled)</code>.</p>	
See also	<code>simple-pane</code>	

list-view

Class

Summary	<p>The list view pane is a choice that displays its items as icons and text in a number of formats.</p> <p>Note: <code>list-view</code> is not implemented on Cocoa</p>	
Package	<code>capi</code>	
Superclasses	<code>list-panel</code>	
Initargs	<code>:view</code>	Specifies which view the list view pane shows. The default is <code>:icon</code> .
	<code>:subitem-function</code>	Returns additional information to be displayed in report view.
	<code>:subitem-print-functions</code>	Used in report view to print the additional information.
	<code>:image-function</code>	Returns an image for an item

:state-image-function
Returns a state image for an item.

:image-lists
A plist of keywords and `image-list` objects.

:columns
Defines the columns used in report view

:auto-reset-column-widths
Determines whether columns automatically
resize. Defaults to `:all`.

:use-large-images
Indicates whether large icons will be used
(generally only if the icon view will be
used). Defaults to `t`.

:use-small-images
Indicates whether small icons will be used.
Defaults to `t`.

:use-state-images
Indicates whether state images will be used.
Defaults to `nil`.

:large-image-width
Width of a large image. Defaults to 32.

:large-image-height
Height of a large image. Defaults to 32.

:small-image-width
Width of a small image. Defaults to 16.

:small-image-height
Height of a small image. Defaults to 16.

`:state-image-width`

Width of a state image. Defaults to *small-image-width*.

`:state-image-height`

Height of a state image. Defaults to *small-image-height*.

Accessors

`list-view-view`
`list-view-subitem-function`
`list-view-subitem-print-functions`
`list-view-image-function`
`list-view-state-image-function`
`list-view-columns`
`list-view-auto-reset-column-widths`

Description

The list view inherits its functionality from `choice`. In many ways it may be regarded as a kind of enhanced list panel, although its behavior is not identical. It supports single selection and extended selection interactions.

The list view displays its items in one of four ways, determined by the value in the *view* slot. An application may use the list view pane in just a single view, or may change the view between all four available views using `(setf list-view-view)`.

See the notes below on using both large and small icon views.

In all views, the text associated with the item (the label) is returned by the *print-function*, as with any other choice.

- The icon view — `:icon`
 In this view, large icons are displayed, together with their label, positioned in the space available.
- The small icon view — `:small-icon`
 In this view, small icons are displayed, together with their label, positioned in the space available.

- The list view — `:list`
In this view, small icons are displayed, arranged in vertical columns.
- The report view — `:report`
In this view, multiple columns are displayed. A small icon and the item's label is displayed in the first column. Additional pieces of information, known as subitems, are displayed in subsequent columns.
To use the view `:report`, *columns* must specify a list of column specifiers. Each column specifier is a `plist`, in which the following keywords are valid:

<code>:title</code>	The column heading
<code>:width</code>	The width of the column in pixels. If this keyword is omitted or has the value <code>nil</code> , the width of the column is automatically calculated, based on the widest item to be displayed in that column.
<code>:align</code>	May be <code>:left</code> , <code>:right</code> or <code>:center</code> to indicate how items should be aligned in this column. The default is <code>:left</code> . Only left alignment is available for the first column.

The *subitem-function* is called on the item to return subitem objects that represent the additional information to be displayed in the subsequent columns. Hence, *subitem-function* should normally return a list, whose length is one less than the number of columns specified. Each subitem is then printed in its column using the appropriate subitem print function. *subitem-print-function* may be either a single print function, to be used for all subitems, or a list of functions: one for each subitem column.

Note that the first column always contains the item label, as determined by the *choice-print-function*.

The *image-function* is called on an item to return an image associated with the item. It can return one of the following:

A pathname or string

This specifies the filename of a file suitable for loading with `load-image`. Currently this must be a bitmap file.

A symbol

The symbol must have been previously registered by means of a call to `register-image-translation`.

An image object

As returned by `load-image`.

An image locator object

Allowing a single bitmap to be created which contains several button images side by side. See `make-image-locator` for more information. On Microsoft Windows, this also allows access to bitmaps stored as resources in a DLL.

An integer

This is a zero-based index into the list view's image list. This is generally only useful if the image list is created explicitly. See `image-list` for more details.

The *state-image-function* is called on an item to determine the state image, an additional optional image used to indicate the state of an item. It can return one of the above, or `nil` to indicate that there is no state image. State images may be used in any view, but are typically used in the report and list views.

If *image-lists* is supplied, it should be a plist containing the following keywords as keys. The corresponding values should be `image-list` objects.

<code>:normal</code>	Specifies an image-list object that contains the large item images. The <i>image-function</i> should return a numeric index into this image-list.
<code>:small</code>	Specifies an image-list object that contains the small item images. The <i>image-function</i> should return a numeric index into this image-list.
<code>:state</code>	Specifies an image-list object that contains the state images. The <i>state-image-function</i> should return a numeric index into this image-list

If both the large icon view (icon view) and one or more of the small icon views (small icon view, list view, report view) are to be used, special considerations apply.

The image lists must be created explicitly, using the `:image-lists` initarg, and the *image-function* must return an integer. Care must be taken to ensure that corresponding images in the `:normal` and `:small` image lists have the same numeric index.

Returning pathnames, strings or image-locators from the image function cause the CAPI to create the image-lists automatically; however, if large and small icon views are mixed, this will lead to incorrect icons (or no icons) being displayed in one or other view.

Note: `list-view` is not implemented on Cocoa.

Note: for some applications `multi-column-list-panel` will suffice instead of `list-view`.

See also

`image-list`
`list-panel`
`make-image-locator`
`multi-column-list-panel`

listener-pane*Class*

Package	<code>capi</code>
Superclasses	<code>interactive-pane</code>
Description	A listener pane is an editor pane that accepts Lisp forms, entered by the user at a prompt, which it then evaluates. All of the output that is sent to <i>*standard-output*</i> is sent to the listener, and finally the results of the evaluation are displayed.
Example	<pre>(capi:contain (make-instance 'capi:listener-pane) :best-width 300 :best-height 200)</pre>
See also	<code>collector-pane</code> <code>interactive-pane</code>

listener-pane-insert-value*Function*

Summary	Evaluates a form and inserts the result in a <code>listener-pane</code> .
Package	<code>capi</code>
Signature	<code>listener-pane-insert-value</code> <i>pane form</i>
Arguments	<i>pane</i> A <code>listener-pane</code> . <i>form</i> A Lisp form.
Description	The function <code>listener-pane-insert-value</code> evaluates the form <i>form</i> and inserts the result in the <code>listener-pane</code> <i>pane</i> , as if it resulted from user input. The result is printed, and the values of the history variables <code>*</code> , <code>**</code> , <code>***</code> , <code>/</code> , <code>//</code> , and <code>///</code> are set. <code>listener-pane-insert-value</code> may be called in any process.

Multiple values in the result of evaluating form are not supported: the first value only is inserted in *pane*

See also `interactive-pane-execute-command`

load-cursor

Function

Summary Loads a cursor.

Package `capi`

Signature `load-cursor filename-or-list => cursor`

Arguments *filename-or-list* A string or a list.

Values *cursor* A cursor object.

Description The function `load-cursor` loads a cursor from your cursor file, or loads a built-in cursor. It returns a cursor object which can be supplied as the value of the `simple-pane :cursor` initarg.

The cursor object can also be set with `(setf simple-pane-cursor)` to change a pane's cursor. This must be done in the process of the pane's interface.

If *filename-or-list* is a string, then it names a file which should be in a suitable format for the platform. On Microsoft Windows, cursor files must be in `.cur` or `.ani` format. On Cocoa, cursor files should be in TIFF format. The file is loaded at the time `load-cursor` is called, so the cursor object does not require the file at the time the cursor is displayed. The cursor object survives saving and delivering the image.

If *filename-or-list* is a list then it names a file or a built-in cursor to be loaded for a particular library, optionally together with arguments to be passed to the library. It should be of the form:

```
( (libname_1 filename_1 arg_1a arg_1b ...)
  (libname_2 filename_2 arg_2a arg_2b ...)
  ...
)
```

where *libname_n* is a keyword naming a supported library (see `default-library` for the values) and *filename_n* is either a string naming the cursor file to load for this library or a keyword naming one of the built-in cursors. *arg_na*, *arg_nb* and so on are library specific arguments. Currently, they are not used on Microsoft Windows, but hotspot arguments are supported on Cocoa as in the example below.

Examples

This example loads a standard Microsoft Windows cursor file:

```
(setq cur1 (capi:load-cursor "arrow_1"))
```

This example loads a standard Windows cursor file, and on Motif uses one of the built-in cursors:

```
(setq cur2
  (capi:load-cursor '(:win32 "3dwms")
    (:motif :v-double-arrow)))
```

This example loads a horizontal double-arrow on Windows, and a vertical double-arrow on Motif:

```
(setq cur3
  (capi:load-cursor '(:win32 :h-double-arrow)
    (:motif :v-double-arrow)))
```

This example loads a custom .cur file:

```
(setq cur4
  (capi:load-cursor
    "C:/Temp/Animated_Cursors/1a.cur"))
```

In this extended example, firstly we load a custom cursor for two platforms:

```
(setq cur
  (capi:load-cursor
    '(:win32
      "c:/WINNT40/Cursors/O_CROSS.CUR")
    (:cocoa
      "/Applications/iPhoto.app/Contents/Resources/retouch-
      cursor.tif"
      :x-hot 2
      :y-hot 2))))
```

Now we display a pane with the custom cursor loaded above:

```
(setq oo
  (capi:contain
    (make-instance
      'capi:output-pane
      :cursor cur
      :input-model
      `(((button-1 :press)
        ,(lambda (&rest x)
            (print x)))))))
```

We can remove the custom cursor:

```
(capi:apply-in-pane-process
  oo
  (lambda ()
    (setf (capi:simple-pane-cursor oo)
          :default)))
```

And we can restore the custom cursor:

```
(capi:apply-in-pane-process
  oo
  (lambda ()
    (setf (capi:simple-pane-cursor oo)
          cur)))
```

See also `simple-pane`

load-sound

Function

Summary Converts data to a loaded sound object.

Package	<code>capi</code>
Signature	<code>load-sound source &key owner => sound</code>
Arguments	<p><i>source</i> A pathname designator or an array returned by <code>read-sound-file</code>.</p> <p><i>owner</i> A CAPI <code>interface</code>, or <code>nil</code>.</p>
Values	<i>sound</i> An array of element type (<code>unsigned-byte 8</code>).
Description	<p>The function <code>load-sound</code> converts <i>source</i> into a loaded sound which can be played by <code>play-sound</code>.</p> <p><i>source</i> can be a pathname designator or an array returned by <code>read-sound-file</code>.</p> <p><i>owner</i> should be a CAPI <code>interface</code> object, or <code>nil</code> which means that the sound's owner is the current top level interface.</p> <p>The loaded sound <i>sound</i> will be unloaded (freed) automatically when its owner is destroyed. To create a sound that is never unloaded, pass the <code>screen</code> as the argument <i>owner</i>.</p>
See also	<p><code>free-sound</code></p> <p><code>play-sound</code></p> <p><code>read-sound-file</code></p>

locate-interface

Generic Function

Summary	Finds an interface of a given class that matches supplied <code>initargs</code> .
Package	<code>capi</code>

See also `collect-interfaces`
`interface-match-p`
`interface-reuse-p`
`reuse-interfaces-p`

lower-interface

Function

Summary The `lower-interface` function pushes the window containing a specified pane to the back of the screen.

Package `capi`

Signature `lower-interface pane`

Description This pushes the window containing *pane* to the back of the screen. To bring it back use `raise-interface`, and to iconify it use `hide-interface`.

See also `hide-interface`
`interface`
`lower-interface`
`raise-interface`
`quit-interface`

make-container

Generic Function

Summary The generic function `make-container` creates a container for a specified element.

Package `capi`

Signature `make-container element &rest interface-args`

Description This creates a container for *element* such that calling `display` on it will produce a window containing *element* on the screen. It will produce a container for any of the following classes of object:

```
simple-pane  
layout  
interface  
pinboard-object  
menu  
menu-item  
menu-component  
list
```

In the case of a `list`, the CAPI tries to see what sort of objects they are and makes an appropriate container. For instance, if they were all simple panes it would put them into a column layout.

The arguments *interface-args* will be passed through to the `make-instance` of the top-level interface, assuming that pane is not a top-level interface itself.

The complementary function `contain` uses `make-container` to create a container for an element which it then displays.

Example

```
(capi:display (capi:make-container  
              (make-instance  
                'capi:text-input-pane)))
```

See also

```
contain  
display  
interface  
element
```

make-docking-layout-controller

Function

Package `capi`

Signature `make-docking-layout-controller => controller`

Values	<i>controller</i>	A docking layout controller.
Description	<p>The function <code>make-docking-layout-controller</code> returns a docking layout controller object for use as the <i>controller</i> initarg in <code>docking-layout</code>.</p> <p>Layouts which share a docking layout controller are known as a Docking Group. See <code>docking-layout</code> for information about Docking Groups.</p>	
See also	<code>docking-layout</code>	

make-foreign-owned-interface*Function*

Summary	Creates a dummy interface which allows another application's window to be the owner of a CAPI dialog.	
Package	<code>capi</code>	
Signature	<code>make-foreign-owned-interface &key <i>handle name</i> => <i>interface</i></code>	
Arguments	<i>handle</i>	A Microsoft Windows hwnd.
	<i>name</i>	A string naming <i>interface</i> .
Values	<i>interface</i>	An instance of <code>foreign-owned-interface</code> .
Description	<p>The function <code>make-foreign-owned-interface</code> creates an instance of <code>foreign-owned-interface</code>. <i>interface</i> can be used as the <i>owner</i> argument when displaying a dialog. For information about dialog owners, see the "Prompting for Input" chapter in the <i>LispWorks CAPI User Guide</i>.</p> <p><i>handle</i> must be supplied and is the window handle (Windows hwnd) of a window in some application. For a CAPI window this window handle can be obtained by <code>simple-pane-handle</code>. For non-CAPI applications, the method of finding the</p>	

window handle will depend on the language and the way windows are represented, so you should consult the appropriate documentation.

name becomes the name of *interface*, and has no other meaning.

`make-foreign-owned-interface` is implemented only on Microsoft Windows.

Example

This example shows how a CAPI window can be the owner of a dialog in another LispWorks image.

Start LispWorks for Windows.

1. In the Listener, do **Tools > Interface > Listen**. This puts the Listener interface in the value of `*`.
2. In the Listener enter `(capi:simple-pane-handle *)`. The returned value is the window handle, it should be an integer. Denote this value by *hwnd*.

Start another LispWorks for Windows image (do not quit the first image). In the Listener of this second LispWorks image:

1. Enter `(setq foi (capi:make-foreign-owned-interface :handle hwnd))`.
2. Enter `(capi:prompt-for-color "Color?" :owner foi)`.

Now note that the Color dialog is owned by the Listener of the first LispWorks image.

make-general-image-set

Function

Summary	Creates an <code>image-set</code> object.
Package	<code>capi</code>
Signature	<code>make-general-image-set &key image-count width height id => image-set</code>

Arguments	<i>image-count</i>	An integer.
	<i>width</i>	An integer or <code>nil</code> .
	<i>height</i>	An integer or <code>nil</code> .
	<i>id</i>	A pathname, string or symbol.
Values	<i>image-set</i>	An <code>image-set</code> object.
Description	<p>The <code>make-general-image-set</code> function creates an <code>image-set</code> object that refers to an image or a file containing an image.</p> <p><i>id</i> is a pathname or string identifying an image file, or a symbol previously registered with <code>register-image-translation</code>.</p> <p><i>width</i> and <i>height</i> are the dimensions of a single sub-image within the main image, and <i>image-count</i> specifies the number of sub-images in the image.</p>	
Examples	<p>See the files</p> <pre>examples/capi/choice/tree-view.lisp examples/capi/choice/extended-selection-tree-view.lisp examples/capi/elements/toolbar.lisp</pre>	
See also	<pre>image-set make-resource-image-set</pre>	

make-icon-resource-image-set*Function*

Summary	Constructs an image set object identifying a icon resource in a DLL.	
Package	<code>capi</code>	
Signature	<pre>make-icon-resource-image-set &key <i>image-count width height</i> <i>library id</i> => <i>image-set</i></pre>	

Arguments	<i>image-count</i>	An integer.
	<i>width</i>	An integer.
	<i>height</i>	An integer.
	<i>library</i>	A string.
	<i>id</i>	A string or an integer.
Values	<i>image-set</i>	An <i>image-set</i> object.
Description	<p>The <i>make-icon-resource-image-set</i> function is only available in LispWorks for Windows. It constructs an image set object that identifies an image stored as a icon resource in a DLL.</p> <p><i>width</i> and <i>height</i> are the dimensions of a single sub-image within the main image, and <i>image-count</i> specifies the number of sub-images in the image.</p> <p><i>library</i> should be a string specifying the name of the DLL.</p> <p><i>id</i> should be either an integer which is the resource identifier of the icon, or a string naming the icon resource.</p>	
See also	<p><i>image-set</i> <i>make-general-image-set</i></p>	

make-image-locator

Function

Summary	Creates an image locator object to use with toolbars, list views and tree views.
Package	<i>capi</i>
Signature	<i>make-image-locator</i> &key <i>image-set index</i>

If *component-p* is true, then `make-menu-for-pane` creates a `menu-component` rather than a `menu`. The default value of *component-p* is `nil`.

menu is set up so that by default each callback inside it is done on the pane *pane* itself. This is the useful feature of `make-menu-for-pane` because it avoids the need to set up items to do their callbacks on *pane* explicitly.

Note that this is merely the default behavior. You can specify different callback behavior on a per-item basis, using *setup-callback-argument* and *callback-data-function* (see `menu-object`), *callback-type* (see `callbacks`) and *data* for `menu-item` (see `item`).

See also `make-pane-popup-menu`
`pane-popup-menu-items`

make-pane-popup-menu

Generic Function

Summary	Generates a popup <code>menu</code> or <code>menu-component</code> .	
Package	<code>capi</code>	
Signature	<code>make-pane-popup-menu</code> <i>pane interface</i> <i>&key title menu-name component-p => menu</i>	
Arguments	<i>pane</i>	A pane in an interface.
	<i>interface</i>	An interface or <code>nil</code> .
	<i>title</i>	A string or <code>nil</code> .
	<i>menu-name</i>	A string or <code>nil</code> .
	<i>component-p</i>	A boolean.
Values	<i>menu</i>	A <code>menu</code> or a <code>menu-component</code> .

Description	<p>The generic function <code>make-pane-popup-menu</code> generates a popup menu for <i>pane</i>.</p> <p><i>interface</i> can be <code>nil</code> if <i>pane</i> has already been created, in which case the <i>interface</i> of <i>pane</i> is used (obtained by the <code>element</code> accessor <code>element-interface</code>).</p> <p><i>title</i> and <i>menu-name</i> provide a title and name for <i>menu</i>. <i>title</i> and <i>menu-name</i> both default to <code>nil</code>.</p> <p>If <i>component-p</i> is true, then <code>make-pane-popup-menu</code> creates a <code>menu-component</code> rather than a menu. The default value of <i>component-p</i> is <code>nil</code>.</p>
Example	<p>This code makes an interface with two <code>graph-panes</code>. The <code>initialize-instance</code> method uses <code>make-pane-popup-menu</code> to add a menu to the menu bar from which the user can perform operations on the graphs.</p> <p>Note that, because <code>make-pane-popup-menu</code> calls <code>make-menu-for-pane</code> to make each menu, the callbacks in the menus are automatically done on the appropriate graph.</p>

```

(capi:define-interface gg ()
  ()
  (:panes
   (g1 capi:graph-pane)
   (g2 capi:graph-pane))
  (:layouts
   (main-layout capi:column-layout '(g1 g2)))
  (:menu-bar)
  (:default-initargs
   :visible-min-width 200
   :visible-min-height 300))

(defmethod initialize-instance :after ((self gg)
                                       &key)

  (with-slots (g1 g2) self
    (setf
     (capi:interface-menu-bar-items self)
     (append
      (capi:interface-menu-bar-items self)
      (list
       (make-instance
        'capi:menu
        :title "Graphs"
        :items
        (list
         (capi:make-pane-popup-menu
          g1 self :title "graph1")

         (capi:make-pane-popup-menu
          g2 self :title "graph2"))))))))

(capi:display (make-instance 'gg))

```

See also `make-menu-for-pane`

make-resource-image-set

Function

Summary Constructs an image set object identifying a bitmap resource in a DLL.

Package `capi`

Signature	<code>make-resource-image-set</code> &key <i>image-count</i> <i>width</i> <i>height</i> <i>library</i> <i>id</i> => <i>image-set</i>	
Arguments	<i>image-count</i>	An integer.
	<i>width</i>	An integer.
	<i>height</i>	An integer.
	<i>library</i>	A string.
	<i>id</i>	A string or an integer.
Values	<i>image-set</i>	An <i>image-set</i> object.
Description	<p>The <code>make-resource-image-set</code> function is only available in LispWorks for Windows. It constructs an image set object that identifies an image stored as a bitmap resource in a DLL.</p> <p><i>width</i> and <i>height</i> are the dimensions of a single sub-image within the main image, and <i>image-count</i> specifies the number of sub-images in the image.</p> <p><i>library</i> should be a string specifying the name of the DLL.</p> <p><i>id</i> should be either an integer which is the resource identifier of the bitmap, or a string naming the bitmap resource.</p>	
See also	<p><code>image-set</code> <code>make-icon-resource-image-set</code> <code>make-general-image-set</code></p>	

make-scaled-general-image-set*Function*

Summary	Constructs an image set object which scales images in another image set.	
Package	<code>capi</code>	
Signature	<code>make-scaled-general-image-set</code> &key <i>width</i> <i>height</i> <i>id</i> => <i>image-set</i>	

Arguments	<i>width</i>	An integer.
	<i>height</i>	An integer.
	<i>id</i>	A pathname, string or symbol.
Values	<i>image-set</i>	An <code>image-set</code> object.
Description	<p>The <code>make-scaled-general-image-set</code> function is only available in LispWorks for Windows. It constructs an image set that provides scaled images based on an <code>image-set</code> object constructed from <i>id</i> as if by <code>make-general-image-set</code>.</p> <p><i>width</i> and <i>height</i> are the dimensions of a single sub-image within the main image. That is, the sub-images are scaled to this size.</p>	
See also	<p><code>image-set</code> <code>make-general-image-set</code></p>	

make-scaled-image-set

Function

Summary	Creates an image set by scaling the images of another image set.	
Package	<code>capi</code>	
Signature	<code>make-scaled-image-set &key <i>image-count</i> <i>width</i> <i>height</i> <i>base-image-set</i> => <i>image-set</i></code>	
Arguments	<i>image-count</i>	An integer.
	<i>width</i>	An integer.
	<i>height</i>	An integer.
	<i>base-image-set</i>	An image set.
Values	<i>image-set</i>	An <code>image-set</code> object.

key is a function that is passed to *sort-function* as its *:key* argument. The default value of *key* is *identity*.

sort is a predicate function that is passed to *sort-function* to compare pairs of items.

reverse-sort is a predicate function that is passed to *sort-function* for reverse sorting.

sort-function is the function that is called to actually do the sorting. Its signature is

```
sort-function items predicate &key key
```

The default value of *sort-function* is *sort*.

Example

```
(setq lp
  (capi:contain
    (make-instance
      'capi:list-panel
      :items '("Apple"
              "Orange"
              "Mangosteen"
              "Pineapple")
      :visible-min-height '(:character 5)
      :sort-descriptions
      (list (capi:make-sorting-description
            :type :length
            :sort
            #'(lambda (x y)
                (> (length x) (length y)))
            :reverse-sort
            #'(lambda (x y)
                (< (length x) (length y))))
          (capi:make-sorting-description
            :type :alphabetic
            :sort 'string-greaterp
            :reverse-sort 'string-lessp))))

(capi:sorted-object-sort-by lp :length)

(capi:sorted-object-sort-by lp :alphabetic)
```

See also `sort-object-items-by`
`sorted-object`
`sorted-object-sort-by`

manipulate-pinboard*Generic Function*

Summary Adds a `pinboard-object` to a pinboard, or removes objects.

Package `capi`

Signature `manipulate-pinboard` *pinboard-layout pinboard-object*
action &key position

Arguments

- pinboard-layout* A `pinboard-layout`.
- pinboard-object* A `pinboard-object`. Can also be a function of one argument, for multiple deletion or a list for multiple addition.
- action* One of `:add`, `:add-top`, `:add-bottom`, `:add-many` or `:delete`. Can also be `:delete-if`, for multiple deletion.
- position* One of `:top` or `:bottom`, or a non-negative integer.

Description The generic function `manipulate-pinboard` adds *pinboard-object* to *pinboard-layout*, or removes one or more `pinboard-objects` from *pinboard-layout*. These operations can also be effected using `(setf layout-description)`, but `manipulate-pinboard` is much more efficient and produces a better display.

If *action* is `:add`, then the `pinboard-object` *pinboard-object* is added according to the value of *position*:

- `:top` On top of the other pinboard objects.
- `:bottom` Below the other pinboard objects.

An integer At index *position* in the sequence of pinboard objects, where 0 is the index of the topmost pinboard object. Values of *position* greater than the number of pinboard objects are interpreted as `:bottom`.

`action :add-top` is the same as passing `action :add` and `position :top`.

`action :add-bottom` is the same as passing `action :add` and `position :bottom`.

`action :add-many` is like calling the function with `action :add` several times, but is more efficient. The value of *pinboard-object* must be a list of `pinboard-objects`, each of which is added at the specified *position*, as for `:add`.

`action :delete` deletes the `pinboard-object` *pinboard-object* from *pinboard-layout*.

When `action` is `:delete-if`, *pinboard-object* should be a function which takes one argument, a `pinboard-object`. This function is applied to each `pinboard-object` in *pinboard-layout* and each object for which it returns true is deleted from *pinboard-layout*.

Note: you can control automatic resizing of *pinboard-object* using `set-object-automatic-resize`.

Example

```
(setq pl
  (capi:contain
    (make-instance 'capi:pinboard-layout
      :visible-min-height 500
      :visible-min-width 200)))
```

Add some `pinboard-objects`:

```
(capi:apply-in-pane-process
pl #'(lambda (pp)
      (dotimes (y 10)
        (let ((yy (* y 40)))
          (capi:manipulate-pinboard
           pp
           (make-instance 'capi:line-pinboard-object
                          :start-x 4 :start-y yy
                          :end-x 54 :end-y (+ 6 yy))
           :add-top)
          (capi:manipulate-pinboard
           pp
           (make-instance 'capi:pinboard-object
                          :x 4 :y (+ 20 yy)
                          :width 50 :height 6
                          :graphics-args
                          '(:background :red))
           :add-top))))))
pl)
```

Remove some pinboard-objects:

```
(capi:apply-in-pane-process
pl
#' (lambda (pp)
      (dotimes (y 15)
        (let ((po (capi:pinboard-object-at-position pp
                                                      10
                                                      (* y
30))))
          (when po (capi:manipulate-pinboard pp
                                              po
                                              :delete))))))
pl)
```

Remove all line-pinboard-objects:

```
(capi:apply-in-pane-process
pl 'capi:manipulate-pinboard pl
#' (lambda (x)
      (typep x 'capi:line-pinboard-object)
      :delete-if))
```

See also

```
pinboard-layout
set-object-automatic-resize
```

map-collection-items

Generic Function

Summary	The generic function <code>map-collection-items</code> calls a specified function on all the items in a collection.
Package	<code>capi</code>
Signature	<code>map-collection-items</code> <i>collection function</i> &optional <i>collect-results-p</i>
Description	Calls <i>function</i> on each item in the <i>collection</i> by calling the <code>collection</code> 's <i>items-map-function</i> . If <i>collect-results-p</i> is non- <code>nil</code> , the results of each call will be returned in a list.
Example	<pre>(setq collection (make-instance 'capi:collection :items '(1 2 3 4 5))) (capi:map-collection-items collection 'princ-to-string t)</pre>
See also	<code>collection</code> <code>choice</code>

map-pane-children

Generic Function

Summary	Calls a function on each of a pane's children.	
Package	<code>capi</code>	
Signature	<code>map-pane-children</code> <i>pane function</i> &key <i>visible test reverse</i>	
Arguments	<i>pane</i>	A CAPI pane.
	<i>function</i>	A function of one argument.
	<i>visible</i>	A boolean. The default value is <code>nil</code> .
	<i>test</i>	A function of one argument, or <code>nil</code> . The default is <code>nil</code> .

reverse A boolean. The default value is `nil`.

Description `map-pane-children` applies *function* to *pane*'s immediate children.

If *visible* is true, then *function* is applied only to the visible children.

If *test* is non-`nil`, it is a function which is applied first to each child, and only those for which *test* returns a true value are then passed to *function*.

If *reverse* is non-`nil`, the order in which the children are processed is reversed.

Example This example constructs a pinboard containing random ellipses. A repainting function is mapped over them, restricted to those with width greater than height.

```

(defun random-color ()
  (aref #(:red :blue :green :yellow :cyan
         :magenta :pink :purple :black :white)
        (random 10)))

(defun random-origin ()
  (list (random 350) (random 250)))

(defun random-size ()
  (list (+ 10 (random 40))
        (+ 10 (random 40))))

(setf ellipses
  (capi:contain
   (make-instance
    'capi:pinboard-layout
    :children
    (loop for i below 40
          for origin = (random-origin)
          for size = (random-size)
          collect
          (make-instance 'capi:ellipse
                        :x (first origin)
                        :y (second origin)
                        :width (first size)
                        :height (second size)
                        :graphics-args
                        (list :foreground
                            (random-color))
                        :filled t))))))

(defun repaint (ellipse)
  (setf (capi:pinboard-object-graphics-args ellipse)
        (list :foreground (random-color)))
  (capi:redraw-pinboard-object ellipse t))

(defun widep (ellipse)
  (capi:with-geometry ellipse
    (> capi:%width% capi:%height%)))

(capi:map-pane-children ellipses 'repaint :test 'widep)

```

See also

`map-pane-descendant-children`

map-pane-descendant-children*Generic Function*

Summary	Calls a function on each of the descendant panes of a pane.	
Package	<code>capi</code>	
Signature	<code>map-pane-descendant-children</code> <i>pane function</i> <i>&key visible test reverse leaf-only</i>	
Arguments	<i>pane</i>	A CAPI pane.
	<i>function</i>	A function of one argument.
	<i>visible</i>	A boolean. The default value is <code>nil</code> .
	<i>test</i>	A function of one argument, or <code>nil</code> . The default is <code>nil</code> .
	<i>reverse</i>	A boolean. The default value is <code>nil</code> .
Description	<p><code>map-pane-descendant-children</code> applies <i>function</i> to <i>pane</i>'s descendent panes (that is, the children and each of their children recursively), depth first.</p> <p>If <i>visible</i> is true, then <i>function</i> is applied only to the visible descendant panes.</p> <p>If <i>test</i> is non-<code>nil</code>, it is a function which is applied first to each descendant pane, and only those for which <i>test</i> returns a true value are then passed to <i>function</i>.</p> <p>If <i>reverse</i> is non-<code>nil</code>, the order in which the children are processed is reversed.</p>	
See also	<code>map-pane-children</code>	

map-typeout*Function*

Package	<code>capi</code>
---------	-------------------

Signature	<code>map-typeout <i>pane</i> &rest <i>args</i></code>
Description	<p>Makes a <code>collector-pane</code> the visible child of a <code>switchable-layout</code>, and returns it as well. The switchable layout is found by looking up the parent hierarchy starting from <code>pane</code>.</p> <p>The switchable layout should have one or more children. If it has one child, a new collector pane is made using <code>args</code> as the <code>initargs</code> with <code>buffer-name</code> defaulting to "Background Output". If it has more than one, it searches through the children to find the first collector pane.</p>
See also	<p><code>unmap-typeout</code> <code>with-random-typeout</code> <code>collector-pane</code></p>

maximum-moving-objects-to-track-edges

Variable

Summary	Limits the tracking of edges in a graph.
Package	<code>capi</code>
Initial Value	15
Description	<p>If there are more than <code>*maximum-moving-objects-to-track-edges*</code> objects being moved in a graph, then edges are not tracked.</p> <p>The value should be an integer.</p>

menu

Class

Summary	The class <code>menu</code> creates a menu for an interface when specified as part of the menu bar (or as a submenu of a menu on the menu bar). It can also be displayed as a context menu.
---------	---

Package	<code>capi</code>
Superclasses	<code>element</code> <code>titled-menu-object</code>
Initargs	<p><code>:items</code> The items to appear in the menu.</p> <p><code>:items-function</code> A function to dynamically compute the items.</p> <p><code>:mnemonic</code> A character, integer or symbol specifying a mnemonic for the menu.</p> <p><code>:mnemonic-escape</code> A character specifying the mnemonic escape. The default value is <code>#\&</code>.</p> <p><code>:mnemonic-title</code> A string specifying the title and a mnemonic.</p> <p><code>:image-function</code> A function providing images for the menu items, or <code>nil</code>.</p>
Accessors	<code>menu-items</code> <code>menu-image-function</code>
Description	<p>A menu has a title, and has items appearing in it, where an item can be either a <code>menu-item</code>, a <code>menu-component</code> or another <code>menu</code>.</p> <p>The simplest way of providing items to a menu is to pass them as the argument <i>items</i>, but if you need to compute the items dynamically you should provide the setup callback <i>items-function</i>. This function should return a list of menu items for the new menu. By default <i>items-function</i> is called on the menu's interface, but a different argument can be specified using the <code>menu-object</code> initarg <i>setup-callback-argument</i>.</p>

If an item is not of type `menu-object`, then it gets converted to a `menu-object` with the item as its data. This function is called before the *popup-callback* and the *enabled-function* which means that they can affect the new items.

To specify a mnemonic in the menu title, you can use the `initarg :mnemonic`. The value *mnemonic* can be:

An integer	The index of the mnemonic in the title.
A character	The mnemonic in the title.
<code>nil</code>	A character is chosen from a list of common mnemonics, or the <code>:default</code> behavior is followed. This is the default.
<code>:default</code>	A mnemonic is chosen using some rules.
<code>:none</code>	The title has no mnemonic.

An alternative way to specify a mnemonic is to pass *mnemonic-title* (rather than *title*) This is a string which provides the text for the menu title and also specifies the mnemonic character. The mnemonic character is preceded in *mnemonic-title* by *mnemonic-escape*, and *mnemonic-escape* is removed from *mnemonic-title* before the text is displayed. For example:

```
:mnemonic-title "&Open File..."
```

At most one character can be specified as the mnemonic in *mnemonic-title*. To make *mnemonic-escape* itself appear in the button, precede it in *mnemonic-title* with *mnemonic-escape*. For example:

```
:mnemonic-title "&Compile && Load File..."
```

If *image-function* is non-`nil`, it should be a function of one argument. *image-function* is called with the data of each menu item and should return one of:

<code>nil</code>	No image is shown.
------------------	--------------------

An `image` object

The menu displays this image.

An image id or `external-image`

The system converts the value to a temporary `image` for the menu item and frees it when it is no longer needed.

If `image-function` is `nil`, no items in the menu have images. This is the default value.

Note: On Cocoa, menu items can contain both images and strings, so the `print-function` should return the appropriate string or "" if no string is required. On Microsoft Windows and Motif, if there is an image then the string is ignored.

Note: When debugging a menu, it may be useful to pop up a window containing a menu with the minimum of fuss. The function `contain` will do just that for you.

Note: To display a menu as a context (right button) menu, use `display-popup-menu`, and to display a menu via a labelled button use `popup-menu-button`.

Note: by default Microsoft Windows hides mnemonics when the user is not using the keyboard. In Windows XP a system preference controls this:

Display > Appearance > Effects > Hide underlined letters...

Examples

```
(capi:contain (make-instance 'capi:menu
                            :title "Test"
                            :items '(:red :green :blue)))

(capi:contain (make-instance
              'capi:menu :title "Test"
                  :items '(:red :green :blue)
                  :print-function
                  'string-capitalize))
```

```
(capi:contain (make-instance
               'capi:menu
               :title "Test"
               :items '(:red :green :blue)
               :print-function 'string-capitalize
               :callback #'(lambda (data interface)
                             (capi:display-message
                              "Pressed ~S" data))))
```

Here is an example showing how to add submenus to a menu:

```
(setq submenu (make-instance 'capi:menu
                              :title "Submenu..."
                              :items '(1 2 3)))

(capi:contain (make-instance
               'capi:menu
               :title "Test"
               :items (list submenu)))
```

Here is an example showing how to use the *items-function*:

```
(capi:contain (make-instance
               'capi:menu
               :title "Test"
               :items-function #'(lambda (interface)
                                   (loop for i below 8
                                         collect (random 10)
                                   ))))
```

Finally, some examples showing how to specify a mnemonic in a menu title:

```
(capi:contain (make-instance
               'capi:menu
               :title "Mnemonic Title"
               :mnemonic 1
               :items '(1 2 3)))

(capi:contain (make-instance
               'capi:menu
               :mnemonic-title "M&nemonic Title"
               :items '(1 2 3)))
```

```
(capi:contain (make-instance
               'capi:menu
               :mnemonic-title "M&e && You"
               :items '("Me" "You")))
```

There are further examples in the directory `examples/capi/applications/`.

See also

- `display-popup-menu`
- `ole-control-add-verbs`
- `menu-component`
- `menu-item`
- `menu-object`
- `popup-menu-button`
- `contain`

menu-component

Class

Summary The class `menu-component` is a choice that is used to group menu items and submenus both visually and functionally. The items contained by the `menu-component` appear separated from other items, menus, or menu components, by separators.

Package `capi`

Superclasses `choice`
 `titled-menu-object`

Initargs `:items` The items to appear in the menu.

`:items-function`
 A setup callback function to dynamically compute the items.

`:selection-function`
 A setup callback function to dynamically compute the selection.

`:selected-item-function`

A setup callback function to dynamically compute the selected item.

`:selected-items-function`

A setup callback function to dynamically compute the selected items.

Description

Because `menu-component` is a choice, the component can have *interaction* `:no-selection`, `:single-selection` or `:multiple-selection` (extended selection does not apply here). This is represented visually in the menu as appropriate to the window system that the CAPI is running on (by ticks in Microsoft Windows, and by radio buttons and check buttons in Motif).

Note that it is not appropriate to have menu components or submenus inside `:single-selection` and `:multiple-selection` components, but it is OK in `:no-selection` components.

items and *items-function* behave as in `menu`.

No more than one of *selection-function*, *selected-item-function* and *selected-items-function* should be non-`nil`. Each defaults to `nil`. If one of these setup callbacks is supplied, it should be a function which is called before the `menu-component` is displayed and which determines which items are selected. By default the setup callback is called on the interface of the `menu-component`, but this argument can be changed by passing the `menu-object` initarg *setup-callback-argument*.

selection-function, if non-`nil`, should return a list of indices suitable for passing to the choice accessor (`setf choice-selection`).

selected-item-function, if non-`nil`, should return an object which is an item in the `menu-component`, or is equal to such an item when compared by the `menu-component`'s *test-function*.

selected-items-function, if non-`nil`, should return a list of such objects.

Example

```
(capi:contain (make-instance
              'capi:menu-component
              :items '(:red :green :blue)
              :print-function 'string-capitalize
              :interaction :single-selection))

(capi:contain (make-instance
              'capi:menu-component
              :items '(:red :green :blue)
              :print-function 'string-capitalize
              :interaction :multiple-selection))

(capi:contain (make-instance
              'capi:menu
              :items (list
                    "An Item"
                    (make-instance
                     'capi:menu-component
                     :items '(:red :green :blue)
                     :print-function
                       'string-capitalize
                     :interaction :no-selection)
                    "Another Item"))))
```

See also

`menu`
`menu-item`

menu-item

Class

Summary

A menu item is an individual item in a menu or menu component, and instances of `menu-item` are created automatically by `define-interface`.

Package

`capi`

Superclasses

`item`
`titled-menu-object`

Initargs	<code>:accelerator</code>	A character, string or plist, or the keyword <code>:default</code> .
	<code>:alternative</code>	A generalized boolean.
	<code>:help-key</code>	An object used for lookup of help. Default value <code>t</code> .
	<code>:mnemonic</code>	A character, integer or symbol specifying a mnemonic for the menu item.
	<code>:mnemonic-escape</code>	A character specifying the mnemonic escape. The default value is <code>#\&</code> .
	<code>:mnemonic-title</code>	A string specifying the text and a mnemonic.
	<code>:selected-function</code>	A setup callback determining whether the item is selected.

Readers `help-key`

Description The text displayed in the menu item is the contents of the *text* slot, or the contents of the *title* slot, otherwise it is the result of applying the *print-function* to the *data*.

selected-function defaults to `nil`, but if non-`nil` it is a function which is called before the `menu-item` is displayed and which determines whether or not the `menu-item` is selected. By default *selected-function* is called on the interface of the `menu-item`, but this argument can be changed by passing the `menu-object` initarg *setup-callback-argument*.

Callbacks are made in response to a user gesture on a `menu-item`. The *callback-type* (see `callbacks`), *callback* and *callback-data-function* (see `menu-object`) are found by looking for a non-`nil` value, first in the `menu-item`, then the `menu-compo-`

ment (if any) and finally the `menu`. This allows a whole menu to have, for example, `callback-type :data` without having to specify this in each item. Some items could override this by having their `callback-type` slot non-`nil` if needed.

To specify a mnemonic in the menu item, you can use the `initarg :mnemonic`, or the `initargs :mnemonic-title` and `:mnemonic-escape`. These `initargs` are all interpreted just as in `menu`.

A menu item should not be used more in more than one place at a time.

`help-key` is interpreted as described for `element`.

`accelerator` can be a character or string specifying a key gesture which will be the accelerator for the menu item.

Note that `both-case-p` characters are not allowed with the single modifier `shift` in the accelerator argument. So instead of

```
:accelerator "shift-x"
```

use

```
:accelerator "X"
```

Note that the `shift` modifier still appears in the menu.

A `both-case-p` character is allowed with `shift` if there are other modifiers, for example

```
:accelerator "alt-shift-x"
```

If `accelerator` is a `character` then the system adds the normal modifier for the platform. That is, `command` on Cocoa and `control` on Microsoft Windows. The shortcut is validated for the platform.

If `accelerator` is a `string` with modifier keys then the system uses it only if it follows the normal conventions for the platform. The shortcut is validated for the platform.

The special virtual modifier name "accelerator" is allowed in string values of *accelerator*. It is interpreted as the normal modifier key for the platform. For example:

```
:accelerator "accelerator-x"
```

means `Control+x` on Microsoft Windows and Motif, and `Command+x` on Cocoa.

If *accelerator* is a plist then its keys are keywords naming some or all of the supported libraries (as returned by `default-library`). The plist's values are characters or strings which the system interprets as above, except that no check is made that the keyboard shortcut is valid for the platform.

accelerator has a special default value `:default`, which means that, depending on `interface-keys-style` for the interface, a standard accelerator is added if the item title matches a standard menu command.

alternative, when true, makes the `menu-item` an "alternative item". Alternative items are invoked if modifiers are held while selecting the "main item". These modifiers are defined by the item's *accelerator*. The main item is the one before the first alternative item, and each alternative item must be within the same menu and menu component. For an example see `examples/capi/elements/accelerators.lisp` and for more information see the section "Alternative menu items" in the *LispWorks CAPI User Guide*.

Example

```
(capi:contain (make-instance 'capi:menu-item
                             :text "Press Me"))

(capi:contain (make-instance 'capi:menu-item
                             :data :red
                             :print-function
                             'string-capitalize))
```

```
(capi:contain (make-instance
              'capi:menu-item
              :data :red
              :print-function 'string-capitalize
              :callback #'(lambda (data interface)
                           (capi:display-message
                            "Pressed ~S"
                            data))))
```

In this example note how the File menu gets accelerators automatically for its standard items:

```
(defun do-menu-item (item)
  (capi:display-message
   (format nil "~A" (capi:item-data item))))

(capi:define-interface mmm () ()
  (:menu-bar f-menu a-menu)
  (:menus
   (f-menu
    "File"
    (("Open..." :data "Open...")
     ("New"       :data "New"))
    :callback 'do-menu-item
    :callback-type :item)
   (a-menu
    "Another Menu"
    (("Open..." :data "Another Open")
     ("New"       :data "Another New")
     ("Blancmange" :data "Blancmange"
              :accelerator #\Ctrl-\b))
    :callback 'do-menu-item
    :callback-type :item))
  (:default-initargs
   :width 300
   :height 200))

;; This causes automatic accelerators on all platforms.

;; That is the default behavior on Microsoft Windows.
(defmethod capi:interface-keys-style ((self mmm))
  :pc)

(capi:contain (make-instance 'mmm))
```

There are further examples in the files `examples/capi/applications/hangman.lisp` and `examples/capi/printing/fit-to-page.lisp`.

See also `choice`
`interface-keys-style`
`menu`
`menu-component`

menu-object

Class

Summary The class `menu-object` is the superclass of all menu objects, and provides functionality for handling generic aspects of menus, menu components and menu items.

Package `capi`

Superclasses `callbacks`

Subclasses `titled-menu-object`

Initargs `:popup-callback`

Callback before the menu appears.

`:enabled-function`

Returns non-`nil` if the menu is enabled.

`:enabled-slot` The object is enabled if the slot is non-`nil`.

`:callback` The selection callback for the object.

`:callback-data-function`

A function to return data for the callback.

`:setup-callback-argument`

If non-`nil`, specifies the argument to the setup callbacks (listed below) that are used to set up the `menu-object`.

Accessors `menu-popup-callback`

Readers	<code>menu-object-enabled</code>
Description	<p>When the menu object is about to appear on the screen, the CAPI does the following:</p> <ol style="list-style-type: none"> 1. The setup callback <i>items-function</i> (if there is one) is called and the result is used to set the items, for <code>menu</code> and <code>menu-component</code>. The argument passed to <i>items-function</i> is the same as for the other setup callbacks (see below). 2. The <i>popup-callback</i> (if there is one) is called and can make arbitrary changes to that object. The <i>popup-callback</i> is always called with the menu object, regardless of the value of <i>setup-callback-argument</i>. 3. The other setup callbacks are called to set up the selection, enabled state and title. These setup callbacks include <i>enabled-function</i> for all <code>menu-objects</code> and <i>title-function</i> for all <code>titled-menu-objects</code>. The additional setup callbacks for <code>menu-component</code> are <i>selection-function</i>, <i>selected-item-function</i>, and <i>selected-items-function</i>. <code>menu-item</code> has the additional setup callback <i>selected-function</i>. By default <i>setup-callback-argument</i> is <code>nil</code>, which means that each of the setup callbacks is called on the interface of the <code>menu-object</code>. If <i>setup-callback-argument</i> is non-<code>nil</code>, then it is passed (instead of the interface) as the argument to each of the setup callbacks. 4. The menu containing the object appears with all of the changes made. <p>Note that <i>enabled-slot</i> is a short-hand means of creating an <i>enabled-function</i> which checks the value of a slot in the menu object's interface.</p> <p>The enabled state of a <code>menu-object</code> is computed each time the menu is displayed, using <i>enabled-function</i> or <i>enabled-slot</i>. Therefore the accessor <code>menu-object-enabled</code> is only useful as a reader.</p>

The *callback* argument is placed in the *selection-callback*, *extend-callback* and *retract-callback* slots unless these are given explicitly, and so will get called when the menu object is selected or deselected.

The *callback-data-function* is a function that is called with no arguments and the value it returns is used as the data to the callbacks.

Example

```
(capi:contain (make-instance
               'capi:menu-item
               :text "Press Me"
               :enabled-function #'(lambda (item)
                                     (eq (random 2)
                                         1))))
```

The next example illustrates the use of *setup-callback-argument*. The *initialize-instance* method adds to the "Some Numbers" menu a sub-menu that lists the selected items in the `list-panel`. By using *setup-callback-argument* in this menu, the setup callbacks (in this case *enabled-function* and *items-function*) are called directly on the `list-panel`.

Note that, while this example uses a CAPI object as the *setup-callback-argument*, any object of any type can be used.

```

(capi:define-interface my-interface ()
  ()
  (:panes
   (list-panel
    capi:list-panel
    :items '(1 2 3 4 5 6 7 8 9 0)
    :interaction :extended-selection
    :visible-min-height '(character 10)))
  (:menus
   (a-menu
    "Some Numbers"
    ("One" "Two")
    ))
  (:menu-bar a-menu))

(defmethod initialize-instance :after
  ((self my-interface) &key)
  (with-slots (a-menu list-panel) self
    (setf (capi:menu-items a-menu)
          (append
           (capi:menu-items a-menu)
           (list
            (make-instance 'capi:menu
                          :items-function
                          'capi:choice-selected-items
                          :setup-callback-argument
                          list-panel
                          :enabled-function
                          'capi:choice-selection
                          :title
                          "Selected Items"))))))

(capi:display (make-instance 'my-interface))

```

See also `menu`
`menu-item`
`menu-component`

merge-menu-bars

Generic Function

Summary Computes the menu bar for a `document-frame`.

Package `capi`

Signature	<code>merge-menu-bars</code> <i>frame document</i> => <i>menus</i>	
Arguments	<i>frame</i>	A <code>document-frame</code> .
	<i>document</i>	An interface or nil.
Values	<i>menus</i>	A list of <code>menu</code> objects.
Description	<p>The generic function <code>merge-menu-bars</code> is called by the system to compute the menu bar for a <code>document-frame</code> interface.</p> <p>The set of visible menus in such an interface is typically made up from those of the frame and those of the active document within it.</p> <p>There is a built-in unspecialized method that appends the menu bars of the two interfaces and is equivalent to this:</p> <pre>(defmethod capi:merge-menu-bars ((frame t) (document t)) (append (capi:interface-menu-bar-items frame) (and document (capi:interface-menu-bar-items document))))</pre> <p>You can customize the menu bar by adding methods which specialize on particular frame and document interface classes.</p>	
See also	<code>document-frame</code> <code>interface</code> <code>menu</code>	

message-pane

Class

Summary	The class displaying the message when a pane is created with the <code>:message</code> initarg.
Package	<code>capi</code>

Superclasses	<code>title-pane</code>
Description	<p>The class <code>message-pane</code> is used to implement the message decoration on subclasses of <code>titled-object</code>.</p> <p>A <code>message-pane</code> with <i>text</i> "Message" is created automatically when a <code>titled-object</code> is created with <i>message</i> "Message".</p>
See also	<code>titled-object</code>

modify-editor-pane-buffer

Function

Summary	The <code>modify-editor-pane-buffer</code> function allows you to modify the contents and fill mode of a specified buffer.
Package	<code>capi</code>
Signature	<code>modify-editor-pane-buffer</code> <i>pane</i> &key <i>contents</i> <i>flag</i> <i>fill</i> <i>fixed-fill</i> <i>force</i>
Description	<p>The <code>modify-editor-pane-buffer</code> function modifies the <code>editor-pane</code> <i>pane</i> according to the keyword arguments.</p> <p>The argument <i>contents</i> (if non-<code>nil</code>) supplies a new string to place in the buffer.</p> <p><i>flag</i>, if given, sets the flag slot of the editor buffer, which is used to mark it for various specialized uses.</p> <p>If <i>fill</i> is non-<code>nil</code> the editor fills each paragraph in the buffer. If <i>fill</i> is a fixnum then the buffer is filled at that width. If <i>fill</i> is <code>:default</code> (the default value) and <i>fixed-fill</i> is supplied then the value <i>fixed-fill</i> is used. Otherwise the buffer is filled to the window width.</p> <p><i>fixed-fill</i> defaults to <code>nil</code>.</p>
See also	<code>editor-pane</code>

mono-screen

Class

Summary	The <code>mono-screen</code> class is created for monochrome screen.
Package	<code>capi</code>
Superclasses	<code>screen</code>
Description	This is a subclass of <code>screen</code> that gets created for monochrome screens. It is primarily available as a means of discriminating on whether or not to use colors in an interface.
See also	<code>color-screen</code>

move-line

Generic Function

Summary	Moves a <code>line-pinboard-object</code> .
Package	<code>capi</code>
Signature	<code>move-line</code> <i>line-pinboard-object</i> <i>start-x</i> <i>start-y</i> <i>end-x</i> <i>end-y</i> <i>&key</i> <i>redisplay</i>
Arguments	<i>line-pinboard-object</i> An instance of <code>line-pinboard-object</code> or a subclass. <i>start-x</i> The x coordinate of the start of the line. <i>start-y</i> The y coordinate of the start of the line. <i>end-x</i> The x coordinate of the end of the line. <i>end-y</i> The y coordinate of the end of the line. <i>redisplay</i> A boolean.
Description	The generic function <code>move-line</code> moves a line to a new location with end points specified by the coordinate arguments.

This automatically adjusts the geometry of the object, taking into account other constraints. Examples of such constraints are the label in a `labelled-line-pinboard-object` and the arrowhead in a `arrow-pinboard-object`.

The default value of `redisplay` is `t`, which means that the changed line is redrawn immediately. If you are moving many objects at the same time, it is useful to pass `:redisplay nil`.

See also `line-pinboard-object`
`line-pinboard-object-coordinates`

multi-column-list-panel

Class

Summary A list panel with multiple columns of text.

Package `capi`

Superclasses `list-panel`

Initargs `:column-function`
A function of one argument. The default is `identity`.

`:item-print-functions`
A function of one argument, or a list of such functions.

`:columns` A list of column specifications.

`:header-args` A plist of keywords and values.

`:auto-reset-column-widths`
A boolean. The default is `t`.

Description The class `multi-column-list-panel` is a list panel which displays multiple columns of text. The columns can each have a title.

Note that this is a subclass of `list-panel`, and hence of `choice`, and inherits the behavior of those classes.

Each item in a `multi-column-list-panel` is displayed in a line of multiple objects. The corresponding objects of each line are aligned in a column.

The *column-function* generates the objects for each item. It should take an item as its single argument and return a list of objects to be displayed. The default *column-function* is `identity`, which works if each item is a list.

The *item-print-functions* argument determines how to calculate the text to display for each element. If *item-print-functions* is a single function, it is called on each object, and must return a string. Otherwise *item-print-functions* should be a sequence of length no less than the number of columns. The text to display for each object is the result (again, a string) of calling the corresponding element of *item-print-functions* on that object.

The *columns* argument specifies the number of columns, and whether the columns have titles and callbacks on these titles.

Each element of *columns* is a specification for a column. Each column specification is a plist of keyword and values, where the allowed keywords are as follows:

- `:title` Specifies the title to use for the column. If any of the columns has a title, a header object is created which displays the titles. The values of the `:title` keywords are passed as the *items* of the header, unless *header-args* specifies `:items`.
- `:adjust` Specifies how to adjust the column. The value can be one of `:right`, `:left`, or `:center`.
- `:width` Specifies the width of the columns.

`:visible-min-width`

Minimum width of the column.

`:gap`

Specifies an additional gap to the right of the text in the column.

The values of `:width`, `:visible-min-width` and `:gap` are interpreted as standard geometric hints. See `element` for information about these hints.

`columns` should indicate how many columns to display. At a minimum the value needs to be `(() ())` for two columns without any titles

`header-args` is a plist of `initargs` passed to the header which displays the titles of the columns. The header object is a `collection`. The following `collection` `initargs` are useful to pass in `header-args`:

`:selection-callback`

The callback for clicking on the header.

`:callback-type`

Defines the arguments of the `selection-callback`.

`:items`

The items of the header object. Note that `:items` overrides `:title` if that is supplied in `columns`.

`:print-function`

Controls how each of `items` is printed, providing the title of each column.

`header-args` may also contain the keyword `:alignments`. The value should be a list of alignment keywords, each of which is interpreted like an `:adjust` value in `columns`. The alignment is applied to the title only.

If `auto-reset-column-widths` is true, then the widths of the columns are recomputed when the items of the `multi-column-list-panel` are set.

Note: similar and enhanced functionality is provided by `list-view`.

Example This example uses the `columns` initarg:

```
(capi:contain
 (make-instance
  'capi:multi-column-list-panel
  :visible-min-width 300
  :visible-min-height :text-height
  :columns '((:title "Fruits"
              :adjust :right
              :width (character 15))
            (:title "Vegetables"
              :adjust :left
              :visible-min-width (character 30)))
  :items '(("Apple" "Artichoke")
           ("Pomegranate" "Pumpkin"))))
```

This example uses `header-args` to add callbacks and independent alignment on the titles:

```
(defun mclp-header-callback (interface item)
  (declare (ignorable interface))
  (capi:display-message "Clicked on ~a" item))

(capi:contain
 (make-instance
  'capi:multi-column-list-panel
  :visible-min-width 300
  :visible-min-height :text-height
  :columns '((:adjust :right
              :width (character 15))
            (:adjust :left
              :visible-min-width (character 30)))
  :header-args '(:items ("Fruits" "Vegetables")
                :selection-callback
                  mclp-header-callback
                :alignments (:left :right))
  :items '(("Apple" "Artichoke")
           ("Pomegranate" "Pumpkin"))))
```

This example uses `column-function` to implement a primitive process browser:

```
(defun get-process-elements (process)
  (list (mp:process-name process)
        (mp:process-whostate process)
        (mp:process-priority process)))

(capi:contain
 (make-instance
  'capi:multi-column-list-panel
  :visible-min-width '(character 70)
  :visible-min-height '(character 15)
  :items (mp:list-all-processes)
  :columns '(:title "Name" :adjust :left
             :visible-min-width (character 30))
            (:title "State" :adjust :center
             :visible-min-width (character 20))
            (:title "Priority" :adjust :center
             :visible-min-width (character 12)))
  :column-function 'get-process-elements))
```

See also `collection`
 `list-panel`
 `list-view`

multi-line-text-input-pane

Class

Summary A pane allowing several lines of text to be entered.

Package `capi`

Superclasses `text-input-pane`

Description The `multi-line-text-input-pane` class behaves like a `text-input-pane`, except that the text entered by the user is allowed to span several lines — that is, it is allowed to contain Newline characters.

See also `text-input-pane`

ole-control-add-verbs

Function

Summary	Adds to the menu entries for the "verbs" that a component in an <code>ole-control-pane</code> supports.
Signature	<code>ole-control-add-verbs</code> <i>pane menu item-identifier</i>
Arguments	<i>pane</i> An <code>ole-control-pane</code> . <i>menu</i> A menu. <i>item-identifier</i> A string or symbol.
Description	<p>The function <code>ole-control-add-verbs</code> adds to the menu entries for the "verbs" that the component supports. The <code>ole-control-pane</code> <i>pane</i> must have an object already, and the <code>menu</code> <i>menu</i> must have already been created, so <code>ole-control-add-verbs</code> is typically called in the <i>popup-callback</i> of <i>menu</i>.</p> <p><i>item-identifier</i> identifies an item in the menu or a component in the menu (but not in a sub-menu), either by being <code>eq</code> to the name of the item or <code>equalp</code> to the title of the item. If the item is found, it is replaced either by a sub-menu with the verbs that the object supports, or, if the object supports only one verb, by an entry for this.</p> <p>When the user selects an added menu item, the verb is passed to the object (by a call to <code>IOleObject::DoVerb</code>).</p> <p>Note: this function is implemented only in LispWorks for Windows. Load the functionality by <code>(require "embed")</code>.</p>
Example	See the example in <code>examples/com/ole/simple-container/doc-viewer-pair.lisp</code>
See also	<code>menu</code> <code>ole-control-pane</code>

ole-control-close-object*Function*

Summary	Closes the object in an <code>ole-control-pane</code> .
Signature	<code>ole-control-close-object pane</code>
Arguments	<code>pane</code> An <code>ole-control-pane</code> .
Description	The function <code>ole-control-close-object</code> closes the object that is currently in the <code>ole-control-pane pane</code> . Note: this function is implemented only in LispWorks for Windows. Load the functionality by <code>(require "embed")</code> .
Example	See the example in <code>examples/com/ole/simple-container/doc-viewer-pair.lisp</code>
See also	<code>ole-control-pane</code>

ole-control-component*Class*

Summary	An implementation of the interfaces in the OLE Control protocol.
Package	<code>capi</code>
Superclasses	<code>com:standard-i-unknown</code>
Initargs	<code>:pane-function</code> A function that is called when OLE embeds the Control in a container. <code>:create-callback</code> A function called just after the pane is created.

`:destroy-callback`

A function called just before the pane is destroyed.

Readers

`ole-control-component-pane`

Description

The class `ole-control-component` provides an implementation of the interfaces in the OLE Control protocol, to allow a CAPI pane to be embedded in an OLE Control container implemented outside LispWorks. It is typically used with the macro `define-ole-control-component` to define a subclass of `ole-control-component` that implements a particular coclass from a type library. Instances of this class are usually created by the COM runtime system, not by explicit calls to `make-instance`.

A function designator *pane-function* must be supplied. *pane-function* that is called when OLE embeds the Control in a container. It receives the component as its argument and should return a CAPI pane that will implement the visual aspects of the control.

Note: The pane returned by *pane-function* must be a `output-pane`, `layout` or `interface` in the current implementation. The pane is stored in the component and can be accessed using the reader `ole-control-component-pane`.

create-callback, if non-`nil`, is a function called when the pane returned by *pane-function* has been created in the window system. The argument is the pane itself. *create-callback* can perform initialization such as loading images.

destroy-callback, if non-`nil`, is a function called when the pane returned by *pane-function* is going to be destroyed. The argument is the pane itself. *destroy-callback* can perform cleanups.

Note: When using an `ole-control-component`, the normal hierarchy of CAPI objects such as a layout and an interface do not exist above it. The layout and control of the top level

window is the responsibility of the application that embeds the control. It can communicate with the control by using COM/Automation.

Note: `ole-control-component` is implemented only in LispWorks for Windows. Load the functionality by

```
(require "embed")
```

See also `define-ole-control-component`

ole-control-doc

Class

Summary	A class that implements the document around the object inside an <code>ole-control-pane</code> .
Package	<code>capi</code>
Superclasses	<code>pinboard-layout</code>
Subclasses	<code>ole-control-frame</code>
Description	<p>The pane class <code>ole-control-doc</code> can be used to implement the document around the object inside an <code>ole-control-pane</code>. That is, it supports the <code>IOleInPlaceUIWindow</code> interface. Note that this is optional, and is rarely useful.</p> <p>To use it the <code>ole-control-doc</code> pane needs to be the parent, not necessarily directly, of an <code>ole-control-pane</code>. When the object calls <code>IOleInPlaceSite::GetWindowContext</code>, it will get (in the <code>ppdoc [out]</code> argument) an <code>IOleInPlaceUIWindow</code> interface associated with the <code>ole-control-doc</code>.</p> <p>A <code>ole-control-doc</code> must have exactly one sub-pane (that is, the length of its <i>description</i> must be 1), but underneath this pane there can be many panes.</p>

Normally the program does not need to do anything else with the `ole-control-doc`. It acts in response to resizing of the window and method calls from the object on the `IOleInPlaceUIWindow` interface.

Note: `ole-control-doc` is implemented only in LispWorks for Windows. Load the functionality by `(require "embed")`.

Note: even though it is a subclass of `pinboard-layout`, normally you should not use the `pinboard-layout` functionality when using `ole-control-doc`.

Example See the example in
`examples/com/ole/simple-container/doc-viewer-pair.lisp`

See also `ole-control-pane`

ole-control-frame

Class

Summary Implements the frame of components in an `ole-control-pane`.

Package `capi`

Superclasses `ole-control-doc`

Description The pane class `ole-control-frame` implements the frame of components, that is it supports the `IOleInPlaceFrame` interface. When an `ole-control-pane` pane is created, it looks upwards in the hierarchy of panes, and if finds an `ole-control-frame` pane it uses this as the frame. It uses the first such pane found. When the object in the `ole-control-pane` calls `IOleInPlaceSite::GetWindowContext`, it gets back in the `ppframe` arg an interface associated with this frame.

Like `ole-control-doc`, a `ole-control-frame` can have only one sub-pane, which itself may contain many panes.

Normally the program does not need to do anything else with the `ole-control-frame`. It acts in response to resizing of the window and method calls from the object on the `IOleInPlaceFrame` interface.

Note that having a frame is optional, and ActiveX does not need it. It is required when embedding an application by `ole-control-insert-object`.

Note: `ole-control-frame` is implemented only in LispWorks for Windows. Load the functionality by `(require "embed")`.

Note: even though it is a subclass of `pinboard-layout`, normally you should not use the `pinboard-layout` functionality when using `ole-control-frame`.

Example	See the example in <code>examples/com/ole/simple-container/doc-viewer-pair.lisp</code>
See also	<code>ole-control-insert-object</code> <code>ole-control-pane</code>

ole-control-i-dispatch

Function

Summary	Returns the <code>com:i-dispatch</code> of the component of an <code>ole-control-pane</code> .	
Signature	<code>ole-control-i-dispatch pane => result</code>	
Arguments	<i>pane</i>	An <code>ole-control-pane</code> .
Values	<i>result</i>	A <code>com:i-dispatch</code> or <code>nil</code> .

Description The function `ole-control-i-dispatch` returns the `com:i-dispatch` (that is, the `IDispatch` interface) of the component, or `nil` if there isn't any. The `com:i-dispatch` is the one that would be returned by `com:query-interface` on the `I-Ole-object`.

Note: calling `ole-control-i-dispatch` does not affect the reference count of the interface.

Note: this function is implemented only in LispWorks for Windows. Load the functionality by `(require "embed")`.

See also `ole-control-pane`

ole-control-insert-object

Function

Summary Embeds a user-specified document in an `ole-control-pane`.

Signature `ole-control-insert-object pane`

Arguments *pane* An `ole-control-pane`.

Description The function `ole-control-insert-object` prompts the user for a document using the Microsoft Windows function `OleUIInsertObject`.

When the user specifies a document in the dialog presented, `ole-control-insert-object` embeds this document in the `ole-control-pane pane`.

Note: this function is implemented only in LispWorks for Windows. Load the functionality by `(require "embed")`.

Example See the example in `examples/com/ole/simple-container/doc-viewer-pair.lisp`

See also `ole-control-pane`

ole-control-ole-object*Function*

Summary	Returns the <code>com:i-ole-object</code> of the component of an <code>ole-control-pane</code> .	
Signature	<code>ole-control-ole-object pane => result</code>	
Arguments	<code>pane</code>	An <code>ole-control-pane</code> .
Values	<code>result</code>	A <code>com:i-ole-object</code> or <code>nil</code> .
Description	<p>The function <code>ole-control-ole-object</code> returns the <code>com:i-ole-object</code> (that is, the <code>IOleObject</code> interface) of the component of the <code>ole-control-pane</code> <code>pane</code>, or <code>nil</code> if there isn't any.</p> <p>Note: calling <code>ole-control-ole-object</code> does not affect the reference count of the interface.</p> <p>Note: this function is implemented only in LispWorks for Windows. Load the functionality by <code>(require "embed")</code></p>	
See also	<code>ole-control-pane</code>	

ole-control-pane*Class*

Summary	A class that implements embedding of external components on Microsoft Windows.	
Package	<code>capi</code>	
Superclasses	<code>pinboard-layout</code>	
Initargs	<code>:component-name</code>	A string or <code>nil</code> .
	<code>:user-component</code>	A COM interface pointer or <code>nil</code> .

:save-name A string.
:insert-callback
 A function.
:close-callback
 A function.
:sinks
 A list of sink specifications.

Description The class `ole-control-pane` is used to implement embedding of external components.

Note: `ole-control-pane` is implemented only in LispWorks for Windows. Load the functionality by (`require "embed"`).

Note: even though it is a subclass of `pinboard-layout`, normally you should not use the `pinboard-layout` functionality when using `ole-control-pane`.

component-name (if non-`nil`) specifies the *component-name* of the pane, as used by `component-name`.

user-component (if non-`nil`) is a COM interface pointer of an object that supports the `I-OLE-OBJECT` interface, and is ready to display as described in `ole-control-user-component`.

save-name is used when creating the `IStorage` object for this component.

insert-callback (if non-`nil`) is a function that takes a single argument, the pane. It is called immediately after a component was inserted into the pane. This can be used for any additional initialization that is required, for example setting the properties of the control.

close-callback (if non-`nil`) is a function that takes a single argument, the pane. It is called just before the component is going to be closed, and can be used to do any cleanups that may be required.

sinks is a list of sink specifications for attaching event handlers to the source interfaces of the control. Each element of *sinks* should be a list of the form:

```
(interface-name &key invoke-callback sink-class sink)
```

The *interface-name* is used to specify the name of the source interface in the control, which is either a string naming the interface or `:default` for the default source interface. If *invoke-callback* is given, then it should be a function which will be called with the pane, method-name, method-kind and arguments vector for each source event. The *sink-class* can be given to set the class of the internal object used for the sink interface. This is similar to calling `attach-simple-sink`. Alternatively, instead of calling *invoke-callback*, the *sink* can be specified directly. This is similar to calling `attach-sink`.

When the `ole-control-pane` is destroyed, the sinks are automatically detached.

There are currently three ways to insert an external component into an `ole-control-pane`. These are:

1. Call `ole-control-user-component`, which asks the user for something to insert.
2. Set the *component-name* of the pane. This can be done either via the initarg `:component-name` or by calling `(setf component-name)`.
3. Set the *user-component* of the pane, either via the initarg `:user-component` or by calling `(setf ole-control-user-component)`.

Example

```
(capi:contain
 (list
  (make-instance 'capi:ole-control-pane
                 :component-name "OWC.Spreadsheet.9"))))
```

See `examples/com/ole/simple-container/sink.lisp` for a full example.

See also `attach-sink`
`component-name`
`detach-sink`
`interface-menu-groups`
`ole-control-add-verbs`
`ole-control-close-object`
`ole-control-i-dispatch`
`ole-control-insert-object`
`ole-control-ole-object`
`ole-control-pane-frame`
`ole-control-user-component`
`report-active-component-failure`

ole-control-pane-frame

Function

Summary Returns the `ole-control-frame` of an `ole-control-pane`.

Signature `ole-control-pane-frame pane => result`

Arguments *pane* An `ole-control-pane`.

Values *result* An `ole-control-frame` or `nil`.

Description The function `ole-control-pane-frame` returns the `ole-control-frame` of the `ole-control-pane` *pane*, if there is one.

Note: this function is implemented only in LispWorks for Windows. Load the functionality by `(require "embed")`.

See also `ole-control-frame`
`ole-control-pane`

ole-control-pane-simple-sink*Class*

Summary	A class that implements a sink interface for an embedded component on Microsoft Windows.
Package	<code>capi</code>
Superclasses	<code>com:simple-i-dispatch</code>
Initargs	<code>:ole-control-pane</code> A class instance.
Description	<p>The class <code>ole-control-pane-simple-sink</code> is used by the function <code>attach-simple-sink</code> to implement a sink interface for an embedded component on Microsoft Windows.</p> <p><i>ole-control-pane</i> is the object of type <code>ole-control-pane</code> to whose source interface the sink is being attached.</p> <p>This class can be subclassed to provide additional functionality in callbacks. See <code>com:simple-i-dispatch</code> in the <i>LispWorks COM/Automation User Guide and Reference Manual</i> for more details.</p> <p>Note: <code>ole-control-pane-simple-sink</code> is implemented only in LispWorks for Windows. Load the functionality by <code>(require "embed")</code>.</p>
See also	<code>attach-simple-sink</code> <code>ole-control-pane</code>

ole-control-user-component*Function*

Summary	Gets and sets the <i>user-component</i> of an <code>ole-control-pane</code> .
Signature	<code>ole-control-user-component pane => user-component</code>

```
(setf ole-control-user-component) user-component pane =>
user-component
```

Arguments *pane* An `ole-control-pane`.
 user-component A COM interface pointer.

Description The function `ole-control-user-component` gets and sets the *user-component* of the `ole-control-pane` *pane*.

 user-component (if non-`nil`) is a COM interface pointer of an object that supports the `I-OLE-OBJECT` interface, and has been opened and initialized and is ready to be displayed. This is typically created by calling `OleCreate`, `OleCreateFromFile`, `OleCreateFromData` or `OleLoad` with *pClientSite* null.

 The *user-component* is closed and released by the `ole-control-pane` *pane*, so after you have called `(setf ole-control-user-component)` you should not try to use it again or release it. Setting *user-component* also sets the pane's *component-name* to `nil`.

 Note: this function is implemented only in LispWorks for Windows. Load the functionality by `(require "embed")`.

See also `ole-control-pane`

option-pane

Class

Summary A pane which offers a choice of items, but which displays only the currently selected item.

Package `capi`

Superclasses `choice`
 `titled-object`
 `simple-pane`

Initargs `:enabled` Non-`nil` if the option pane is enabled.

`:visible-items-count`

An integer specifying the maximum length of the popup menu, or the symbol `:default`.

`:popup-callback`

A function called just before the popup menu appears, or `nil`.

`:image-function`

A function providing images for items, or `nil`.

`:separator-item`

An item that acts as a separator between other items, or `nil`.

`:enabled-positions`

A list of fixnums, or the keyword `:all`.

Accessors

`option-pane-enabled`
`option-pane-image-function`
`option-pane-visible-items-count`
`option-pane-popup-callback`
`option-pane-separator-item`
`option-pane-enabled-positions`

Description

The class `option-pane` provides a pane which offers a choice between a number of items via a popup menu. Only the currently selected item is displayed.

The class `option-pane` inherits from `choice`, and so has all of the standard choice behavior such as selection and callbacks. It also has an extra *enabled* slot along with an accessor which is used to enable and disable the option pane.

If *visible-items-count* is an integer then the popup menu is no longer than this, and is scrollable if there are more items.

If *visible-items-count* is `:default`, then the popup menu is no longer than 10. This is the default value.

When *popup-callback* is non-`nil`, it should be a function of one argument that will be called just before the popup menu appears when the user clicks on it. The single argument to the function is the option pane and the return value is ignored. If required, the function can change the items or selection of the pane. The default value of *popup-callback* is `nil`.

If *image-function* is non-`nil`, it should be a function of one argument. *image-function* is called with each item and should return one of:

`nil` No image is shown.

An `image` object

 The pane displays this image.

An image id or `external-image`

 The system converts the value to a temporary `image` for the item and frees it when it is no longer needed.

If *image-function* is `nil`, no items have images. This is the default value.

Note: `:image-function` is currently only implemented for Microsoft Windows and Cocoa.

separator-item should be an item (compared using *test-function*) that acts as a separator between other items. A separator item is not selectable. The default value `nil` means that there are no separators (regardless of *test-function*).

Note: on Motif, the separator is represented simply as a blank item between the other items.

If *enabled-positions* is `:a11` then all the items can be selected. Otherwise the value is a list of fixnums indicating the positions in the item list which can be selected. The default value is `:a11`.

Note: on Motif, there is no visible representation of the disabled items.

Example

This example sets the selection and changes the enabled state of an `option-pane`:

```
(setq option-pane (capi:contain
                   (make-instance 'capi:option-pane
                                   :items '(1 2 3 4 5)
                                   :selected-item 3)))

(capi:apply-in-pane-process
 option-pane #'(setf capi:choice-selected-item)
 5 option-pane)

(capi:apply-in-pane-process
 option-pane #'(setf capi:option-pane-enabled)
 nil option-pane)

(capi:apply-in-pane-process
 option-pane #'(setf capi:option-pane-enabled)
 t option-pane)
```

This example illustrates the use of *visible-items-count*:

```
(capi:contain
 (make-instance 'capi:option-pane
                :items
                (loop for i below 20 collect i)
                :visible-items-count 6))
```

There is a further example in the file `examples/capi/choice/option-pane.lisp`.

output-pane

Class

Summary

An output pane is a pane whose display and input behavior can be controlled by the programmer.

Package

`capi`

Superclasses	<code>titled-object</code> <code>simple-pane</code> <code>gp:graphics-port-mixin</code>
Subclasses	<code>pinboard-layout</code> <code>editor-pane</code>
Initargs	<p><code>:display-callback</code> A function that knows how to redisplay the pane.</p> <p><code>:input-model</code> A list of input specifications, otherwise known as a command table.</p> <p><code>:scroll-callback</code> A function called when the pane is scrolled, or <code>nil</code>. The default is <code>nil</code>.</p> <p><code>:pane-can-scroll</code> A generalized boolean specifying whether the pane itself is responsible for drawing into the visible area.</p> <p><code>:focus-callback</code> A function called when the pane gets or loses the input focus, or <code>nil</code>. The default is <code>nil</code>.</p> <p><code>:resize-callback</code> A function called when the pane is resized, or <code>nil</code>. The default is <code>nil</code>.</p> <p><code>:create-callback</code> A function called just after the pane is created.</p> <p><code>:destroy-callback</code> A function called just before the pane is destroyed.</p>

`:graphics-options`

A platform-specific plist of options controlling how graphics are drawn.

Accessors

`output-pane-display-callback`
`output-pane-focus-callback`
`output-pane-resize-callback`
`output-pane-scroll-callback`
`output-pane-create-callback`
`output-pane-destroy-callback`

Readers

`output-pane-input-model`
`output-pane-graphics-options`

Description

The class `output-pane` is a subclass of `gp:graphics-port-mixin` which means that it supports many of the graphics ports drawing operations. When the CAPI needs to redisplay a region of the output pane, the *display-callback* gets called with the `output-pane` and the *x*, *y*, *width* and *height* of the region that needs redrawing. The *display-callback* should then use graphics port operations to redisplay that area. To force an area to be re-displayed, use the function `invalidate-rectangle`.

The *input-model* provides a means to get callbacks on mouse and keyboard gestures. An *input-model* is a list of mappings from gesture to callback, where each mapping is a list

(gesture callback . extra-callback-args)

gesture specifies the type of gesture, which can be *gesture spec*, *character*, *button*, *key*, *command* or *motion*.

In a *gesture spec* mapping, *gesture* can be simply the keyword `:gesture-spec`, which matches any keyboard input. For specific mappings, *gesture* is a list

(:gesture-spec data [modifier])*

in which *data* is a character object or an integer between 0 and `char-code-limit` (interpreted as the character object obtained by `code-char`), or a keyword naming a function key,

and each *modifier* is one of the keywords `:shift`, `:control` and `:meta`. Note that the `:meta` modifier is received only when the keys style is `:emacs` (see `interface-keys-style`). On Cocoa, the *modifier* value `:hyper` is interpreted as the `Command` key for mouse input (though note that `Command` is reserved for menu accelerators in interfaces). So to match the mouse gesture `Command+Click` you would use:

```
(:button-1 :press :hyper)
```

as the *gesture* in the mapping.

Also *data* can be a string which is interpreted as a gesture spec as if by `sys:coerce-to-gesture-spec`. See the *Lisp-Works Reference Manual* for a description of this and other functions for manipulating gesture spec objects.

In a character mapping, *gesture* can be simply the keyword `:character`, which matches any character input. For specific mappings, *gesture* can be a list containing a single character object *char*, or a list

```
(char)
```

Note: where input would match both a gesture spec mapping and a character mapping, the gesture spec mapping takes precedence.

In a button mapping, *gesture* should be list

```
(button action [modifiers]*)
```

where *button* is one of `:button-1`, `:button-2` or `:button-3` denoting the mouse buttons. *action* is one of `:press`, `:release`, `:second-press`, `:third-press`, `:nth-press` and `:motion`, and each *modifier* is one of the keywords `:shift`, `:control` and `:meta`. However, `:meta` gestures are not generated in non-IDE windows on Cocoa. `:third-press` and `:nth-press` are supported only on Cocoa and Motif.

In a key mapping, *gesture* should contain the key in question (or the keyword `:key` meaning any key) along with an optional action (one of `:press` or `:release`) and zero or more keyboard modifiers.

In a motion mapping, *gesture* can either be defined in terms of dragging a button (in which case it is defined as a button gesture with *action* `:motion`), or it can be defined for motions whilst no button is down by just specifying the keyword `:motion` with no additional arguments.

In a command mapping, *gesture* should be a command which is defined using `define-command`, and provides an alias for a gesture. The following commands are predefined:

```
:post-menu      (:button-3 :release) on Microsoft Win-
                  dows.
```

```
                (:button-3 :press) on Motif.
```

```
                (:button-1 :press :control) on Mac OS
                  X.
```

```
:control-post-menu
```

```
                (:button-3 :press :control) on
                  Microsoft Windows, Motif and Mac OS X.
```

```
:keyboard-post-menu
```

```
                (:gesture-spec :f10 :shift) on
                  Microsoft Windows, Motif and Mac OS X.
```

Note that it is recommended you follow the style guidelines and conventions of the platform you are developing for when mapping gestures to results.

When user input matches *gesture*, *callback* is called with standard arguments and any *extra-callback-args* as extra arguments. The standard arguments are the `output-pane`, the x cursor position, the y cursor position, and in the case of gesture spec, character or key mappings, the input object that matched.

Button mappings with *action* `:press` are matched on the first button click, and they pass the standard arguments to their *callback*. Button mappings with *action* `:second-press` and `:third-press` are matched on the second and third button click made in quick succession, and again they pass the standard arguments to their *callback*. Button mappings with *action* `:nth-press` are matched on the *n*th button click made in quick succession when there is not a more specific match with `:press`, `:second-press` or `:third-press`. Then the integer *n* is also passed as the fourth argument to *callback*, representing the number of times that the button has been pressed in quick succession. If there is a `:press`, `:second-press` or `:third-press` handler then that is invoked instead of `:nth-press` for the corresponding number of presses.

Note: mouse gestures with `:press`, `:second-press`, `:third-press` and `:nth-press` actions can each be expected to be followed by a `:release` action.

Note: In some circumstances `:motion` events can be received even when the `output-pane` does not have the input focus. See window style `:motion-events-without-focus` under `interface` for details.

If *pane-can-scroll* is true then the pane is responsible for handling scrolling, by redrawing. It should draw into the visible area according to the scroll parameters. This is known as internal scrolling and an example is `editor-pane`. If *pane-can-scroll* is `nil`, then the CAPI is responsible for scrolling over the data range. The default value is `nil`. This is known as ordinary scrolling and there is an example in `output-panes/scroll-test.lisp`.

When the output pane is scrolled, the CAPI calls the *scroll-callback* if this is non-`nil`. The arguments of the scroll callback are the `output-pane`, the direction (`:vertical`, `:horizontal` or `:pan`), the scroll operation (`:move`, `:drag`, `:step` or `:page`), the amount of scrolling (an integer), and a keyword argument `:interactive`. This has value `t` if the scroll was

invoked interactively, and value `nil` if the scroll was programmatic, such as via the function `scroll`. In the Mac OS X Cocoa implementation the direction is always `:pan`. See the following CAPI example files:

```
output-panes/scroll-test.lisp
output-panes/scrolling-without-bar.lisp
graphics/scrolling-test.lisp
```

focus-callback, if non-`nil`, is a function of two arguments. The first argument is the `output-pane` itself, and the second is a boolean. When the `output-pane` gets the focus, *focus-callback* is called with second argument `t`, and when the `output-pane` loses the focus, *focus-callback* is called with second argument `nil`.

resize-callback, if non-`nil`, is a function of five arguments called when the `output-pane` is resized. The first argument is the `output-pane` itself, and the rest are its new geometry: *x*, *y*, *width* and *height*.

create-callback, if non-`nil`, is a function of one argument which is called just after the pane is created (but before it becomes visible). The argument is the pane itself. This function can perform initialization such as loading images.

destroy-callback, if non-`nil`, is a function of one argument which is called just before the pane is destroyed, for example when the window is closed or the pane is removed from its layout. The argument is the pane itself. This function can perform cleanup operations (though note that images associated with the pane are automatically freed).

graphics-options is currently only used by the Mac OS X Cocoa implementation. The single option defined is

`:text-rendering`, with allowed values:

<code>:glyph</code>	Draw glyphs directly using Core Graphics. This only draws characters with glyphs in the chosen font.
---------------------	--

`:atsui` Draw using ATSUI APIs where possible. This is slower but can handle more characters.

Examples Firstly, here is an example that draws a circle in an output pane.

```
(defun display-circle (self x y width height)
  (declare (ignore x y width height))
  (gp:draw-circle self 200 200 200 :filled t))

(capi:contain (make-instance
              'capi:output-pane
              :display-callback 'display-circle)
              :best-width 200 :best-height 200)
```

Here is an example that shows how to use a button gesture.

```
(defun test-callback (self x y)
  (capi:display-message
   "Pressed button 1 at (~S,~S) in ~S" x y self))

(capi:contain
 (make-instance
  'capi:output-pane
  :title "Press button 1:"
  :input-model `(((button-1 :press)
                    test-callback)))
 :best-width 200 :best-height 200)
```

This example illustrates gesture spec mappings.

```
(defun draw-input (self x y gspec)
  (let ((data (sys:gesture-spec-data gspec))
        (mods (sys:gesture-spec-modifiers gspec)))
    (gp:draw-string
     self
     (with-output-to-string (ss)
      (sys:print-pretty-gesture-spec
       gspec ss :force-shift-for-upcase nil))
     x y)))

(capi:contain
 (make-instance
  'capi:output-pane
  :title "Press keys in the pane..."
  :input-model '(:gesture-spec
                 draw-input)))
:best-width 200 :best-height 200)

(capi:contain
 (make-instance
  'capi:output-pane
  :title "Press Control-a in the pane..."
  :input-model '(:gesture-spec "Control-a"
                 draw-input)))
:best-width 200 :best-height 200)
```

Here is a simple example that draws the character typed at the cursor point.

```
(defun draw-character (self x y character)
  (gp:draw-character self character x y))

(capi:contain
 (make-instance
  'capi:output-pane
  :title "Press keys in the pane..."
  :input-model '(:character draw-character)))
:best-width 200 :best-height 200)
```

This example shows how to use the motion gesture.

```

(defun draw-red-blob (self x y)
  (gp:draw-circle self x y 3
    :filled t
    :foreground :red))

(capi:contain
 (make-instance
  'capi:output-pane
  :title "Drag button-1 across this pane."
  :input-model '(((:button-1 :motion)
                  gp:draw-point)
                ((:button-1 :motion :control)
                 draw-red-blob)))
 :best-width 200 :best-height 200)

```

This example illustrates the use of *focus-callback*:

```

(capi:contain
 (make-instance
  'capi:output-pane
  :focus-callback
  #'(lambda (x y)
      (format t
              "Pane ~a ~:[lost~;got~] the focus~%"
              x y))))

```

This example illustrates the use of *graphics-options* to specify ATSUI drawing on Cocoa:

```

(defvar *string*
  (coerce (loop for i from 0 below 60
               collect (code-char (* 5 i)))
    'text-string))

(capi:contain
 (make-instance 'capi:output-pane
  :visible-min-width 400
  :visible-max-height 50
  :display-callback
  #'(lambda (pane x y w h)
      (gp:draw-string pane
                       *string*
                       10 10))
  :graphics-options
  '(:text-rendering :atsui)))

```

There are further examples in the directory `examples/capi/output-panes/`.

See also `define-command`
`pinboard-object`
`scroll`

over-pinboard-object-p

Generic Function

Summary Tests whether a point lies within the boundary of a pinboard object.

Package `capi`

Signature `over-pinboard-object-p` *pinboard-object* *x* *y*

Description The generic function `over-pinboard-object-p` returns `non-nil` if the *x* and *y* coordinates specify a point within the boundary of a pinboard object. To find the actual object at this position, use `pinboard-object-at-position`.

The default method returns `t` if *x* and *y* are within the bounding area of the pinboard object. A method is supplied for `line-pinboard-object` and you may add methods for your own `pinboard-object` subclasses.

See also `pinboard-object-at-position`
`pinboard-object-overlap-p`
`pinboard-object`
`pinboard-layout`

page-setup-dialog

Function

Summary Displays the page setup dialog for a given printer.

Package	<code>capi</code>
Signature	<code>page-setup-dialog &key screen owner printer continuation</code>
Description	<p>The <code>page-setup-dialog</code> function displays the page setup dialog for <i>printer</i>. If <i>printer</i> is not specified, the dialog for the current printer is displayed.</p> <p>The CAPI screen on which to display the dialog is given by <i>screen</i>, which is the current screen by default.</p> <p><i>owner</i> specifies an owner window for the dialog. See the "Prompting for Input" chapter in the <i>LispWorks CAPI User Guide</i> for details.</p> <p>If <i>continuation</i> is non-<code>nil</code>, then it must be a function with a lambda list that accepts one argument. The <i>continuation</i> function is called with the values that would normally be returned by <code>page-setup-dialog</code>. On Cocoa, passing <i>continuation</i> causes the dialog to be made as a window-modal sheet and <code>display-dialog</code> returns immediately, leaving the dialog on the screen. The <code>with-dialog-results</code> macro provides a convenient way to create a <i>continuation</i> function.</p>
See also	<code>current-printer</code>

pane-adjusted-offset

Generic Function

Summary	The <code>pane-adjusted-offset</code> generic function calculates the offset required to place a pane correctly in a layout.
Package	<code>capi</code>
Signature	<code>pane-adjusted-offset pane adjust available-size actual-size &key &allow-other-keys</code>

Description This function calculates the offset required by the *adjust* keyword so that the pane *pane* is placed correctly within the available space in its parent layout. It is called by all of the layouts that inherit from `x-y-adjustable-layout` to interpret the values of *x-adjust* and *y-adjust*.

Typically the value of *adjust* will be a keyword or a list of the form (*keyword n*) where *n* is an integer. These values of *adjust* are interpreted as by `pane-adjusted-position`.

However, new methods can accept alternative values for *adjust* where required and can also add extra keywords. For example, `grid-layout` allows *adjust* to be a list of adjust values, and then passes the offset into this list as an additional keyword.

Example

```
(setq button-panel (make-instance 'capi:button-panel
                                  :items '(1 2 3)))

(capi:pane-adjusted-offset button-panel
                           :center 200 100)

(capi:pane-adjusted-offset button-panel
                           :left 200 100)

(capi:pane-adjusted-offset button-panel
                           :right 200 100)
```

See also `layout`
`x-y-adjustable-layout`

pane-adjusted-position

Generic Function

Summary Calculates how to place a pane correctly within a layout, given a minimum and maximum position.

Package `capi`

Signature `pane-adjusted-position pane adjust min-position max-position`
`&key &allow-other-keys`

Description The `pane-adjusted-position` generic function calculates the position required by the `adjust` argument so that the pane `pane` is placed correctly within the available space in its parent layout, given a minimum and maximum position. It is a complementary function to `pane-adjusted-offset`, and the default method actually calls `pane-adjusted-offset` with the gap between the two positions, and then adds on the minimum position to get the new position.

The default method accepts the following values for `adjust`.

- `:top` Place *pane* at the top of the region.
- `:bottom` Place *pane* at the bottom of the region.
- `:left` Place *pane* at the left of the region.
- `:right` Place *pane* at the right of the region.
- `:center` Place *pane* in the center of the region.
- `(:top n)` Place the top of *pane* *n* pixels below the top of the region.
- `(:bottom n)` Place the bottom of *pane* *n* pixels above the bottom of the region.
- `(:left n)` Place the left of *pane* *n* pixels after the left of the region.
- `(:right n)` Place the right of *pane* *n* pixels before the right of the region.
- `(:center n)` Place the center of *pane* *n* pixels below the center of the region.

However, new methods can accept alternative values for `adjust` where required and can also add extra keywords. For example, `grid-layout` allows `adjust` to be a list of adjust values, and then passes the offset into this list as an additional keyword. It is preferable to add new methods to `pane-adjusted-offset` as these changes will be seen by the default method of `pane-adjusted-position`.

```

Example      (setq button-panel (make-instance 'capi:button-panel
                                         :items '(1 2 3)))

              (capi:pane-adjusted-position button-panel
                                         :center 100 200)

              (capi:pane-adjusted-position button-panel
                                         :right 100 200)

              (capi:pane-adjusted-position button-panel
                                         :left 100 200)
    
```

```

See also    layout
            graph-pane
            x-y-adjustable-layout
    
```

pane-close-display

Function

Summary Closes the X display of a pane.

Package `capi`

Signature `pane-close-display pane => closedp`

Arguments *pane* A CAPI element.

Values *closedp* A boolean.

Description The function `pane-close-display` closes the X display connection on which *pane* is currently displayed. This destroys all the other panes on the same connection.

closedp is true if the connection was closed.

Note: `pane-close-display` works in the X11/Motif implementation only, and not on Microsoft Windows.

pane-got-focus

Generic Function

Summary	A function called when the focus is set programmatically.	
Package	<code>capi</code>	
Signature	<code>pane-got-focus</code> <i>interface</i> <code>pane</code>	
Arguments	<i>interface</i>	The interface of <i>pane</i> .
	<i>pane</i>	A CAPI element.
Description	<p>The generic function <code>pane-got-focus</code> is called just before the focus is set by <code>set-object-automatic-resize</code>.</p> <p>The supplied primary method does nothing. You may add methods on your own interface classes, which can be useful for example when the focus is set programmatically to a pane which is hidden inside a <code>tab-layout</code> or <code>switchable-layout</code>. Your method can check for this case and modify the layout as required.</p>	
See also	<code>set-object-automatic-resize</code>	

pane-has-focus-p

Function

Summary	Determines whether a pane has the focus.	
Package	<code>capi</code>	
Signature	<code>pane-has-focus-p</code> <i>pane</i> => <i>focusp</i>	
Arguments	<i>pane</i>	A CAPI element.
Values	<i>focusp</i>	A boolean.

Description	<p>The function <code>pane-has-focus-p</code> is the predicate for whether <i>pane</i> currently has the input focus.</p> <p>Note: On Motif, <code>pane-has-focus-p</code> cannot be used in menu functions such as the <i>enabled-function</i> or <i>popup-callback</i> of a menu item. It will always return <code>nil</code>, because the focus is on the menu button when the user clicks on it.</p>
Example	<p>This function descends down a layout's hierarchy to find the child that currently has the focus:</p> <pre>(defun find-pane-with-focus (layout) (capi:map-pane-descendant-children layout #'(lambda (p) (when (capi:pane-has-focus-p p) (return-from find-pane-with-focus p))))))</pre>
See also	<code>accepts-focus-p</code>

pane-initial-focus*Generic Function*

Summary	Gets or sets the initial focus pane.
Package	<code>capi</code>
Signature	<code>pane-initial-focus</code> <i>pane-with-children</i> => <i>pane</i>
Signature	<code>(setf pane-initial-focus)</code> <i>pane</i> <i>pane-with-children</i> => <i>pane</i>
Arguments	<i>pane-with-children</i> A pane with children.
Values	<i>pane</i> A child of <i>pane-with-children</i> .
Description	The generic function <code>pane-initial-focus</code> returns the child of <i>pane-with-children</i> that has the input focus when <i>pane-with-children</i> is first displayed.

(`setf pane-initial-focus`) may be used to set the initial focus pane, but only before *pane-with-children* has been created. If the setter is called after *pane-with-children* has been created, an error is signalled.

pane-with-children should be a pane with child panes such as a layout, an interface, a button-panel or a toolbar.

See also `pane-has-focus-p`

pane-popup-menu-items

Generic Function

Summary	Generates the items for the menu associated with a pane.	
Package	<code>capi</code>	
Signature	<code>pane-popup-menu-items pane interface => items</code>	
Arguments	<i>pane</i>	A pane in interface <i>interface</i> .
	<i>interface</i>	An interface.
Values	<i>items</i>	A list in which each element is a <code>menu-item</code> , <code>menu-component</code> or <code>menu</code> .
Description	<p>The generic function <code>pane-popup-menu-items</code> generates the items for the menu associated with the pane <i>pane</i>. The default method of <code>make-pane-popup-menu</code> calls <code>pane-popup-menu-items</code> to find the items for the menu. If <code>pane-popup-menu-items</code> returns <code>nil</code>, then <code>make-pane-popup-menu</code> returns <code>nil</code>.</p> <p>To specify items for menus associated with panes in your interfaces, define <code>pane-popup-menu-items</code> methods specialized on your interface class.</p>	

For most supplied CAPI pane classes, the system method returns `nil`. The exceptions are `editor-pane` and `graph-pane`. To inherit the items from the system method (or other more general method), call `call-next-method`.

Note: `pane-popup-menu-items` is intended to allow multiple calls on the same pane, to generate menus in different places (as in the example in `make-pane-popup-menu`). Therefore the `menu-objects` that it returns, and their descendent `menu-objects`, must be constructed each time that `pane-popup-menu-items` is called, so that no two menus share any menu item.

Note: the *items* returned by `pane-popup-menu-items` may specify the arguments for their callbacks, but it is not required. If they do not specify the arguments, then `make-pane-popup-menu` (by calling `make-menu-for-pane`) sets up the callbacks such that they are called on the pane *pane*.

Example

The methods below specialized on interface class `eg`:

1. Append the items that were returned by the system method in the bottom of the menu for the `editor-pane`, and
2. Add them as a sub-menu for the menu of the `graph-pane`.

```

(capi:define-interface edgraph ()
  ()
  (:panes
   (e1 capi:editor-pane)
   (g1 capi:graph-pane))
  (:layouts
   (main-layout capi:column-layout '(e1 g1)))
  (:menu-bar )
  (:default-initargs
   :visible-min-width 200
   :visible-min-height 300))

(defun my-callback (pane)
  (capi:display-message "Callback on pane ~S." pane))

(defmethod capi:pane-popup-menu-items
  ((self capi:editor-pane) (interface edgraph))
  (list*
   (make-instance 'capi:menu-item
                  :title "Item for My Editor Menu."
                  :selection-callback 'my-callback)
   (call-next-method)))

(defmethod capi:pane-popup-menu-items
  ((self capi:graph-pane) (interface edgraph))
  (list
   (make-instance 'capi:menu-item
                  :title "Item for My Graph Menu."
                  :selection-callback 'my-callback)
   (capi:make-menu-for-pane self (call-next-method)
                            :title "Default Graph Menu")))

(capi:display (make-instance 'edgraph))

```

See also `make-pane-popup-menu`

pane-string

Generic Function

Summary Returns the text displayed in an `editor-pane`.

Package `capi`

Signature `pane-string pane => text`

Arguments	<i>pane</i>	An <code>editor-pane</code> .
Values	<i>text</i>	A string.
Description	The generic function <code>pane-string</code> returns as a string the text of the buffer that is currently displayed in the <code>editor-pane</code> <i>pane</i> .	
See also	<code>editor-pane</code>	

parse-layout-descriptor

Generic Function

Summary	Returns the geometry object associated with a layout's child.	
Package	<code>capi</code>	
Signature	<code>parse-layout-descriptor</code> <i>child-descriptor</i> <i>interface</i> <i>layout</i>	
Description	<p>The generic function <code>parse-layout-descriptor</code> takes a description of a layout's child, and returns the geometry object associated with that child. It is called by <code>interpret-description</code> to parse individual children in a layout.</p> <p>The default method accepts a <i>child-desc</i> argument which can be a pane (subclass of <code>simple-pane</code> or <code>pinboard-object</code>), a geometry object, or a symbol naming a slot in the interface which contains such a pane.</p>	
See also	<code>interpret-description</code> <code>define-layout</code> <code>layout</code>	

password-pane

Class

Summary The password pane is a pane designed for entering passwords, such that when the password is entered it is not visible on the screen.

Package `capi`

Superclasses `text-input-pane`

Initargs `:overwrite-character`
 A `base-char`.

Readers `password-pane-overwrite-character`

Description The password pane inherits most of its functionality from `text-input-pane`. It starts with the initial text and caret position specified by the arguments *text* and *caret-position* respectively, and limits the number of characters entered with the *max-characters* argument (which defaults to `nil`, meaning there is no maximum).

The password pane can be enabled and disabled with the `text-input-pane` accessor `text-input-pane-enabled`.

overwrite-character is a `base-char` which is the character to display instead of the real characters. The default value of *overwrite-character* is `#*`.

Examples

```
(setq password-pane (capi:contain
                      (make-instance
                       'capi:password-pane
                       :callback
                       #'(lambda (password interface)
                           (capi:display-message
                            "Password: ~A"
                            password))))))

(capi:text-input-pane-text password-pane)
```

```
(setq password-pane
  (capi:contain
    (make-instance 'capi:password-pane
      :max-characters 5
      :text "abc"
      :overwrite-character #\$)))

(capi:password-pane-overwrite-character password-pane2)
```

See also `editor-pane`
`text-input-pane`

play-sound

Function

Summary `Plays a loaded sound.`

Package `capi`

Signature `play-sound sound &key wait`

Arguments `sound` A sound object returned by `load-sound`.
`wait` A generalized boolean.

Description The function `play-sound` plays the loaded sound `sound`.
If `wait` is true then `play-sound` will not return until `sound` has finished playing. That is, it plays the sound synchronously. The default value of `wait` is `nil`.

Note: `:wait t` is only implemented on Microsoft Windows.

See also `load-sound`

pinboard-layout

Class

Summary	The class <code>pinboard-layout</code> provides two very useful pieces of functionality for displaying CAPI windows. Firstly it is a layout that allows its children to be positioned anywhere within itself (like a pinboard). Secondly it supports <code>pinboard-objects</code> which are rectangular areas within the layout which have size and drawing functionality.
Package	<code>capi</code>
Superclasses	<code>output-pane</code> <code>layout</code>
Subclasses	<code>simple-pinboard-layout</code>
Initargs	<code>:highlight-style</code> A keyword. <code>:fit-size-to-children</code> A generalized boolean.
Description	<p>When a <code>pinboard-layout</code> lays out its children, it positions them at the <code>x</code> and <code>y</code> specified as hints (using <code>:x</code> and <code>:y</code>), and sizes them to their minimum size (which can be specified using <code>:visible-min-width</code> and <code>:visible-max-width</code>).</p> <p>If <code>fit-size-to-children</code> is true, the <code>pinboard-layout</code> is made sufficiently large to accomodate all of its children, and grows if necessary when a child is added. This is the default behavior. Otherwise the pinboard layout has a minimum size of one pixel by one pixel which is not affected by the size of its children. If you need the sizing capabilities, then use the class <code>simple-pinboard-layout</code> which surrounds a single child, and adopts the size constraints of that child.</p> <p>The pinboard layout handles the display of pinboard objects itself by calculating which objects are visible in the region that needs redrawing, and then by calling the generic func-</p>

tion `draw-pinboard-object` on these objects in the order that they are specified in the layout description. This means that if two pinboard objects overlap, the later one in the layout description will be on top of the other one. In other words, the description defines the Z-order for objects of type `pinboard-object`. For information about controlling this order, see `layout` and `manipulate-pinboard`.

Note: objects of type `simple-pane` are drawn directly by the windowing system and cannot be clipped relative to `pinboard-objects`, which are drawn by CAPI. Therefore `simple-panes` always appear on top in a pinboard, and their position in the description does not affect the Z-order.

Highlighting of the layout's children by `highlight-pinboard-object` is controlled by the value of `highlight-style`, as follows:

<code>:invert</code>	Swaps the foreground and background colors.
<code>:standard</code>	Uses system colors.
<code>:default</code>	Calls <code>draw-pinboard-object-highlighted</code> .

The default value of `highlight-style` is `:default`.

Example

Here is an example of a pinboard layout placing simple panes at arbitrary positions inside itself.

```
(capi:contain
 (make-instance
  'capi:pinboard-layout
  :description
  (list (make-instance
         'capi:text-input-pane
         :x 20
         :y 100)
        (make-instance
         'capi:push-button-panel
         :x 30
         :y 200
         :items '(1 2 3))))
 :best-width 300 :best-height 300)
```

Here are some examples of the use of pinboard objects with pinboard layouts.

```
(capi:contain
 (make-instance
  'capi:pinboard-layout
  :description
  (list
   (make-instance
    'capi:image-pinboard-object
    :image
    (sys:lispworks-file
     "examples/capi/graphics/lwsplash.bmp")
    :x 20 :y 20)))
 :best-width 540 :best-height 415)

(capi:contain
 (make-instance
  'capi:pinboard-layout
  :description (list
                (make-instance
                 'capi:item-pinboard-object
                 :text "Hello"
                 :x 40 :y 10)
                (make-instance
                 'capi:line-pinboard-object
                 :x 10 :y 30
                 :visible-min-width 100)))
 :best-width 200 :best-height 200)
```

There are further examples in the directories `examples/capi/applications/` and `examples/capi/graphics/`.

See also `manipulate-pinboard`
`pinboard-object`
`redraw-pinboard-object`

pinboard-object

Class

Summary Provides a rectangular area in a `pinboard-layout` with drawing capabilities.

Package	<code>capi</code>	
Superclasses	<code>capi-object</code>	
Subclasses	<code>item-pinboard-object</code> <code>image-pinboard-object</code> <code>line-pinboard-object</code> <code>drawn-pinboard-object</code> <code>rectangle</code>	
Initargs	<code>:pinboard</code>	The output pane on which the pinboard object is drawn.
	<code>:activep</code>	If <code>t</code> , the pinboard object is made active.
	<code>:graphics-args</code>	A plist of Graphics Ports drawing options.
	<code>:help-key</code>	An object used for lookup of help. Default value <code>t</code> .
		The following initargs are geometry hints, influencing the initial size and position of a <code>pinboard-object</code> :
	<code>:x</code>	The x position of the pinboard object in the pinboard.
	<code>:y</code>	The y position of the pinboard object in the pinboard.
	<code>:external-min-width</code>	The minimum width of the pinboard object in the pinboard.
	<code>:external-min-height</code>	The minimum height of the pinboard object in the pinboard.
	<code>:external-max-width</code>	The maximum width of the pinboard object in the pinboard.

`:external-max-height`
 The maximum height of the pinboard object in the pinboard.

`:visible-min-width`
 The minimum visible width of the pinboard object.

`:visible-min-height`
 The minimum visible height of the pinboard object.

`:visible-max-width`
 The maximum visible width of the pinboard object.

`:visible-max-height`
 The maximum height of the pinboard object.

`:internal-min-width`
 The minimum width of the display region.

`:internal-min-height`
 The minimum height of the display region.

`:internal-max-width`
 The maximum width of the display region.

`:internal-max-height`
 The maximum height of the display region.

Accessors `pinboard-object-pinboard`
`pinboard-object-activep`
`pinboard-object-graphics-args`

Readers `help-key`

Description The class `pinboard-object` provides a rectangular area in a `pinboard-layout` with drawing and highlighting capabilities. A pinboard object behaves just like a simple pane within layouts, meaning that they can be placed into rows, columns and other layouts, and that they size them-

selves in the same way. The main distinction is that a pinboard object is a much smaller object than a simple pane as it does not need to create a native window for itself.

Each pinboard object is placed into a pinboard layout (or into a layout itself inside a pinboard layout), and then when the pinboard layout wishes to redisplay a region of itself, it calls the function `draw-pinboard-object` on each of the pinboard objects that are contained in that region (in the order that they are specified as children to the layout).

The *graphics-args* slot allows drawing options to be set. These include the font, the background and foreground colors, and others (see the section Graphics State in the *LispWorks CAPI User Guide* for details).

help-key and the geometry hints are interpreted as described for `element`.

By default a `pinboard-object` does not accept the input focus.

There are a number of predefined pinboard objects provided by the CAPI. They are as follows:

`item-pinboard-object`

Draws a title.

`line-pinboard-object`

Draws a line.

`right-angle-line-pinboard-object`

Draws a right-angled line.

`image-pinboard-object`

Draws an image.

`drawn-pinboard-object`

Uses a user-defined display function.

The main user of pinboard objects in the CAPI is the graph pane, which uses `item-pinboard-object` and `line-pinboard-object` to display its nodes and edges respectively.

To force a pinboard object to redraw itself, either call the function `invalidate-rectangle` on it (in which case the redrawing is done immediately), or call `redraw-pinboard-object` in which case the redrawing may be cached and displayed at a later date.

Call the generic functions `highlight-pinboard-object` and `unhighlight-pinboard-object` to highlight a pinboard and remove its highlighting.

You can control automatic resizing of a pinboard object using `set-object-automatic-resize`.

Examples See the file `examples/capi/graphics/pinboard-test.lisp`.

See also `pinboard-layout`
`draw-pinboard-object`
`graph-pane`
`highlight-pinboard-object`
`redraw-pinboard-object`
`redraw-pinboard-layout`
`unhighlight-pinboard-object`

pinboard-object-at-position

Generic Function

Summary The generic function `pinboard-object-at-position` returns the uppermost pinboard object containing a specified point.

Package `capi`

Signature `pinboard-object-at-position pinboard x y`

Description	This function returns the uppermost pinboard object in the pinboard that contains the point specified by <i>x</i> and <i>y</i> . It determines this by mapping over every pinboard object within the pinboard until it finds one for which the generic function <code>over-pinboard-object-p</code> returns <code>t</code> .
Example	<pre>(setq pinboard (capi:contain (make-instance 'capi:pinboard-layout) :best-width 300 :best-height 300)) (make-instance 'capi:item-pinboard-object :text "Hello world" :x 100 :y 100 :parent pinboard) (capi:pinboard-object-at-position pinboard 0 0) (capi:pinboard-object-at-position pinboard 110 110)</pre>
See also	<p><code>over-pinboard-object-p</code> <code>pinboard-object-overlap-p</code> <code>pinboard-object</code> <code>pinboard-layout</code></p>

pinboard-object-graphics-arg*Generic Function*

Summary	Gets or sets the value of a particular drawing option in a <code>pinboard-object</code> .
Package	<code>capi</code>
Signature	<code>pinboard-object-graphics-arg</code> <i>self</i> <i>keyword</i> => <i>value</i>
Signature	<code>(setf pinboard-object-graphics-arg)</code> <i>value</i> <i>self</i> <i>keyword</i> => <i>value</i>
Arguments	<i>self</i> A <code>pinboard-object</code> .

	<i>keyword</i>	A keyword denoting a Graphics Ports drawing option.
Values	<i>value</i>	The value of the drawing option <i>keyword</i> in <i>self</i> .
Description		The generic function <code>pinboard-object-at-position</code> returns or sets the value of the Graphics Ports drawing option <i>keyword</i> in <i>self</i> . See the section Graphics State in the <i>LispWorks CAPI User Guide</i> for details of the drawing options.
See also	<code>pinboard-object</code>	

pinboard-object-overlap-p

Generic Function

Summary		The generic function <code>pinboard-object-overlap-p</code> returns <code>non-nil</code> if a specified region overlaps with the region of a specified pinboard object.
Package	<code>capi</code>	
Signature	<code>pinboard-object-overlap-p</code>	<i>pinboard-object</i> <i>top-left-x top-left-y</i> <i>bottom-right-x bottom-right-y</i>
Description		Returns <code>non-nil</code> if the specified region overlaps with the region of the pinboard object.
See also	<code>pinboard-object-at-position</code> <code>over-pinboard-object-p</code> <code>pinboard-object</code> <code>pinboard-layout</code>	

pinboard-pane-position*Generic Function*

Summary	Gets and sets the location of an object inside its parent <code>pinboard-layout</code> .	
Package	<code>capi</code>	
Signature	<code>pinboard-pane-position self => x, y</code>	
Signature	<code>(setf pinboard-pane-position) x, y self => x, y</code>	
Arguments	<i>self</i>	A <code>pinboard-object</code> or <code>simple-pane</code> .
Values	<i>x, y</i>	The horizontal and vertical coordinates in the <code>pinboard-layout</code> parent of <i>self</i> .
Description	<p>The generic function <code>pinboard-pane-position</code> returns as multiple values <i>x, y</i> the coordinates of <i>self</i> inside its parent <code>pinboard-layout</code>.</p> <p><code>(setf pinboard-pane-position)</code> sets the location of <i>self</i> in its parent.</p>	

Examples

```
(let* ((po (make-instance 'capi:item-pinboard-object
                          :text "5x5" :x 5 :y 5
                          :graphics-args
                          '(:background :red)))
      (pl (capi:contain
           (make-instance 'capi:pinboard-layout
                          :description (list po)
                          :visible-min-width 200
                          :visible-min-height 200))))
      (capi:execute-with-interface
       (capi:element-interface pl)
       #'(lambda (po)
          (dotimes (x 20)
            (mp:wait-processing-events 1)
            (let ((new-x (* (1+ x) 10))
                  (new-y (* 5 (+ 2 x))))
              (setf (capi:item-text po)
                    (format nil "~ax~a" new-x new-y))
              (setf (capi:pinboard-pane-position po)
                    (values new-x new-y))))))
       po))
```

See also `pinboard-layout`
`pinboard-pane-size`

pinboard-pane-size

Generic Function

Summary Gets and sets the size of an object inside its parent `pinboard-layout`.

Package `capi`

Signature `pinboard-pane-size self => width, height`

Signature `(setf pinboard-pane-position) width, height self
=> width, height`

Description The generic function `pinboard-pane-size` returns as multiple values *width, height* the dimensions of *self*.

`(setf pinboard-pane-size)` sets the dimensions of *self*.

```

Example      (let* ((po (make-instance 'capi:pinboard-object
                               :x 5 :y 5
                               :width 5 :height 5
                               :graphics-args
                               '(:background :red)))
              (pl (capi:contain
                   (make-instance 'capi:pinboard-layout
                                   :description (list po)
                                   :visible-min-width 200
                                   :visible-min-height 200))))
              (capi:execute-with-interface
               (capi:element-interface pl)
               #'(lambda(po)
                   (dotimes (x 20)
                     (mp:wait-processing-events 1)
                     (let ((new-x (* (1+ x) 10))
                           (new-y (* 5 (+ 2 x))))
                       (setf (capi:pinboard-pane-size po)
                             (values new-x new-y))))
                     po))

```

See also `pinboard-layout`
`pinboard-pane-position`

popup-confirmer

Function

Summary The `popup-confirmer` function creates a dialog with pre-defined implementations of **OK** and **Cancel** buttons and a user specified pane in a layout with the buttons.

Package `capi`

Signature `popup-confirmer` *pane message &rest interface-args &key modal title title-font value-function exit-function apply-function apply-check apply-button ok-function ok-check ok-button no-button no-function all-button all-function cancel-button help-button help-function buttons print-function callbacks callback-type button-position buttons-uniform-size-p foreground background font screen focus owner x y position-relative-to button-container button-font continuation*
=> *result, successp*

Arguments

pane A CAPI pane or interface.

message A string or nil.

modal, screen, focus, owner, x, y, and position-relative-to
 These are passed to `display-dialog`.

title A string specifying the title of the dialog window.

title-font The font used in the title.

value-function Controls the value returned, and whether a value can be returned.

exit-function Called on exiting the dialog.

apply-function, apply-check, apply-button
 Define the callback, check function and title of an **Apply** button.

ok-function, ok-check, ok-button
 Define the callback, check function and title of an **OK** button.

no-button, no-function
 Define the title and callback of a **No** button.

all-button, all-function
 Define the title and callback of an **All** button.

	<i>cancel-button</i>	Defines the title of a Cancel button.
	<i>help-button, help-function</i>	Define the title and callback of a Help button.
	<i>buttons</i>	Defines extra buttons.
	<i>print-function</i>	Displays <i>ok-button</i> , <i>no-button</i> , <i>cancel-button</i> , <i>apply-button</i> and <i>all-button</i> as button titles.
	<i>callbacks</i>	Defines callbacks for <i>buttons</i> .
	<i>callback-type</i>	Specifies the callback-type of <i>buttons</i> .
	<i>button-position</i>	One of <code>:bottom</code> , <code>:top</code> , <code>:left</code> , <code>:right</code> .
	<i>buttons-uniform-size-p</i>	Controls relative button sizes.
	<i>foreground, background</i>	Specify colors.
	<i>font</i>	A font or a font description.
	<i>button-font</i>	A font or a font description.
	<i>button-container</i>	A layout controlling where the buttons of the dialog appear.
	<i>continuation</i>	A function or <code>nil</code> .
Values	<i>result</i>	The result of <i>value-function</i> , or <i>pane</i> , or <code>nil</code> .
	<i>successp</i>	<code>nil</code> if the dialog was cancelled, <code>t</code> otherwise.
Description	<p>The function <code>popup-confirmer</code> provides the quickest means to create new dialogs, as it will create and implement OK, Cancel and other buttons as required by your dialog, and will place a user-specified pane in a layout along with the buttons.</p> <p>Generally the <code>Return</code> key selects the dialog's OK button and the <code>Escape</code> key selects the Cancel button, if there is one.</p>	

The argument *value-function* should provide a callback which is passed *pane* and should return the value to return from `popup-confirmer`. If *value-function* is not supplied, then *pane* itself will be returned as *result*. If the *value-function* wants to indicate that the dialog cannot return a value currently, then it should return a second value that is `non-nil`.

The *ok-check* function is passed the result returned by the *value-function* and should return `non-nil` if it is acceptable for that value to be returned. These two functions are used by `popup-confirmer` to decide when the **OK** button should be enabled, thus stopping the dialog from returning with invalid data. The **OK** button's state can be updated by a call to `redisplay-interface` on the top-level, so the dialog should call it when the button may enable or disable.

The arguments *ok-button*, *no-button* and *cancel-button* are the text strings for each button, or `nil` meaning do not include that button. The *ok-button* returns successfully from the dialog (with the result of *value-function*), the *no-button* means continue but return `nil`, and the *cancel-button* aborts the dialog. Note that there are clear expectations on the part of users as to the functions of these buttons — check the style guidelines of the platform you are developing for.

apply-button, if passed, specifies the title of an extra button which appears near to the **OK** button. *apply-check* and *apply-function* define its functionality.

all-button, if passed, specifies the title of an extra button which is always enabled and which appears near to the *apply-button* (if that exists) or the **OK** button. *all-function* defines its functionality.

help-button, if passed, specifies the title of a help button which appears to the right of the **Cancel** button. *help-function* defines its functionality.

print-function is called on the various *button* arguments to generate a string to display for each button title.

button-position specifies where to put the buttons. The default is `:bottom`.

buttons-uniform-size-p specifies whether the buttons are all the same size, regardless of the text on them. The default is `t`, but `nil` can be passed to make each button only as wide as its text.

foreground and *background* specify colors to use for the parts of the dialog other than *pane*, including the buttons

font specifies the font to use in the *message*.

button-font specifies the font to use in the buttons.

button-container indicates where the buttons of the dialog appear. It must be a layout which is a descendent of *pane*. The description of this layout is automatically set to the button-panel containing the buttons.

The arguments *exit-function*, *ok-function* and *no-function* are the callbacks that get done when exiting, pressing **OK** and pressing **No** respectively. The *exit-function* defaults to `exit-confirmer`, the *ok-function* defaults to the *exit-function* and the *no-function* defaults to a function exiting the dialog with `nil`.

The arguments *buttons*, *callbacks* and *callback-type* are provided as a means of extending the available buttons. The buttons provided by *buttons* will be placed after the buttons generated by `popup-confirmer`, with the functions in *callbacks* being associated with them. Finally *callback-type* will be provided as the callback type for the buttons.

If any of *callbacks* need to access *pane*, you could use `confirmer-pane` together with a *callback-type* that passes the interface.

If *continuation* is non-`nil`, then it must be a function with a lambda list that accepts two arguments. The *continuation* function is called with the values that would normally be returned by `popup-confirmer`. On Cocoa, passing *continuation* causes the dialog to be made as a window-modal sheet

and `popup-confirmer` returns immediately, leaving the dialog on the screen. The `with-dialog-results` macro provides a convenient way to create a *continuation* function.

All other arguments will be passed to the call to `make-instance` for the interface that will be displayed using `display-dialog`. Thus geometry information, colors, and so on can be passed in here as well. By default, the dialog will pick up the foreground, background and font of *pane*.

Examples

Here are two simple examples which implement the basic functionality of two CAPI prompters: the first implements a simple `prompt-for-string`, while the second implements `prompt-for-confirmation`.

```
(capi:popup-confirmer
  (make-instance 'capi:text-input-pane
                :callback
                  'capi:exit-confirmer)
  "Enter some text:"
  :value-function 'capi:text-input-pane-text)

(capi:popup-confirmer nil
  "Yes or no?"
  :callback-type :none
  :ok-button "Yes"
  :no-button "No"
  :cancel-button nil
  :value-function #'(lambda (dummy) t))
```

This example demonstrates the use of `:redisplay-interface` to make the **OK** button enable and disable on each keystroke.

```
(defun pane-integer (pane)
  (ignore-errors (values
                  (read-from-string
                   (capi:text-input-pane-text
                    pane))))))
```

```
(capi:popup-confirmer
  (make-instance 'capi:text-input-pane
                 :callback 'capi:exit-confirmer
                 :change-callback :redisplay-interface)
  "Enter an integer"
  :value-function 'pane-integer
  :ok-check 'integerp)
```

An example illustrating the use of `:button-container`:

```
(let* ((bt (make-instance 'capi:simple-layout
                          :title "Button Container"
                          :title-position :left))
      (tip1 (make-instance 'capi:text-input-pane
                           :title "Top"))
      (tip2 (make-instance 'capi:text-input-pane
                           :title "Bottom")))
  (layout (make-instance 'capi:column-layout
                        :description
                        (list tip1
                              bt
                              tip2))))
(capi:popup-confirmer layout nil
  :title
  "Dialog using button-container"
  :button-container bt))
```

An example with all the defined buttons in use:

```

(defun all-buttons-dialog (&optional (num 20))
  (let ((pane
        (make-instance 'capi:list-panel
                        :items
                        (loop for ii from 1
                              to num
                              collect
                              (format nil "~r" ii))
                        :visible-min-width
                        '(character 20))))
    (capi:popup-confirmer
     pane
     "All Buttons"
     :callback-type :none
     :button-position :right
     :cancel-button "Cancel Button"
     :ok-button "OK Button"
     :ok-function #'(lambda (x)
                      (declare (ignorable x))
                      (capi:exit-dialog
                       (capi:choice-selected-item pane)))
     :no-button "No Button"
     :no-function
     #'(lambda ()
         (capi:exit-dialog
          (cons :no
                (capi:choice-selected-item pane))))
     :apply-button "Apply Button"
     :apply-function
     #'(lambda ()
         (capi:display-message
          "Applying to ~a"
          (capi:choice-selected-item pane)))
     :help-button "Help Button"
     :help-function
     #'(lambda ()
         (capi:display-message
          "~a is ~:[an odd~;an even~] number"
          (capi:choice-selected-item pane)
          (oddp (capi:choice-selection pane))))
     :all-button "All Button"
     :all-function
     #'(lambda()
         (capi:exit-dialog
          (capi:collection-items pane))))))
  (all-buttons-dialog))

```

A dialog with arbitrary buttons:

```
(capi:popup-confirmer
 (make-instance 'capi:text-input-pane)
 "Dialog with arbitrary buttons"
 :buttons '(:abc :xyz)
 :callbacks
 (list #'(lambda (data)
          (capi:display-message
           "Button ~A was pressed" data))
        #'(lambda (data)
          (capi:display-message
           "Button with ~A was pressed, exiting with
~S" data data)
          (capi:exit-dialog data))))
 :callback-type :data)
```

See also

- abort-dialog
- abort-exit-confirmer
- confirmer-pane
- display-dialog
- exit-confirmer
- exit-dialog

popup-menu-button

Class

Summary	A button with a popup menu.	
Package	capi	
Superclasses	item	
Initargs	:menu	A menu or nil.
	:menu-function	A function designator or nil.
Accessors	popup-menu-button-menu	
	popup-menu-button-menu-function	

Description	<p>The class <code>popup-menu-button</code> provides a button with a popup menu, which is displayed when the user clicks on the button.</p> <p>If <i>menu-function</i> is non-<code>nil</code>, it should be function of one argument (the pane) and should return a <code>menu</code> object. Otherwise, <i>menu</i> should be a <code>menu</code> object.</p> <p><code>popup-menu-button</code> inherits from <code>item</code>, so you can supply <i>text</i>, <i>data</i> and so on.</p>
Example	See the example in <code>capi/elements/popup-menu-button.lisp</code>
See also	<code>menu</code>

print-capi-button

Generic Function

Summary	Generates the text for a button.
Package	<code>capi</code>
Signature	<code>print-capi-button <i>button</i> => <i>text</i></code>
Arguments	<i>button</i> A button.
Values	<i>text</i> A string.
Description	<p>The generic function <code>print-capi-button</code> is used to generate the text for a button.</p> <p>You can add methods for your own button classes.</p>
See also	<code>button</code>

print-collection-item*Generic Function*

Summary	Prints an item as a string.
Package	<code>capi</code>
Signature	<code>print-collection-item</code> <i>item</i> <i>collection</i>
Arguments	<p><i>item</i> An <code>item</code> or an Lisp object.</p> <p><i>collection</i> A <code>collection</code> or any Lisp object.</p>
Description	<p>The generic function <code>print-collection-item</code> prints <i>item</i> as a string. It is used when <i>item</i> is known to be an item in <i>collection</i>.</p> <p>An <code>item</code> in a collection prints using the first of these which returns non-<code>nil</code>: the item's <i>text</i>, the item's <i>print-function</i>, the collection's <i>print-function</i> or the item's <i>data</i>. An <code>item</code> not known to be in the collection is printed simply using <code>print-object</code>.</p> <p>The method on <code>(t collection)</code> uses the collection's <i>print-function</i>.</p>
Example	<pre>(setq collection (make-instance 'capi:collection :items '(1 2 3 4 5) :print-function #'(lambda (x) (format nil "<~A:>" x)))) (capi:print-collection-item 2 collection)</pre> <p>In this example we provide our own <code>print-collection-item</code> method:</p>

```
(defclass my-tree-view (capi:tree-view) ())

(defmethod capi:print-collection-item ((item capi:item)
                                       (tree my-tree-view))
  (string-capitalize (svref (capi:item-data item) 0)))

(capi:contain
 (make-instance 'my-tree-view
                :roots
                (list (make-instance 'capi:item
                                     :data
                                     (vector "foo")))))
```

See also `get-collection-item`
`collection`

print-dialog

Function

Summary Displays a print dialog and returns a printer object.

Package `capi`

Signature `print-dialog &key screen owner first-page last-page
print-selection-p print-pages-p print-copies-p
continuation => printer`

Values *printer* A printer, or `nil`.

Description The function `print-dialog` displays a print dialog and returns a printer object. The printer object returned will print multiple copies if requested by the user.

If *print-pages-p* is `t`, the user can select a range of pages to print. This should always be the case unless the application only produces single page output. If *print-pages* is `t`, *first-page* and *last-page* can be used to initialize the page range. For example, they could be set to be the first and last pages of the document.

The *print-copies-p* argument indicates whether the application handles production of multiple copies for drivers that do not support this function. Currently this should be `nil` if the application uses Page Sequential printing and `t` if the application uses Page on Demand printing.

If *print-selection-p* is `t`, the user is given the option of printing the current selection. Only specify this if the application has a notion of selection and selecting printing functionality is provided.

The dialog is displayed on the current screen unless *screen* specifies otherwise.

owner specifies an owner window for the dialog. See the "Prompting for Input" chapter in the *LispWorks CAPI User Guide* for details.

If *continuation* is non-`nil`, then it must be a function with a lambda list that accepts one argument. The *continuation* function is called with the values that would normally be returned by `print-dialog`. On Cocoa, passing *continuation* causes the dialog to be made as a window-modal sheet and `print-dialog` returns immediately, leaving the dialog on the screen. The `with-dialog-results` macro provides a convenient way to create a *continuation* function.

Note that the printer object itself is opaque but programmatic setting of some printer options is available via the function `set-printer-options`.

See also `print-file`
`print-text`
`set-printer-options`

print-editor-buffer

Function

Summary Prints the contents of an editor buffer to the printer.

Package	<code>capi</code>
Signature	<code>print-editor-buffer</code> <i>buffer</i> &key <i>start end printer interactive font</i>
Description	<p>The <code>print-editor-buffer</code> function prints the contents of <i>buffer</i> to <i>printer</i>, which is the current printer by default.</p> <p>By default the entire editor buffer is printed, but by specifying <i>start</i> and <i>end</i> to be editor points, a part of the buffer can be printed. See the <i>LispWorks Editor User Guide</i> for information about editor points.</p> <p>If <i>interactive</i> is <code>t</code>, the default value, then a printer dialog is displayed.</p> <p><i>font</i> is interpreted as described for <code>print-text</code>.</p>
See also	<p><code>print-file</code></p> <p><code>print-text</code></p>

print-file

Function

Summary	Prints the contents of a specified file.
Package	<code>capi</code>
Signature	<code>print-file</code> <i>file</i> &key <i>printer interactive font</i>
Description	<p>The <code>print-file</code> function prints <i>file</i> to <i>printer</i>, which defaults to the current printer. If <i>interactive</i> is <code>t</code>, then a print dialog is displayed. This is the default behavior.</p> <p><i>font</i> is interpreted as described for <code>print-text</code>.</p>
See also	<p><code>print-editor-buffer</code></p> <p><code>print-text</code></p>

print-rich-text-pane*Function*

Summary	Prints the contents of a <code>rich-text-pane</code> , on Microsoft Windows.	
Package	<code>capi</code>	
Signature	<code>print-rich-text-pane</code> <i>pane</i> &key <i>jobname printer interactive selection</i> => <i>result</i>	
Arguments	<i>pane</i>	A <code>rich-text-pane</code> .
	<i>jobname</i>	A string, or <code>nil</code> .
	<i>printer</i>	A printer, or <code>nil</code> .
	<i>interactive</i>	A boolean.
	<i>selection</i>	A boolean.
Values	<i>result</i>	A boolean.
Description	<p>The function <code>print-rich-text-pane</code> prints the contents in <i>pane</i>.</p> <p><i>jobname</i> is the name of the print job. The default value is <code>nil</code>, meaning that the name "Document" is used.</p> <p><i>printer</i> is the printer to use. The default value is <code>nil</code>, meaning that the <code>current-printer</code> is used.</p> <p><i>interactive</i>, if true, specifies that a <code>print-dialog</code> is displayed before printing. The default value of <i>interactive</i> is <code>t</code>.</p> <p><i>selection</i> is a boolean specifying what to print. If true, only the current selection is printed. If <code>nil</code>, all the contents of <i>pane</i> are printed. The default value is <code>nil</code>.</p> <p>Note: <code>print-rich-text-pane</code> is supported only on Microsoft Windows.</p>	
See also	<code>rich-text-pane</code>	

print-text

Function

Summary	Prints plain text to a printer.
Package	<code>capi</code>
Signature	<code>print-text line-function &key printer tab-spacing interactive font</code>
Description	<p>The <code>print-text</code> function prints plain text to a printer specified by <i>printer</i>, and defaulting to the current printer.</p> <p>The <i>line-function</i> is called repeatedly with no arguments to enumerate the lines of text. It should return <code>nil</code> when the text is exhausted.</p> <p>The <i>tab-spacing</i> argument, which defaults to 8, specifies the number of spaces printed when a tab character is encountered.</p> <p>If <i>interactive</i> is <code>t</code>, then a print dialog is displayed. This is the default behavior.</p> <p><i>font</i> should be a <code>gp:font</code> object, or a Font Description object, or a symbol which is a font alias as defined by <code>define-font-alias</code>. The printed text is line wrapped on the assumption that the font is fixed width, so be sure to pass a suitable font. The default value of <i>font</i> is a Font Description for a fixed pitch font of size 10.</p>
See also	<code>print-editor-buffer</code> <code>print-file</code>

printer-configuration-dialog

Function

Summary	Displays a dialog allowing the user to configure printers.
Package	<code>capi</code>

Signature	<code>printer-configuration-dialog &key <i>screen owner</i></code>
Description	<p>The <code>printer-configuration-dialog</code> function displays the printer configuration dialog that allows users to add and configure PostScript printers.</p> <p>This applies only on Unix.</p> <p>The <code>screen</code> argument specifies a CAPI screen on which to display the dialog. The <code>owner</code> argument controls which interface owns the dialog. If it is specified it should be a currently displayed CAPI interface; it defaults to the current top level interface.</p> <p>The general options that are available are described under <code>install-postscript-printer</code>. In addition, printer-specific options (which are defined in the printer PPD file) are available.</p> <p>The printers that are visible in the dialog are defined by files in the directories in the list <code>*printer-search-path*</code>.</p>
See also	<code>install-postscript-printer</code> <code>*printer-search-path*</code>

printer-metrics

Structure Type

Summary	The type of objects containing printer metrics.
Package	<code>capi</code>
Description	<p>A <code>printer-metrics</code> object is returned by <code>get-printer-metrics</code>. The readers for the slots of a <code>printer-metrics</code> object are described below.</p> <p><code>printer-metrics-device-height</code> and <code>printer-metrics-device-width</code> respectively return the height and width of the printable page in the internal units used by the printer driver or printing subsystem of the</p>

printer. These functions should not be used to determine the aspect ratio of the printable page as some printers have size units that differ in the x and y directions.

`printer-metrics-dpi-x` and `printer-metrics-dpi-y` return the number of printer device units per inch in the x and y directions respectively. This typically corresponds to the printer resolution, although in some cases this may not be known. For example, a generic PostScript language compatible driver might always return 300dpi, even though it cannot know the resolution of the printer the PostScript file will actually be printed on.

`printer-metrics-height` and `printer-metrics-width` respectively return the height and width of the printable area in millimeters.

`printer-metrics-left-margin` and `printer-metrics-top-margin` respectively return the current left margin and current top margin of the printable area in millimeters.

`printer-metrics-max-height` and `printer-metrics-max-width` respectively return the greatest possible height and width of the printable area in millimeters.

`printer-metrics-min-left-margin` and `printer-metrics-min-top-margin` respectively return the smallest possible left margin and top margin of the printable area in millimeters.

`printer-metrics-paper-height` and `printer-metrics-paper-width` respectively return the height and width of the paper selected for this printer in millimeters.

See also

`get-printer-metrics`

ppd-directory*Variable*

Summary The directory in which LispWorks looks for PPD files.

Package `capi`

Initial value `nil`

Description The variable `*ppd-directory*` specifies where LispWorks looks for PostScript Printer Definition (PPD) files.

This applies only on Unix/Linux.

The directory which is the value of `*ppd-directory*` should contain PPD files (files with extension `ppd`) either directly, or under subdirectories. The PPD files under each subdirectory are grouped together, with the name of the directory as the group name. PPD files in `*ppd-directory*` itself are grouped under the "Other" group.

printer-port-handle*Function*

Summary Returns the underlying handle to a printer port.

Package `capi`

Signature `printer-port-handle &optional port => handle`

Arguments *port* A printer port.

Values *handle* Platform-dependent.

Description The function `printer-port-handle` returns a platform-dependent value which represents the underlying handle to the printer port.

On Microsoft Windows, *handle* is the HDC for the printer device.

If *port* is passed it should be the value bound to *var* in `with-print-job`. If *port* is not supplied it defaults to the current printer port (dynamically bound within `with-print-job`).

See also `with-print-job`

printer-port-supports-p

Function

Summary Detects if the printer port can support a certain feature.

Package `capi`

Signature `printer-port-supports-p feature &optional port
=> supportedp, validp`

Arguments *feature* A keyword.

port A printer port.

Values *supportedp* A boolean.

validp A boolean.

Description The function `printer-port-supports-p` detects if the printer port can support the feature named by *feature*.

If *port* is passed it should be the value bound to *var* in `with-print-job`. If *port* is not supplied it defaults to the current printer port (dynamically bound within `with-print-job`).

supportedp indicates if the feature is supported.

validp indicates if the feature was recognised.

Currently the only value of *feature* that is recognised is `:postscript` and the *supportedp* value is true if the printer supports PostScript.

See also `with-print-job`

printer-search-path*Variable*

Summary	Specifies where to look for printer definition files.
Package	<code>capi</code>
Initial value	<code>("~/lispworks-printers/" nil)</code>
Description	<p>The variable <code>*printer-search-path*</code> specifies where to look for printer definition files.</p> <p>This applies only on Unix/Linux.</p> <p>The value is a list containing directory pathname designators specifying where to look for printer definition files. The list can also include the value <code>nil</code>, which is interpreted as the <code>printers</code> directory in the LispWorks library.</p> <p>To find known printers the system loads all files in these directories. If there are duplicate printer definitions, the printer in the first directory takes precedence.</p> <p>The default path is useful when printing from the Common LispWorks IDE, but applications that want to allow users to use printers should set the list appropriately.</p> <p>The first path in the <code>*printer-search-path*</code> list is regarded as the "local" path. New printers are saved in this path. When the user edits a printer that was found in another directory on <code>*printer-search-path*</code> and then tries to save it, the system prompts for whether to overwrite the original or save it in the "local" directory.</p> <p>The printer files can be copied to other directories, on the same machine, and hence to install printers in different directories.</p> <p>A printer file can be copied to other machines, provided the printer is installed on the other machine and the PPD file is available in the same path.</p>

process-pending-messages

Function

Summary	Processes all the pending messages in the current process.
Package	<code>capi</code>
Signature	<code>process-pending-messages <i>ignored</i> => nil</code>
Arguments	The single argument is ignored.
Description	The function <code>process-pending-messages</code> processes all the pending messages in the current process, and then returns <code>nil</code> . It is useful when your code needs to continuously do something, but also needs to respond to user input or other messages.

progress-bar

Class

Summary	A pane that is used to show progress during a lengthy task.
Package	<code>capi</code>
Superclasses	<code>range-pane</code> <code>titled-object</code> <code>simple-pane</code>
Description	<p>This pane is used to display progress during a lengthy task. It has no interactive behavior.</p> <p>The <code>range-pane</code> accessors (<code>setf range-start</code>) and (<code>setf range-end</code>) are used to specify the range of values the progress bar can display.</p> <p>The accessor (<code>setf range-slug-start</code>) is used to set the progress indication.</p>
See also	<code>range-pane</code> <code>titled-object</code>

prompt-for-color*Function*

Summary	Presents a dialog box allowing the user to choose a color.	
Package	<code>capi</code>	
Signature	<code>prompt-for-color message &key color colors owner => result, successp</code>	
Arguments	<i>message</i>	A string.
	<i>color</i>	A color specification.
	<i>colors</i>	A list.
	<i>owner</i>	An owner window.
Values	<i>result</i>	A color specification, or <code>nil</code> .
	<i>successp</i>	A boolean.
Description	<p>The function <code>prompt-for-color</code> pops up a dialog box allowing the user to choose a color.</p> <p><i>message</i> supplies a title for the dialog on Motif. On Microsoft Windows <i>message</i> is ignored.</p> <p><i>color</i> provides the default color in the dialog.</p> <p><i>colors</i> is a list of custom color specifications that the user can choose from.</p> <p>For a description of color specifications, see the "The Color System" chapter in the <i>LispWorks CAPI User Guide</i>.</p> <p><i>owner</i> specifies an owner window for the dialog. See the "Prompting for Input" chapter in the <i>LispWorks CAPI User Guide</i> for details.</p>	

prompt-for-confirmation

Function

Summary	Displays a dialog box with a message and Yes and No buttons.	
Package	<code>capi</code>	
Signature	<code>prompt-for-confirmation message &key screen owner cancel-button default-button continuation => result, successp</code>	
Arguments	<i>message</i>	A string.
	<i>screen</i>	A screen.
	<i>owner</i>	An owner window.
	<i>cancel-button</i>	A boolean.
	<i>default-button</i>	A keyword, or <code>nil</code> .
	<i>continuation</i>	A function or <code>nil</code> .
Values	<i>result</i>	A boolean.
	<i>successp</i>	A boolean.
Description	<p>The function <code>prompt-for-confirmation</code> displays a dialog box containing <i>message</i>, with Yes and No buttons. When either Yes or No is pressed, it returns two values:</p> <ul style="list-style-type: none">• a boolean indicating whether Yes was pressed• <code>⊔</code> (for compatibility with other prompt functions) <p><i>cancel-button</i> specifies whether a Cancel button also appears on the dialog. When Cancel is pressed, <code>abort</code> is called and the dialog is dismissed. The default value of <i>cancel-button</i> is <code>nil</code>.</p> <p><i>default-button</i> specifies which button has the input focus when the dialog appears (and is thus selected when the user immediately presses <code>Return</code>). The value <code>:ok</code> means Yes, the value <code>:cancel</code> means Cancel, and any other value means No. The default value of <i>default-button</i> is <code>nil</code>.</p>	

owner specifies an owner window for the dialog. See the "Prompting for Input" chapter in the *LispWorks CAPI User Guide* for details.

If *continuation* is non-`nil`, then it must be a function with a lambda list that accepts two arguments. The *continuation* function is called with the values that would normally be returned by `prompt-for-continuation`. On Cocoa, passing *continuation* causes the dialog to be made as a window-modal sheet and `prompt-for-confirmation` returns immediately, leaving the dialog on the screen. The `with-dialog-results` macro provides a convenient way to create a *continuation* function.

Example

```
(capi:prompt-for-confirmation "Continue?")

(multiple-value-bind (res success)
  (capi:prompt-for-confirmation "Yes, No or Cancel"
                               :cancel-button t)
  (if success
      res
      (abort)))
```

See also `confirm-yes-or-no`

prompt-for-directory

Function

Summary Displays a dialog prompting the user for a directory.

Package `capi`

Signature `prompt-for-directory message &key if-does-not-exist pathname pane-args popup-args owner continuation => result, successp`

Arguments

<i>message</i>	A string.
<i>if-does-not-exist</i>	One of <code>:ok</code> , <code>:prompt</code> or <code>:error</code> .
<i>pathname</i>	A pathname, or <code>nil</code> .

	<i>pane-args</i>	Arguments to pass to the pane.
	<i>popup-args</i>	Arguments to pass to the confirmer.
	<i>owner</i>	An owner window.
	<i>continuation</i>	A function or <code>nil</code> .
Values	<i>result</i>	A directory pathname, or <code>nil</code> .
	<i>successp</i>	A boolean.
Description	<p>The function <code>prompt-for-directory</code> prompts the user for a directory pathname using a dialog box. Like all the prompters, <code>prompt-for-directory</code> returns two values: the directory pathname and a flag indicating success. The <i>successp</i> flag will be <code>nil</code> if the dialog was cancelled, and <code>t</code> otherwise.</p> <p>On Windows and Motif, if <i>if-does-not-exist</i> is <code>:ok</code>, a non-existent directory can be chosen. When set to <code>:prompt</code>, if a non-existent directory is chosen, the user is prompted for whether the directory should be created. When set to <code>:error</code>, the user cannot choose a non-existent directory. The default value of <i>if-does-not-exist</i> is <code>:prompt</code>.</p> <p>On Cocoa it is never possible to choose a non-existent directory, and the value of <i>if-does-not-exist</i> is ignored.</p> <p><i>pathname</i>, if non-<code>nil</code>, supplies an initial directory for the dialog. The default value for <i>pathname</i> is <code>nil</code>, and with this value the dialog initializes with the current working directory.</p> <p><i>owner</i> specifies an owner window for the dialog. See the "Prompting for Input" chapter in the <i>LispWorks CAPI User Guide</i> for details.</p> <p>If <i>continuation</i> is non-<code>nil</code>, then it must be a function with a lambda list that accepts two arguments. The <i>continuation</i> function is called with the values that would normally be returned by <code>prompt-for-directory</code>. On Cocoa, passing <i>continuation</i> causes the dialog to be made as a window-modal sheet and <code>prompt-for-directory</code> returns immediately, leav-</p>	

ing the dialog on the screen. The `with-dialog-results` macro provides a convenient way to create a *continuation* function.

The prompt itself is created by passing an appropriate pane to `popup-confirmer`. Arguments can be passed to the `make-instance` of the pane and the call to `popup-confirmer` using *pane-args* and *popup-args* respectively. Currently, the pane used to create the file prompter is internal to the CAPI.

See also `popup-confirmer`
`prompt-for-file`

prompt-for-file *Function*

Summary	Displays a dialog prompting the user for a filename.	
Package	<code>capi</code>	
Signature	<code>prompt-for-file</code> <i>message</i> &key <i>pathname</i> <i>ok-check</i> <i>filter</i> <i>filters</i> <i>if-exists</i> <i>if-does-not-exist</i> <i>operation</i> <i>owner</i> <i>pane-args</i> <i>popup-args</i> <i>continuation</i> => <i>filename</i> , <i>successsp</i> , <i>filter-name</i>	
Arguments	<i>message</i>	A string or <code>nil</code> .
	<i>pathname</i>	A pathname designator or <code>nil</code> .
	<i>ok-check</i>	A function or <code>nil</code> .
	<i>filter</i>	A string or <code>nil</code> .
	<i>filters</i>	A property list.
	<i>if-exists</i>	One of <code>:ok</code> or <code>:prompt</code> .
	<i>if-does-not-exist</i>	One of <code>:ok</code> , <code>:prompt</code> or <code>:error</code> .
	<i>operation</i>	One of <code>:open</code> or <code>:save</code> .
	<i>owner</i>	An owner window.

	<i>continuation</i>	A function or <code>nil</code> .
Values	<i>filename</i>	A pathname or <code>nil</code> .
	<i>successp</i>	A boolean.
	<i>filter-name</i>	A string.
Description		The function <code>prompt-for-file</code> prompts the user for a file using a dialog box.
		<p><i>pathname</i>, if non-<code>nil</code>, is a pathname designator providing a default filename for the dialog.</p> <p><i>ok-check</i>, if non-<code>nil</code>, should be a function which takes a pathname designator argument and returns a true value if the pathname is valid.</p> <p><i>filter</i> specifies the initial filter expression. The default value is <code>"*.*"</code>. An example filter expression with multiple filters is <code>"*.LISP;*.LSP"</code>.</p> <p><i>filter</i> is used on all platforms. However on Motif, if <i>filter</i> contains multiple file types, only the first of these is used.</p> <p>On Cocoa <code>prompt-for-file</code> supports the selection of application bundles as files if they match the filter. For example, they will match if the filter expression contains <code>*.app</code> or <code>*.*</code>.</p> <p><i>filters</i> is a property list of filter names and filter expressions, presenting filters which the user can select in the dialog. If the <i>filter</i> argument is not one of the expressions in <i>filters</i>, an extra filter called <code>"Files"</code> is added for this expression.</p> <p>On Microsoft Windows the default value of <i>filters</i> is:</p>

```

("Lisp Source Files" "*.LISP;*.LSP"
 "Lisp Fasl" "*.OFASL"
 "Text Documents" "*.DOC;*.TXT"
 "Image Files" "*.BMP;*.DIB;*.ICO;*.CUR"
 "All Files" "*.*")

```

The "Lisp Fasls" extension may vary depending on the implementation.

On Cocoa the default value of *filters* is:

```
("Lisp Source Files" "*.lisp;*.lsp"
 "Text Documents" "*.txt;*.text"
 "All Files" "**.*")
```

filters is ignored on Motif.

When *if-exists* is `:ok`, an existing file can be returned. Otherwise the user is prompted about whether the file can be overwritten. The default for *if-exists* is `:ok` when *operation* is `:open` and `:prompt` when *operation* is `:save`.

When *if-does-not-exist* is `:ok`, a non-existent file can be chosen. When it is `:prompt`, the user is prompted if a non-existent file is chosen. When it is `:error`, the user cannot choose a non-existent file. The default for *if-does-not-exist* is `:prompt` if *operation* is `:open` and `:ok` if *operation* is `:save`.

operation chooses the style of dialog used, in LispWorks for Windows only. The default value is `:open`.

owner, if non-`nil`, specifies an owner window for the dialog. See the "Prompting for Input" chapter in the *LispWorks CAPI User Guide* for details.

If *continuation* is non-`nil`, then it must be a function with a lambda list that accepts three arguments. The *continuation* function is called with the values that would normally be returned by `prompt-for-file`. On Cocoa, passing *continuation* causes the dialog to be made as a window-modal sheet and `prompt-for-file` returns immediately, leaving the dialog on the screen. The `with-dialog-results` macro provides a convenient way to create a *continuation* function.

On Motif, the prompt itself is created by passing an appropriate pane to `popup-confirmer`. Arguments can be passed to the `make-instance` of the pane and the call to `popup-confirmer` using *pane-args* and *popup-args* respectively. Currently,

the pane used to create the file prompter is internal to the CAPI. *pane-args* and *popup-args* are ignored on Microsoft Windows.

filename is the full pathname of the file selected, or `nil` if the dialog was cancelled.

successp is a flag which is `nil` if the dialog was cancelled, and `t` otherwise.

On Microsoft Windows `prompt-for-file` returns a third value: *filter-name* is the name of the filter that was selected in the dialog.

```
Example (capi:prompt-for-file "Enter a filename:")

(capi:prompt-for-file "Enter a filename:"
                    :pathname "/usr/bin/cal")

(capi:prompt-for-file "Enter a filename:"
                    :ok-check 'probe-file)
```

```
See also popup-confirmer
         prompt-for-string
         prompt-for-directory
```

prompt-for-files

Function

Summary	Displays a dialog which returns multiple filenames.	
Package	<code>capi</code>	
Signature	<code>prompt-for-files</code> <i>message</i> & <i>key</i> <i>pathname</i> <i>ok-check</i> <i>filter</i> <i>filters</i> <i>if-exists</i> <i>if-does-not-exist</i> <i>operation</i> <i>owner</i> <i>pane-args</i> <i>popup-args</i> <i>continuation</i> <i>=></i> <i>filenames</i> , <i>successp</i> , <i>filter-name</i>	
Values	<i>filenames</i>	A list.
	<i>successp</i>	A boolean.

filter-name A string.

Description

The function `prompt-for-files` presents the user with a dialog box similarly to `prompt-for-file`, but in which multiple filenames can be selected.

The arguments are as for `prompt-for-file`, except that *filters* defaults to:

```
( "MS Word files" "*.doc"
  "HTML files" "*.htm;*.html"
  "Plain Text files" "*.txt;*.text"
  "All files" "**.*")
```

filenames is a list of filenames, or `nil` if the user cancels the dialog.

successp is a flag which is `nil` if the dialog was cancelled, and `t` otherwise.

filter-name is the name of the filter that was selected in the dialog.

If *continuation* is non-`nil`, then it must be a function with a lambda list that accepts three arguments. The *continuation* function is called with the values that would normally be returned by `prompt-for-files`. On Cocoa, passing *continuation* causes the dialog to be made as a window-modal sheet and `prompt-for-files` returns immediately, leaving the dialog on the screen. The `with-dialog-results` macro provides a convenient way to create a *continuation* function.

Note: `prompt-for-files` is currently implemented only in LispWorks for Windows and Cocoa.

See also

`prompt-for-file`

prompt-for-font*Function*

Summary

Presents a dialog box allowing the user to choose a font.

Package	<code>capi</code>
Signature	<code>prompt-for-font message &key font owner => result, successp</code>
Arguments	<p><i>message</i> A string.</p> <p><i>font</i> A font, a font description, or <code>nil</code>.</p> <p><i>owner</i> An owner window, or <code>nil</code>.</p>
Values	<p><i>result</i> A font, or <code>nil</code>.</p> <p><i>successp</i> A boolean.</p>
Description	<p>The function <code>prompt-for-font</code> displays a dialog box allowing the user to choose a font.</p> <p><i>message</i> supplies a title for the dialog.</p> <p><i>font</i>, if non-<code>nil</code>, provides defaults for the dialog box. The default value is <code>nil</code>.</p> <p><i>owner</i> specifies an owner window for the dialog. See the "Prompting for Input" chapter in the <i>LispWorks CAPI User Guide</i> for details.</p> <p>For a description of Graphics Ports fonts and font descriptions, see the <i>LispWorks CAPI User Guide</i>.</p>
See also	<code>find-best-font</code>

prompt-for-form

Function

Summary	Displays a text input pane and prompts the user for a form.
Package	<code>capi</code>
Signature	<code>prompt-for-form message &key package initial-value evaluate quotify ok-check value-function pane-args popup-args continuation => result, okp</code>

Description	<p>The function <code>prompt-for-form</code> prompts the user for a form by providing a text input pane that the form can be typed into.</p> <p>The form is read in the <i>package</i> if specified or <i>*package*</i> if not. If <i>evaluate</i> is non-<code>nil</code> then the result is the evaluation of the form, otherwise it is just the form itself. The printed version of <i>initial-value</i> will be placed into the text input pane as a default, unless <i>quotify</i>, which defaults to <i>evaluate</i>, specifies otherwise. If <i>value-function</i> is provided it overrides the default value function which reads the form and evaluates it when required. If the <i>ok-check</i> is provided it will be passed the entered form and should return <code>t</code> if the form is a valid result.</p> <p>If <i>continuation</i> is non-<code>nil</code>, then it must be a function with a lambda list that accepts two arguments. The <i>continuation</i> function is called with the values that would normally be returned by <code>prompt-for-form</code>. On Cocoa, passing <i>continuation</i> causes the dialog to be made as a window-modal sheet and <code>prompt-for-form</code> returns immediately, leaving the dialog on the screen. The <code>with-dialog-results</code> macro provides a convenient way to create a <i>continuation</i> function.</p> <p>The prompter is created by calling <code>prompt-for-string</code>. Arguments can be passed to the <code>make-instance</code> of the pane and the call to <code>popup-confirmer</code> using <i>pane-args</i> and <i>popup-args</i> respectively, and an input history can be implemented by supplying a <i>history-function</i> or <i>history-symbol</i> in <i>popup-args</i>.</p>
Example	<p>Try the following examples, and each time enter (+ 1 2) into the input pane.</p> <pre>(capi:prompt-for-form "Enter a form:") (capi:prompt-for-form "Enter a form:" :evaluate nil)</pre>
See also	<pre>prompt-for-forms prompt-for-string popup-confirmer text-input-pane</pre>

prompt-for-forms

Function

Summary	Displays a text input pane prompting the user for a number of forms.
Package	<code>capi</code>
Signature	<code>prompt-for-forms message &key package initial-value value-function pane-args popup-args continuation => result, okp</code>
Description	<p>The function <code>prompt-for-forms</code> prompts the user for a number of forms by providing a text input pane that the forms can be typed into, and it returns the forms in a list. The forms are read in the specified <i>package</i> or <i>*package*</i> if not. If <i>evaluate</i> is non-<code>nil</code> then the result is the evaluation of the form, else it is just the form itself.</p> <p>The printed version of <i>initial-value</i> will be placed into the text input pane as a default.</p> <p>If <i>continuation</i> is non-<code>nil</code>, then it must be a function with a lambda list that accepts two arguments. The <i>continuation</i> function is called with the values that would normally be returned by <code>prompt-for-forms</code>. On Cocoa, passing <i>continuation</i> causes the dialog to be made as a window-modal sheet and <code>prompt-for-forms</code> returns immediately, leaving the dialog on the screen. The <code>with-dialog-results</code> macro provides a convenient way to create a <i>continuation</i> function.</p> <p>The prompter is created by passing an appropriate pane (in this case a text input pane) to <code>popup-confirmer</code>. Arguments can be passed to the <code>make-instance</code> of the pane and the call to <code>popup-confirmer</code> using <i>pane-args</i> and <i>popup-args</i> respectively.</p>
Example	<p>Try the following example, and enter 1 2 3 into the input pane.</p> <pre>(capi:prompt-for-forms "Enter some forms:")</pre>

See also `prompt-for-form`
`prompt-for-string`
`popup-confirmer`
`text-input-pane`

prompt-for-integer*Function*

Summary Prompts the user for an integer.

Package `capi`

Signature `prompt-for-integer message &key min max initial-value ok-check pane-args popup-args continuation => result, successp`

Arguments

<i>message</i>	A string.
<i>min</i>	An integer or <code>nil</code> .
<i>max</i>	An integer or <code>nil</code> .
<i>initial-value</i>	An integer or <code>nil</code> .
<i>ok-check</i>	A function or <code>nil</code> .
<i>pane-args</i>	Arguments to pass to the pane.
<i>popup-args</i>	Arguments to pass to the confirmer.
<i>continuation</i>	A function or <code>nil</code> .

Description The function `prompt-for-integer` pops up a `text-input-pane` and prompts the user for an integer, which is returned in *result*.

When *min* or *max* are specified the allowable result is constrained accordingly.

initial-value determines the initial value displayed in the dialog. *initial-value* defaults to the value of *min*, or if *min* is `nil` then no initial value is displayed.

Further restrictions can be applied by passing an *ok-check* function. *ok-check* should take one argument, the currently entered number, and should return `t` if it is valid. If *ok-check* is `nil` (the default) then there is no further restriction.

If *continuation* is non-`nil`, then it must be a function with a lambda list that accepts two arguments. The *continuation* function is called with the values that would normally be returned by `prompt-for-integer`. On Cocoa, passing *continuation* causes the dialog to be made as a window-modal sheet and `prompt-for-integer` returns immediately, leaving the dialog on the screen. The `with-dialog-results` macro provides a convenient way to create a *continuation* function.

The prompter is created by passing `text-input-pane` to `popup-confirmer`. Arguments can be passed to the `make-instance` of the pane and the call to `popup-confirmer` using *pane-args* and *popup-args* respectively.

Examples

```
(capi:prompt-for-integer "Enter an integer:")  
(capi:prompt-for-integer "Enter an integer:" :max 10)  
(capi:prompt-for-integer "Enter an integer:"  
                          :min 100 :max 200)  
(capi:prompt-for-integer "Enter an integer:"  
                          :ok-check 'evenp)
```

See also

```
prompt-for-string  
popup-confirmer  
text-input-pane
```

prompt-for-items-from-list

Function

Summary Prompts with a choice of items.

Package `capi`

Signature	<code>prompt-for-items-from-list</code> <i>items message &key pane-args popup-args interaction choice-class continuation => result, successp</i>	
Arguments	<i>items</i>	A sequence.
	<i>message</i>	A string.
	<i>pane-args</i>	Arguments to pass to the pane.
	<i>popup-args</i>	Arguments to pass to the confirmer.
	<i>interaction</i>	One of <code>:single-selection</code> , <code>:multiple-selection</code> , or <code>:extended-selection</code> .
	<i>choice-class</i>	A class name.
	<i>continuation</i>	A function or <code>nil</code> .
Description	The function <code>prompt-for-items-from-list</code> is similar to <code>prompt-with-list</code> . <i>interaction</i> defaults to <code>:extended-selection</code> .	
See also	<code>prompt-with-list</code>	

prompt-for-number*Function*

Summary	Prompts the user for a number.	
Package	<code>capi</code>	
Signature	<code>prompt-for-number</code> <i>message &key min max initial-value ok-check pane-args popup-args continuation => result, successp</i>	
Arguments	<i>message</i>	A string.
	<i>min</i>	A number or <code>nil</code> .
	<i>max</i>	A number or <code>nil</code> .
	<i>initial-value</i>	A number or <code>nil</code> .

ok-check A function or `nil`.
pane-args Arguments to pass to the pane.
popup-args Arguments to pass to the confirmer.
continuation A function or `nil`.

Description The function `prompt-for-number` pops up a `text-input-pane` and prompts the user for a number, which is returned in *result*.

The functionality corresponds exactly to that of `prompt-for-integer`, except that all types of numbers are allowed.

See also `prompt-for-integer`

prompt-for-string

Function

Summary Displays a text input pane and prompts the user for a string.

Package `capi`

Signature `prompt-for-string message &key pane-args popup-args ok-check value-function text initial-value print-function history-symbol history-function continuation => result, okp`

Description The function `prompt-for-string` prompts the user for a string and returns that string in *result* and a flag *okp* indicating that the dialog was not cancelled. The initial string can either be supplied directly as a string using the *text* argument, or by passing *initial-value* and a *print-function* for that value. *print-function* defaults to `princ-to-string`. The value returned can be converted into a different value by passing a *value-function*, which by default is the identity function. This *value-function* gets passed the text that was entered into the pane, and should return both the value to return and a flag

that should be `non-nil` if the value that was entered is not acceptable. If an *ok-check* is passed, then it should return `non-nil` if the value about to be returned is acceptable.

`prompt-for-string` creates an instance of `text-input-pane` or `text-input-choice` depending on the value of *history-function*. Arguments can be passed to the `make-instance` of this pane using *pane-args*. `prompt-for-string` then passes this pane to `popup-confirmer`. Arguments can be passed to the call to `popup-confirmer` using *popup-args*.

history-symbol, if `non-nil`, provides a symbol whose value is used to store an input history, when *history-function* is not supplied. The default value of *history-symbol* is `nil`.

history-function, if supplied, should be a function designator for a function with signature:

```
history-function &optional push-value
```

history-function is called with no argument to obtain the history which is used as the *items* of the `text-input-choice`, and with the latest input to update the history.

The default value of *history-function* is `nil`. In this case, if *history-symbol* is `non-nil` then a history function is constructed which stores its history in the value of that symbol.

If *continuation* is `non-nil`, then it must be a function with a lambda list that accepts two arguments. The *continuation* function is called with the values that would normally be returned by `prompt-for-string`. On Cocoa, passing *continuation* causes the dialog to be made as a window-modal sheet and `prompt-for-string` returns immediately, leaving the dialog on the screen. The `with-dialog-results` macro provides a convenient way to create a *continuation* function.

Examples

```
(capi:prompt-for-string "Enter a string:")
```

```
(capi:prompt-for-string
 "Enter an integer:"
 :initial-value 10
 :value-function #'(lambda (x)
                    (let ((integer
                          (ignore-errors
                           (read-from-string x))))
                      (values integer
                              (not (integerp integer))
                              )))))
```

See also `popup-confirmer`
`text-input-pane`

prompt-for-symbol

Function

Summary Prompts the user for a symbol.

Package `capi`

Signature `prompt-for-symbol message &key initial-value symbols package ok-check pane-args popup-args continuation => result, okp`

Description The function `prompt-for-symbol` prompts the user for a symbol which they should enter into the pane.

initial-value, if non-`nil`, should be a symbol which is initially displayed in the pane.

The symbols that are valid can be constrained in a number of ways.

symbols, if non-`nil`, should be a list of all valid symbols. The default is `nil`, meaning all symbols are valid.

package, if non-`nil`, is a package in which the symbol must be available. The value `nil` means that the value of `*package*` is used, and this is the default.

ok-check is a function which when called on a symbol will return non-`nil` if the symbol is valid.

The prompter is created by calling `prompt-for-string`. Arguments can be passed to the `make-instance` of the pane and the call to `popup-confirmer` using *pane-args* and *popup-args* respectively, and an input history can be implemented by supplying a *history-function* or *history-symbol* in *popup-args*.

If *continuation* is non-`nil`, then it must be a function with a lambda list that accepts two arguments. The *continuation* function is called with the values that would normally be returned by `prompt-for-symbol`. On Cocoa, passing *continuation* causes the dialog to be made as a window-modal sheet and `prompt-for-symbol` returns immediately, leaving the dialog on the screen. The `with-dialog-results` macro provides a convenient way to create a *continuation* function.

Examples

```
(capi:prompt-for-symbol "Enter a symbol:")

(capi:prompt-for-symbol "Enter a symbol:"
                        :package 'cl)

(capi:prompt-for-symbol "Enter a symbol:"
                        :symbols
                        '(foo bar baz))

(capi:prompt-for-symbol "Enter a symbol:"
                        :ok-check #'(lambda (symbol)
                                     (string< symbol "B")))
```

This last example shows how to implement a symbol prompter with an input history:

```
(defvar *my-history* (list "cdr" "car"))

(capi:prompt-for-symbol "Enter a symbol"
                        :popup-args
                        '(:history-symbol *my-history*))
```

See also

```
prompt-for-form
prompt-for-string
popup-confirmer
text-input-pane
```

prompt-for-value

Function

Summary	Prompts the user for a form to evaluate.
Package	<code>capi</code>
Signature	<code>prompt-for-value message &key package initial-value value-function pane-args popup-args continuation</code>
Description	<p>The function <code>prompt-for-value</code> prompts the user for a form and returns the result of evaluating that form.</p> <p>The form is read in the <i>package</i> if specified or <code>*package*</code> if not and the result is the evaluation of the form.</p> <p>If <i>initial-value</i> is supplied it provides a default form.</p> <p>If <i>value-function</i> is supplied it overrides the default value function which reads the form and evaluates it.</p> <p>If <i>continuation</i> is non-<code>nil</code>, then it must be a function with a lambda list that accepts two arguments. The <i>continuation</i> function is called with the values that would normally be returned by <code>prompt-for-value</code>. On Cocoa, passing <i>continuation</i> causes the dialog to be made as a window-modal sheet and <code>prompt-for-value</code> returns immediately, leaving the dialog on the screen. The <code>with-dialog-results</code> macro provides a convenient way to create a <i>continuation</i> function.</p> <p>The prompter is created by passing a <code>text-input-pane</code> to <code>popup-confirmer</code>. Arguments can be passed to the <code>make-instance</code> of the pane and the call to <code>popup-confirmer</code> using <i>pane-args</i> and <i>popup-args</i> respectively.</p>
Example	<pre>(capi:prompt-for-value "Square" :initial-value '(+ 1 2 3) :value-function #'(lambda (text) (let ((res (eval (read-from-string text)))) (* res res))))</pre>

See also `prompt-for-form`

prompt-with-list

Function

Summary	Prompts the user to select an item or items from a <code>choice</code> .	
Package	<code>capi</code>	
Signature	<code>prompt-with-list items message &key choice-class interaction value-function pane-args popup-args continuation => result, successp</code>	
Arguments	<i>items</i>	A sequence.
	<i>message</i>	A string.
	<i>choice-class</i>	A class name.
	<i>interaction</i>	One of <code>:single-selection</code> , <code>:multiple-selection</code> , or <code>:extended-selection</code> .
	<i>value-function</i>	A function, or <code>nil</code> .
	<i>pane-args</i>	Arguments to pass to the pane.
	<i>popup-args</i>	Arguments to pass to the confirmer.
	<i>continuation</i>	A function or <code>nil</code> .
Description	<p>The function <code>prompt-with-list</code> prompts the user with a <code>choice</code>. The user's selection is normally returned by the prompter.</p> <p><i>items</i> supplies the items of the <code>choice</code>.</p> <p><i>message</i> supplies a title for the <code>choice</code>.</p> <p><i>choice-class</i> determines the type of <code>choice</code> used in the dialog. <i>choice-class</i> defaults to <code>list-panel</code>, and must be a subclass of <code>choice</code>.</p>	

interaction determines the interaction style of the *choice* in the dialog. By default *interaction* is `:single-selection`. For single selection, the dialog has an **OK** and a **Cancel** button, while for other selection styles it has **Yes**, **No** and **Cancel** buttons where **Yes** means accept the selection, **No** means accept a null selection and **Cancel** behaves as normal.

The primary returned value is usually the selected items, but a *value-function* can be supplied that gets passed the result and can then return a new result. If *value-function* is `nil` (this is the default), then *result* is simply the selection.

If *continuation* is non-`nil`, then it must be a function with a lambda list that accepts two arguments. The *continuation* function is called with the values that would normally be returned by `prompt-with-list`. On Cocoa, passing *continuation* causes the dialog to be made as a window-modal sheet and `prompt-with-list` returns immediately, leaving the dialog on the screen. The `with-dialog-results` macro provides a convenient way to create a *continuation* function.

The prompter is created by passing an appropriate pane (in this case an instance of class *choice-class*) to `popup-confirmer`. Arguments can be passed to the `make-instance` of the pane and the call to `popup-confirmer` using *pane-args* and *popup-args* respectively. The initial selection can be specified using *choice* initargs `:selection`, `:selected-item` or `:selected-items` in *pane-args*.

Examples

```
(capi:prompt-with-list
  '(1 2 3 4 5) "Select an item:")

(capi:prompt-with-list
  '(1 2 3 4 5) "Select some items:"
  :interaction :multiple-selection
  :selection '(0 2 4))

(capi:prompt-with-list
  '(1 2 3 4 5) "Select an item:"
  :interaction :multiple-selection
  :choice-class 'capi:button-panel)
```

```
(capi:prompt-with-list
  '(1 2 3 4 5) "Select an item:"
  :interaction :multiple-selection
  :choice-class 'capi:button-panel
  :pane-args
  '(:layout-class capi:column-layout))
```

See also `popup-confirmer`
`list-panel`
`choice`

prompt-with-message

Function

Summary Prompts the user to select an item or items from a `choice`.

Package `capi`

Signature `prompt-with-message` *message* &key *owner* *continuation*

Arguments

<i>message</i>	A string.
<i>owner</i>	An owner window, or <code>nil</code> .
<i>continuation</i>	A function or <code>nil</code> .

Description The function `prompt-with-message` displays *message* in a dialog owned by *owner*.

`prompt-with-message` prompts the user with a `choice`. The user's selection is normally returned by the prompter.

If *continuation* is non-`nil`, then it must be a function with a lambda list that accepts two arguments. The *continuation* function is called with the values that would normally be returned by `prompt-with-message`. On Cocoa, passing *continuation* causes the dialog to be made as a window-modal sheet and `prompt-with-message` returns immediately, leaving the dialog on the screen. The `with-dialog-results` macro provides a convenient way to create a *continuation* function.

Examples `(capi:prompt-with-message
"No items were deleted.")`

See also `display-message-for-pane`
`display-message`

push-button

Class

Summary A `push-button` is a pane that displays either a piece of text or an image and when it is pressed it performs an action.

Package `capi`

Superclasses `button`
`titled-object`

Initargs `:alternate-callback`

A callback invoked on Microsoft Windows and Cocoa when pressing the mouse button over the `push-button` while a platform-specific modifier key is held down.

`:press-callback`

A callback invoked on Microsoft Windows and Motif when pressing the mouse button over the `push-button`.

Accessors `button-alternate-callback`
`button-press-callback`

Description The class `push-button` inherits most of its behavior from `button`. Note that it is normally best to use a `push-button-panel` rather than make the individual buttons yourself, as the button panel provides functionality for handling groups of buttons. However, push buttons can be used if you need to have more control over the button's behavior.

press-callback, if non-`nil`, should be a function which is called when the user presses the mouse left button over the push button. The arguments to *press-callback* are as specified by *callback-type*. This initarg is not supported on Cocoa.

alternate-callback, if non-`nil`, should be a function. On Microsoft Windows, it is called instead of *callback* when the button is clicked with the `Control` key held down. On Cocoa, it is called instead of *callback* when the button is clicked with the `Command` key held down. *alternate-callback* is not implemented for Motif or for other classes of `button`.

Example

```
(setq button (capi:contain
              (make-instance
               'capi:push-button
               :text "Press Me"
               :data '(:some :data)
               :callback #'(lambda (data interface)
                            (capi:display-message
                             "Pressed ~S"
                             data))))))

(capi:apply-in-pane-process
 button #'(setf capi:button-enabled) nil button)

(capi:apply-in-pane-process
 button #'(setf capi:button-enabled) t button)
```

See also

```
radio-button
check-button
button-panel
push-button-panel
```

push-button-panel

Class

Summary A `push-button-panel` is a pane containing a group of buttons.

Package `capi`

Superclasses `button-panel`

Description The class `push-button-panel` inherits all of its behavior from `button-panel`, which itself inherits most of its behavior from `choice`. Thus, the push button panel can accept items, callbacks, and so on.

Examples

```
(defun test-callback (data interface)
  (capi:display-message
   "Pressed ~S" data))

(capi:contain (make-instance 'capi:push-button-panel
  :title "Press a button:"
  :items
  ("Press Me" "No, Me")
  :selection-callback
  'test-callback))

(capi:contain (make-instance 'capi:push-button-panel
  :title "Press a button:"
  :items
  ("Press Me" "No, Me")
  :selection-callback
  'test-callback
  :layout-class
  'capi:column-layout))

(capi:contain (make-instance 'capi:push-button-panel
  :title "Press a button:"
  :items '(1 2 3 4 5 6 7 8 9)
  :selection-callback
  'test-callback
  :layout-class
  'capi:grid-layout
  :layout-args
  '(:columns 3)))
```

There is a further example in the file
`examples/capi/buttons/buttons.lisp`.

See also `push-button`
`radio-button-panel`
`check-button-panel`

quit-interface*Function*

Summary	Closes the top level interface containing a specified pane.	
Package	<code>capi</code>	
Signature	<code>quit-interface <i>pane</i> &key <i>force</i> => <i>result</i></code>	
Arguments	<i>pane</i>	A CAPI element.
	<i>force</i>	A boolean. The default value is <code>nil</code> .
Values	<i>result</i>	<code>t</code> if the interface was closed, <code>nil</code> otherwise.
Description	<p>The function <code>quit-interface</code> closes the top level interface containing <i>pane</i>, but first it verifies that it is okay to do this by calling the interface's <i>confirm-destroy-function</i>. If it is OK to close the interface, it then calls <code>destroy</code> to do so. If <i>force</i> is true, then neither the <i>confirm-destroy-function</i> or the <i>destroy-callback</i> are called, and the window is just closed immediately.</p> <p>Note: <code>quit-interface</code> must only be called in the process of the top level interface of <i>pane</i>. Menu callbacks on that interface will be called in that process, but otherwise you probably need to use <code>execute-with-interface</code> or <code>apply-in-pane-process</code>.</p>	
Example	<p>Here are two examples demonstrating the use of <code>quit-interface</code> with the <i>destroy-callback</i> and the <i>confirm-destroy-function</i>.</p>	

```
(setq interface (capi:display
  (make-instance
    'capi:interface
    :title "Test Interface"
    :destroy-callback
    #'(lambda (interface)
        (capi:display-message
         "Quitting ~S" interface))))))
```

```
(capi:apply-in-pane-process
 interface 'capi:quit-interface interface)
```

With this second example, the user is prompted as to whether or not to quit the interface.

```
(setq interface (capi:display
                 (make-instance
                  'capi:interface
                  :title "Test Interface"
                  :confirm-destroy-function
                  #'(lambda (interface)
                     (capi:confirm-yes-or-no
                      "Really quit ~S"
                      interface))))))

(capi:apply-in-pane-process
 interface 'capi:quit-interface interface)
```

See also `destroy`
`display`
`interface`

radio-button

Class

Summary A button that can be either selected or deselected, but when selecting it any other buttons in its group will be cleared.

Package `capi`

Superclasses `button`
`titled-object`

Description The class `radio-button` inherits most of its behavior from `button`. Note that it is normally best to use a `radio-button-panel` rather than make the individual buttons yourself, as the `button-panel` provides functionality for handling groups of buttons. However, radio buttons are provided in case you need to have more control over the button's behavior.

```

Example      (setq button (capi:contain
                (make-instance 'capi:radio-button
                               :text "Press Me")))

              (capi:apply-in-pane-process
                button #'(setf capi:button-selected) t button)

              (capi:apply-in-pane-process
                button #'(setf capi:button-selected) nil button)

              (capi:apply-in-pane-process
                button #'(setf capi:button-enabled) nil button)

              (capi:apply-in-pane-process
                button #'(setf capi:button-enabled) t button)

```

There is a further example in the file
[examples/capi/buttons/buttons.lisp](#).

```

See also    push-button
            check-button
            button-panel
            radio-button-panel

```

radio-button-panel

Class

Summary A pane containing a group of buttons of which only one can be selected at any time.

Package `capi`

Superclasses `button-panel`

Description The class `radio-button-panel` inherits all of its behavior from `button-panel`, which itself inherits most of its behavior from `choice`. Thus, the radio button panel can accept items, callbacks, and so forth.

```

Example  (capi:contain (make-instance
                    'capi:radio-button-panel
                    :title "Select a color:"
                    :items '(:red :green :blue)
                    :print-function 'string-capitalize))

          (setq buttons (capi:contain
                        (make-instance
                         'capi:radio-button-panel
                         :title "Select a color:"
                         :items '(:red :green :blue)
                         :print-function 'string-capitalize
                         :layout-class 'capi:column-layout)))

          (capi:choice-selected-item buttons)

```

There is a further example in the file
examples/capi/buttons/buttons.lisp.

See also `radio-button`
`push-button-panel`
`check-button-panel`

raise-interface

Function

Summary Raises the interface containing a specified pane to the front of the screen.

Package `capi`

Signature `raise-interface pane`

Description The function `raise-interface` raises the window containing *pane* to the front of the screen. To push it to the back use `lower-interface`, and to iconify it use `hide-interface`.

```

Example      (setq pane (capi:contain
                (make-instance
                 'capi:text-input-pane)))

              (capi:apply-in-pane-process
                pane 'capi:lower-interface pane)

              (capi:apply-in-pane-process
                pane 'capi:raise-interface pane)
    
```

```

See also     activate-pane
              hide-interface
              interface
              lower-interface
              quit-interface
    
```

range-pane

Class

Summary This class exists to support the `progress-bar` and `slider` classes. Consult the reference pages for these classes for further information.

Package `capi`

Superclasses `None`

Subclasses `progress-bar`
`scroll-bar`
`slider`

Initargs

<code>:start</code>	The lowest value of the range.
<code>:end</code>	The highest value of the range.
<code>:slug-start</code>	The start of the slug, corresponding to the current value of the range.
<code>:slug-end</code>	The end of the slug.
<code>:callback</code>	Called when the user changes the value.

`:orientation` One of `:horizontal` (the default) or `:vertical`.

Accessors `range-start`
`range-end`
`range-slug-start`
`range-slug-end`
`range-callback`
`range-orientation`

read-sound-file

Function

Summary Reads data from a sound file.

Package `capi`

Signature `read-sound-file source => array`

Arguments `source` A pathname designator.

Values `array` An array of element type (`unsigned-byte 8`).

Description The function `read-sound-file` reads data from `source` and returns an array of its contents.

Note: `read-sound-file` can be called during image building.

See also `load-sound`

rectangle

Class

Summary A `pinboard-object` that draws a rectangle.

Package `capi`

Superclasses `pinboard-object`

Subclasses	None.
Initargs	<code>:filled</code> A boolean, default value <code>nil</code> .
Accessors	<code>filled</code>
Description	The class <code>rectangle</code> provides a simple <code>pinboard-object</code> that draws a rectangle. <i>filled</i> determines whether the rectangle is filled.

redisplay-collection-item

Generic Function

Summary	Redisplays the area in a <code>collection</code> that belongs to an item.
Package	<code>capi</code>
Signature	<code>redisplay-collection-item</code> <i>collection</i> <i>item</i>
Description	The generic function <code>redisplay-collection-item</code> redisplay <i>item</i> in <i>collection</i> . There are methods supplied for <code>graph-pane</code> and <code>tree-view</code> .
See also	<code>collection</code>

redisplay-interface

Generic Function

Summary	Updates the state of an interface.
Package	<code>capi</code>
Signature	<code>redisplay-interface</code> <i>interface</i>

Description The generic function `redisplay-interface` updates the state of an interface, such as enabling and disabling menus, buttons, and so forth, that might have changed since the last call. When using this as a callback, you can use `:redisplay-interface` instead of the symbol, and then it will get passed the correct arguments regardless of the callback type.

Note: This method is called by `popup-confirmer` to update its button's enabled state, and so it should be called when state changes in a dialog.

See also `interface`
 `redisplay-menu-bar`
 `redraw-pinboard-layout`
 `display`

redisplay-menu-bar

Function

Summary Updates the menu bar of an interface.

Package `capi`

Signature `redisplay-menu-bar interface`

Description The function `redisplay-menu-bar` updates the interface's menu bar, such that menus become enabled and disabled as appropriate.

Compatibility Note This function has been superseded by `redisplay-interface`, which updates the menu bar, but also updates other state objects such as buttons, list panels and so on.

See also `interface`
 `redisplay-interface`

redraw-pinboard-layout*Function*

Summary	Redraws any pinboard objects within a specified rectangle.
Package	<code>capi</code>
Signature	<code>redraw-pinboard-layout</code> <i>pinboard</i> <i>x</i> <i>y</i> <i>width</i> <i>height</i> &optional <i>redisplay</i>
Description	The function <code>redraw-pinboard-layout</code> causes any pinboard objects within the given rectangle of the pinboard layout to get redrawn. If <i>redisplay</i> is <code>nil</code> , then the redisplay will be cached until a later update. The default for <i>redisplay</i> is <code>t</code> .
See also	<code>pinboard-object</code> <code>redraw-pinboard-object</code>

redraw-pinboard-object*Function*

Summary	Redraws a specified pinboard object.
Package	<code>capi</code>
Signature	<code>redraw-pinboard-object</code> <i>object</i> &optional <i>redisplay</i>
Description	The function <code>redraw-pinboard-object</code> causes the pinboard object <i>object</i> to be redrawn, unless <i>redisplay</i> is <code>nil</code> in which case the redisplay will be cached until a later update. The default for <i>redisplay</i> is <code>t</code> .
Example	There are examples in the directory <code>examples/capi/graphics/</code> .

See also `pinboard-object`
`pinboard-layout`
`redraw-pinboard-layout`

reinitialize-interface

Generic Function

Summary Reinitializes an existing `interface`.

Package `capi`

Signature `reinitialize-interface` *interface* &rest *initargs*

Description The generic function `reinitialize-interface` reinitializes an existing instance of a subclass of `interface`.

`reinitialize-interface` is called automatically by `find-interface` when this re-uses an interface.

 You can add methods to specialize on subclasses of `interface` which you define.

See also `find-interface`
 `interface-reuse-p`

remove-capi-object-property

Function

Summary Removes a property from the property list of an object.

Package `capi`

Signature `remove-capi-object-property` *object* *property*

Description The `remove-capi-object-property` function removes a property from the property list of an object.

All CAPI objects contain a property list, similar to the symbol `plist`. The functions `capi-object-property` and `(setf capi-object-property)` are the recommended ways of setting properties, and `remove-capi-object-property` is the way to remove a property.

```
Example (setq pane (make-instance 'capi:list-panel
                               :items '(1 2 3)))

(capi:capi-object-property pane 'test-property)
(setf (capi:capi-object-property pane 'test-property)
      "Test")
(capi:capi-object-property pane 'test-property)
(capi:remove-capi-object-property pane 'test-property)
(capi:capi-object-property pane 'test-property)
```

```
See also capi-object-property
         capi-object
```

remove-items

Generic Function

Summary	Removes some items from a collection.
Package	<code>capi</code>
Signature	<code>remove-items</code> <i>collection list-or-predicate</i>
Arguments	<i>collection</i> A collection. <i>list-or-predicate</i> A list, or a function of one argument returning a boolean value.
Description	The generic function <code>remove-items</code> removes from the <code>collection</code> <i>collection</i> those items determined by <i>list-or-predicate</i> .

If *list-or-predicate* is list, then the items removed are those matching some element of *list-or-predicate*, compared by the *test-function* of *collection*. Otherwise, the items removed are those for which the function *list-or-predicate* returns true.

This is logically equivalent to recalculating the collection items and then calling `(setf collection-items)`. However, `remove-items` is more efficient and causes less flickering on screen.

`remove-items` can only be used when the `collection` has the default *items-get-function* `svref`.

See also `append-items`
`collection`
`replace-items`

replace-dialog

Function

Summary	Replaces a replaceable dialog.
Package	<code>capi</code>
Signature	<code>replace-dialog</code> <i>interface</i> &rest <i>args</i> => nil
Arguments	<i>interface</i> An interface. <i>args</i> Other arguments as for <code>display-dialog</code> .
Description	The function <code>replace-dialog</code> displays a dialog in the same way the <code>display-dialog</code> does, except that it also destroys the existing dialog. <i>interface</i> is a CAPI interface to be displayed as a dialog. The arguments <i>args</i> are interpreted the same as the arguments to <code>display-dialog</code> , except that <i>modal</i> is ignored. <code>replace-dialog</code> displays the dialog like <code>display-dialog</code> .

See also `display-replacable-dialog`

replace-items

Generic Function

Summary Replaces some items in a collection.

Package `capi`

Signature `replace-items collection items &key start new-selection`

Arguments

<i>collection</i>	A collection.
<i>items</i>	A list.
<i>start</i>	A non-negative integer.
<i>new-selection</i>	A list specifying the selection.

Description The generic function `replace-items` replaces some items in the collection *collection* from *items*. `replace-items` can only be used when the `collection` has the default *items-get-function* `svref`.

start should be a non-negative integer and less than the number of items in *collection*.

Items in *collection* are replaced starting at index *start*, and proceeding until the end of the list *items*, or the end of the items in *collection*. If *items* is too long, the surplus is quietly ignored. `replace-items` never alters the number of items in the collection.

If supplied, *new-selection* should be a list of items specifying the new selection in collection. To specify no selection, pass `nil`.

If *new-selection* is not supplied, then `replace-items` attempts to preserve the selection. If some of the selected items are replaced, then the selection on these items is removed, but if a selected item simply moves, then the selection moves with it.

See also `append-items`
`collection`
`remove-items`

report-active-component-failure

Generic Function

Summary	Reports on failures to find or create a component.
Package	<code>capi</code>
Signature	<code>report-active-component-failure</code> <i>pane</i> <i>component-name</i> <i>error-string</i> <i>function-name</i> <i>hresult</i>
Arguments	<i>pane</i> An <code>ole-control-pane</code> . <i>component-name</i> A string or <code>nil</code> . <i>error-string</i> A string. <i>hresult</i> An integer or <code>nil</code> .
Description	The generic function <code>report-active-component-failure</code> is used to report on failures to find or create a component. <i>component-name</i> is the name of the component it tried to find. <i>error-string</i> is the error string. <i>function-name</i> is the name of the function that actually failed. <i>hresult</i> is the <code>hresult</code> that came back. It may be <code>nil</code> if the error is that the <code>guid</code> of the named component could not be found.

When the system fails to open the component, it calls `report-active-component-failure`, with the first argument the `ole-control-pane` *pane*. The default method for `ole-control-pane` tries to call `report-active-component-failure` again on its top level interface. The default method on `interface` calls `error`.

You can add your own methods, specializing on subclasses of `ole-control-pane` or subclasses of `interface`.

Note: this function is implemented only in LispWorks for Windows. Load the functionality by `(require "embed")`.

Example	See the example in <code>examples/com/ole/simple-container/doc-viewer-pair.lisp</code>
See also	<code>ole-control-pane</code>

reuse-interfaces-p

Function

Summary	Determines whether global interface re-use is enabled.	
Package	<code>capi</code>	
Signature	<code>reuse-interfaces-p => result</code>	
Signature	<code>(setf reuse-interfaces-p) value => value</code>	
Arguments	<i>value</i>	A boolean.
Values	<i>result</i>	A boolean.
Description	The function <code>reuse-interfaces-p</code> is the predicate for whether global interface re-use is enabled.	
	The function <code>(setf reuse-interfaces-p)</code> enables or disables global interface re-use.	

If global re-use is enabled, then `locate-interface` and `find-interface` may return existing interfaces. If global re-use is disabled, then `locate-interface` returns `nil` and `find-interface` returns a new interface.

See also `find-interface`
`locate-interface`

rich-text-pane

Class

Summary A text pane with extended formatting.

Package `capi`

Superclasses `simple-pane`

Initargs

- `:character-format`
A plist.
- `:paragraph-format`
A plist.
- `:change-callback`
A function called when a change is made.
- `:protected-callback`
A function determining whether the user may edit a protected part of the text.
- `:filename` A file to display.
- `:text` A string or `nil`.
- `:text-limit` An integer.

Accessors `rich-text-pane-change-callback`
`rich-text-pane-limit`
`rich-text-pane-text`

Description

The class `rich-text-pane` provides a text editor which supports character and paragraph formatting of its text.

Note: `rich-text-pane` is supported only on Microsoft Windows, and Cocoa in Mac OS X 10.3 and later. Some of its features are supported only on Microsoft Windows, as mentioned below.

character-format is the default character format. It is a plist which is interpreted in the same way as the *attributes-plist* argument of `set-rich-text-pane-character-format`. The default value of *character-format* is `nil`.

paragraph-format is the default paragraph format. It is a plist which is interpreted in the same way as the *attributes-plist* argument of `set-rich-text-pane-paragraph-format`. The default value of *paragraph-format* is `nil`.

change-callback, if non-`nil`, is a function of two arguments: the pane itself, and a keyword denoting the type of change. This second argument is either `:text` or `:selection`. The default value of *change-callback* is `nil`.

protected-callback is a function of four arguments: the pane itself, bounding indexes of the protected text, and a boolean which is true when the change would affect the selection. If the change would affect just a single character, this last argument is `nil`. If *protected-callback* returns `nil`, then the change is not performed. *protected-callback* is supported only on Microsoft Windows.

filename, if non-`nil`, should be a string or pathname naming a file to display in the pane. *filename* takes precedence over *text* if both are non-`nil`.

text, if non-`nil`, should be a string which is displayed in the pane if *filename* is `nil`.

text-limit, if non-`nil`, should be an integer which is an upper bound for the length of text displayed in the pane.

Note: *change-callback* and *protected-callback* are not yet implemented on Cocoa.

Note: The functions that are specific to `rich-text-pane` cannot be called before the pane is created. If you need to perform operations on the pane before it appears, and which cannot be performed using the `initargs`, the best approach is to define an `:after` method on `interface-display` on the class of the interface containing the `rich-text-pane`, and perform the operations inside this method.

See also

```
print-rich-text-pane
rich-text-pane-character-format
rich-text-pane-operation
set-rich-text-pane-character-format
rich-text-pane-paragraph-format
set-rich-text-pane-paragraph-format
```

rich-text-pane-character-format

Function

Summary	Returns the character format.
Package	<code>capi</code>
Signature	<code>rich-text-pane-character-format <i>pane</i> &key <i>selection</i></code> <code>=> <i>result</i></code>
Arguments	<i>pane</i> A <code>rich-text-pane</code> . <i>selection</i> A boolean.
Values	<i>result</i> A plist.
Description	The function <code>rich-text-pane-character-format</code> returns as a plist the current character attributes for <i>pane</i> .

selection determines the range for which the attributes are returned. If *selection* is `nil`, then the range is all the text in *pane*, otherwise the range is the current selection. The default value of *selection* is `t`.

An attribute appears in *result* only if its value is the same over all of the range. Therefore this form

```
(getf
 (capi:rich-text-pane-character-format pane) :bold
 :unknown)
```

will return:

- `t` if all the selection is bold
- `nil` if all the selection is not bold
- `:unknown` if the selection is only partially bold.

For the possible attributes, see `set-rich-text-pane-character-format`.

See also `rich-text-pane`
`set-rich-text-pane-character-format`

rich-text-pane-operation

Function

Summary	Gets and sets values and performs various operations on the pane.	
Package	<code>capi</code>	
Signature	<code>rich-text-pane-operation <i>pane operation</i> &rest <i>args</i> => <i>result, result?</i></code>	
Arguments	<code><i>pane</i></code>	A <code>rich-text-pane</code> .
	<code><i>operation</i></code>	A keyword specifying the operation to perform.

	<i>args</i>	The value or values to use, when the operation is setting something.
Values	<i>result</i>	Various, see below.
	<i>result2</i>	Returned only for <i>operation</i> <code>:get-selection</code> , see below.
Description	The valid values of <i>operation</i> on Microsoft Windows and Cocoa are:	
	<code>:paste</code> , <code>:cut</code> OR <code>:copy</code>	<i>result</i> is a boolean indicating whether it is currently possible to perform a <code>:paste</code> , <code>:cut</code> or <code>:copy</code> operation.
	<code>:paste</code> , <code>:cut</code> , OR <code>:copy</code>	Performs the indicated operation.
	<code>:select-all</code>	Selects all the text.
	<code>:set-selection</code>	<i>args</i> should be two integers <i>start</i> and <i>end</i> . Sets the selection to the region bounded by <i>start</i> (inclusive) and <i>end</i> (exclusive).
	<code>:get-selection</code>	Returns as multiple values the bounding indexes of the selection. <i>result</i> is the start (inclusive) and <i>result2</i> is the end (exclusive). If there is no selection, both values are the index of the insertion point.
	<code>:can-undo</code> OR <code>:can-redo</code>	<i>result</i> is a boolean indicating whether it is currently possible to perform an <code>:undo</code> or <code>:redo</code> operation.
	<code>:undo</code>	Undoes the last editing operation. Note that, after typing, it is the whole input, rather than a single character, that is undone. The

`:undo` operation may be repeated successively, to undo previous editing operations in turn.

Note: with RichEdit 1.0, `:undo` does not work repeatedly - it only undoes one previous editing operation. See `rich-text-version`

`:redo` Undoes the effect of the last `:undo` operation. The `:redo` operation may be repeated successively, to cancel the effect of previous `:undo` operations in turn.

Note: with RichEdit 1.0, `:redo` does not work. See `rich-text-version`.

`:get-modified` *result* is the value of a boolean modified flag. This flag can be set by the `:set-modified` operation. Also, editing the text sets it to true.

`:set-modified` Sets the modified flag. The argument is a boolean.

`:save-file` Saves the text to a file. Details below.

`:load-file` Loads the text from a file. Details below.

Additionally these values of *operation* are valid on Microsoft Windows, only:

`:get-word-wrap`

Returns a value indicating the word wrap, which can be the keyword `:none`. *result* can also be the keyword `:window` or a CAPI printer object, meaning that the text wraps according to the width of the window or the printer.

:set-word-wrap

Sets the word wrap. The argument can be as described for **:get-word-wrap**, and additionally it can be the keyword **:printer**, meaning the **current-printer**.

:hide-selection

Specifies whether the selection should be hidden (not highlighted) when *pane* does not have the focus. The argument is a boolean.

For operations **:save-file** and **:load-file**, *args* is a lambda list

filename* &*key selection format plain-text

filename is the file to save or load.

selection is a boolean, with default value **nil**.

format is **nil** or a keyword naming the file format. Values include **:rtf** and **:text** meaning Rich Text Format and text file respectively.

plain-text is a boolean, with default value **nil**.

With *operation* **:save-file**, if *selection* is true, only the current selection is saved. If *selection* is **nil**, all the text is saved. The default value of *format* is **:rtf** and there are two further allowed values, **:rtfnoobjs** and **:textized**. These are like **:rtf** and **:text** except in the way they deal with COM objects. See the documentation for **SF_RTFNOOBS** and **SF_TEXTIZED** in the **EM_STREAMOUT** entry in the MSDN for details. When saving with *format* **:rtf** or **:rtfnoobjs**, if *plain-text* is true, then keywords that are not common to all languages are ignored. With other values of *format*, *plain-text* has no effect.

With *operation* **:load-file**, if *selection* is true, the unselected text is preserved. If there is a selection, the new text replaces it. If there is no selection, the new text is inserted at the cur-

rent insertion point. If *selection* is `nil`, all the text is replaced. The default value of *format* is `nil`, meaning that the RTF signature is relied upon to indicate a Rich Text Format file. If *plain-text* is true, then keywords that are not common to all languages are ignored.

Examples

```
(setq rtp
      (capi:contain
       (make-instance
        'capi:rich-text-pane
        :text (format nil "First paragraph.~%Second
paragraph, a little longer.~%Another paragraph, which
should be long long enough that it spans more than one
line. ~%" ))))
```

Set the selection to characters 9 to 18:

```
(capi:rich-text-pane-operation rtp :set-selection 9 18)
```

Write all the text to a file in text format:

```
(capi:rich-text-pane-operation
 rtp :save-file "mydoc.txt" :format :text)
```

Paste:

```
(capi:rich-text-pane-operation rtp :paste)
```

See also

```
rich-text-pane
rich-text-version
```

rich-text-pane-paragraph-format*Function*

Summary

Returns the paragraph format.

Package

```
capi
```

Signature

```
rich-text-pane-paragraph-format pane => result
```

Arguments

```
pane                    A rich-text-pane.
```

Values	<i>result</i>	A plist.
Description	The function <code>rich-text-pane-paragraph-format</code> returns as a plist the paragraph attributes of the current paragraphs in <i>pane</i> . For the possible attributes, see <code>set-rich-text-pane-paragraph-format</code> .	
See also	<code>rich-text-pane</code>	

rich-text-version

Function

Summary	Identifies the version of RichEdit in use, on Microsoft Windows.	
Package	<code>capi</code>	
Signature	<code>rich-text-version => result</code>	
Values	<i>result</i>	A keyword indicating the version of the RichEdit control in use.
Description	<i>result</i> is <code>:rich-edit-2.0</code> if RichEdit 2.0 or newer is loaded. Otherwise <i>result</i> is <code>:rich-edit-1.0</code> . <code>rich-text-version</code> is supported only on Microsoft Windows.	
See also	<code>rich-text-pane</code>	

right-angle-line-pinboard-object

Class

Summary	A subclass of <code>pinboard-object</code> that displays a line drawn around two edges of the area enclosed by the pinboard object.	
---------	---	--

Package	<code>capi</code>
Superclasses	<code>line-pinboard-object</code>
Initargs	<code>:type</code> The type of line.
Description	<p>A subclass of <code>line-pinboard-object</code> which displays a line around the edge of the pinboard object rather than diagonally.</p> <p><i>type</i> can be one of two values.</p> <p><code>:vertical-first</code> Draw top-left to bottom-left to bottom-right.</p> <p><code>:horizontal-first</code> Draw top-left to top-right to bottom-right.</p> <p>The main use of this class is to produce graphs with right-angled edges rather than diagonal ones.</p>
Example	<pre>(capi:contain (make-instance 'capi:right-angle-line-pinboard-object :start-x 20 :start-y 20 :end-x 280 :end-y 100)) (capi:contain (make-instance 'capi:right-angle-line-pinboard-object :start-x 20 :start-y 120 :end-x 280 :end-y 200 :type :horizontal-first))</pre>
See also	<code>pinboard-layout</code>

row-layout

Class

Summary The `row-layout` class lays its children out in a row.

Package	<code>capi</code>	
Superclasses	<code>grid-layout</code>	
Initargs	<code>:ratios</code>	The size ratios between the layout's children.
	<code>:adjust</code>	The vertical adjustment for each child.
	<code>:gap</code>	The gap between each child.
	<code>:uniform-size-p</code>	If <code>t</code> , each child in the row has the same width.
Accessors	<code>layout-ratios</code>	
Description	<p>This lays its children out by inheriting the behavior from <code>grid-layout</code>. The <i>description</i> is a list of the layout's children, and the layout also translates the initargs <i>ratios</i>, <i>adjust</i>, <i>gap</i> and <i>uniform-size-p</i> into the grid layout's equivalent arguments <i>x-ratios</i>, <i>y-adjust</i>, <i>x-gap</i> and <i>x-uniform-size-p</i>.</p> <p><i>description</i> may also contain the keyword <code>:divider</code> which automatically creates a divider as a child of the <code>row-layout</code>. When specifying <code>:ratios</code> in a row with <code>:divider</code>, you should use <code>nil</code> to specify that the divider is given its minimum size.</p>	

Examples

```
(setq row (capi:contain
  (make-instance
    'capi:row-layout
    :description
    (list
      (make-instance 'capi:push-button
        :text "Press me")
      (make-instance 'capi:title-pane
        :text "Title")
      (make-instance 'capi:list-panel
        :items '(1 2 3)))
    :adjust :center)))

(capi:apply-in-pane-process
 row #'(setf capi:layout-y-adjust) :bottom row)

(capi:apply-in-pane-process
 row #'(setf capi:layout-y-adjust) :top row)
```

This last example shows a row with a stretchable dummy pane between two other elements which are fixed at their minimum size. Try resizing it:

```
(capi:contain
  (make-instance 'capi:row-layout
    :description
    (list (make-instance 'capi:push-button
      :text "foo")
      nil
      (make-instance 'capi:push-button
        :text "bar")))
  :ratios '(nil 1 nil)))
```

Compatibility Note **layout-divider-default-size** and *row-layout-divider* are not supported in LispWorks 4.4 and later.

See also *column-layout*

screen*Class*

Summary A **screen** is an object that represents the known monitor screens.

Package	<code>capi</code>	
Superclasses	<code>capi-object</code>	
Subclasses	<code>color-screen</code> <code>mono-screen</code>	
Initargs	<code>:width</code> <code>:height</code> <code>:number</code> <code>:depth</code> <code>:interfaces</code>	The width in pixels of the screen. The height in pixels of the screen. The screen number. The number of color planes in the screen. A list of all of the interfaces visible on the screen.
Readers	<code>screen-width</code> <code>screen-height</code> <code>screen-number</code> <code>screen-depth</code> <code>screen-interfaces</code> <code>screen-width-in-millimeters</code> <code>screen-height-in-millimeters</code>	
Description	<p>When the CAPI initializes itself it creates one or more screen objects and they are then used to specify where a window is to appear. A <code>screen</code> object can also be queried for information that the program may need to know about the screen that it is working on, such as its width, height and depth.</p> <p>On Microsoft Windows and Cocoa there is exactly one CAPI screen. When there are multiple monitors, there are several rectangles of pixels within the single CAPI screen.</p> <p>On Motif, there is one CAPI screen for each X11 screen.</p>	
Compatibility Note	<p>In LispWorks for Macintosh 4.3 there is one CAPI screen for each Cocoa screen. In LispWorks for Macintosh 4.4 and later, there is exactly one CAPI screen.</p>	

```

Example      (setq screen (capi:convert-to-screen))
              (capi:screen-width screen)
              (capi:screen-height screen)
              (capi:display (make-instance
                            'capi:interface :title "Test")
                            :screen screen)
              (capi:screen-interfaces screen)

```

See also `convert-to-screen`

screen-active-interface

Function

Summary Returns the active interface on a screen.

Package `capi`

Signature `screen-active-interface screen => interface`

Arguments *screen* A screen or document-container

Values *interface* An interface, or nil.

Description The function `screen-active-interface` returns the currently active interface on the screen *screen*, or nil if no CAPI interface is active or if this cannot be determined.

`screen-active-interface` also works with `document-container`, returning the active interface within the container.

See also `document-container`
`screen`

screen-active-p

Function

Summary	Determines whether a screen is active.	
Package	<code>capi</code>	
Signature	<code>screen-active-p screen => result</code>	
Arguments	<code>screen</code>	A screen.
Values	<code>result</code>	A boolean.
Description	The function <code>screen-active-p</code> is the predicate for whether a screen is active.	
See also	<code>screen</code>	

screen-logical-resolution

Function

Summary	Returns the logical resolution of <code>screen</code> .	
Package	<code>capi</code>	
Signature	<code>screen-logical-resolution screen => xlogres, ylogres</code>	
Arguments	<code>screen</code>	A screen.
Values	<code>xlogres, ylogres</code>	Integers representing the logical resolution of <code>screen</code> in DPI.
Description	The function <code>screen-logical-resolution</code> returns the logical resolution of <code>screen</code> , as dots per inch in the x and y directions.	
See also	<code>screen</code>	

screen-internal-geometry*Function*

Summary	Returns the geometry of the usable region of a screen or document container.	
Package	<code>capi</code>	
Signature	<code>screen-internal-geometry <i>screen</i> => <i>x</i>, <i>y</i>, <i>width</i>, <i>height</i></code>	
Arguments	<i>screen</i>	A screen.
Values	<i>x</i>	An integer.
	<i>y</i>	An integer.
	<i>width</i>	An integer.
	<i>height</i>	An integer.
Description	<p>The function <code>screen-internal-geometry</code> returns the geometry (as <i>x</i>, <i>y</i>, <i>width</i> and <i>height</i>) of the part of the screen that can be used to display windows. This region excludes any borders, the Mac OS X dock, and so on.</p> <p>On Microsoft Windows <code>screen-internal-geometry</code> works with <code>document-container</code>, returning the current size of the container (which may vary over time).</p>	
See also	<code>document-container</code> <code>screen</code>	

screens*Function*

Summary	Returns the active screens for a library.	
Package	<code>capi</code>	
Signature	<code>screens &optional <i>library</i> => <i>result</i></code>	

Arguments	<i>library</i>	A library name, a list, or <code>:any</code> .
Values	<i>result</i>	A list.
Description	<p>The function <code>screens</code> returns as a list all the active screens for <i>library</i>.</p> <p>A library name is a keyword naming a library, currently <code>:win32</code> on Microsoft Windows, <code>:motif</code> on Motif and <code>:cocoa</code> on Mac OS X.</p> <p><i>library</i> can be a library name, or a list of library names, or the keyword <code>:any</code>, meaning all the libraries. The default value of <i>library</i> is the result of <code>default-library</code>.</p>	
See also	<p><code>default-library</code> <code>screen</code></p>	

scroll

Generic Function

Summary	Moves the scrollbar and calls the <i>scroll-callback</i> .	
Package	<code>capi</code>	
Signature	<code>scroll self scroll-dimension scroll-operation scroll-value &rest options</code>	
Arguments	<i>self</i>	A pane that supports scrolling.
	<i>scroll-dimension</i>	<code>:vertical</code> , <code>:horizontal</code> OR <code>:pan</code> .
	<i>scroll-operation</i>	<code>:move</code> , <code>:step</code> OR <code>:page</code> .
	<i>scroll-value</i>	An integer, or a list of two integers, or a keyword, or a list of two keywords.
	<i>options</i>	A list.
Description	The generic function <code>scroll</code> works for panes that support scrolling - these are subclasses of <code>output-pane</code> and <code>layout</code> .	

`scroll` moves the scrollbar of a scrollable pane according to *scroll-dimension*, *scroll-operation* and *scroll-value*. It then calls the *scroll-callback* (see `output-pane`) with these arguments and *options*.

scroll-dimension determines whether the scrolling is vertical, horizontal or, if the value is `:pan`, in both dimensions.

scroll-operation determines the extent of the scroll. The value `:move` means that the pane scrolls to the position on the scroll range given by *scroll-value*, regardless of the current scroll position. The value `:step` means scroll from the current scroll position by *scroll-value* times the scroll step size. In the case of panes which do their own scrolling the scroll step size is determined by the operating system (OS). In the case of panes for which the CAPI computes the scroll, the scroll step size is as described in *with-geometry*. The value `:page` means scroll from the current scroll position by *scroll-value* times the scroll page size (which is also determined by the OS or the pane's geometry).

scroll-value should be an integer or keyword if *scroll-dimension* is `:horizontal` or `:vertical`. Allowed keyword values are `:start` and `:end`. *scroll-value* should be a list of two integers or keywords representing the horizontal and vertical scroll values if *scroll-dimension* is `:pan`.

options is a list containing arbitrary user data.

Compatibility
Note

`scroll` supersedes `set-scroll-position`, which is deprecated and no longer exported. The call

```
(capi:scroll pane :pan :move (list x y))
```

is equivalent to

```
(capi:set-scroll-position pane x y)
```

See also

`ensure-area-visible`
`get-scroll-position`
`output-pane`

`set-horizontal-scroll-parameters`
`set-vertical-scroll-parameters`
`with-geometry`

scroll-bar

Class

Summary	A pane which displays a scroll bar.	
Package	<code>capi</code>	
Superclasses	<code>range-pane</code> <code>simple-pane</code> <code>titled-object</code>	
Initargs	<code>:line-size</code>	The distance scrolled by the scroll-line gesture.
	<code>:page-size</code>	The distance scrolled by clicking inside the scroll bar.
	<code>:callback</code>	A function called after a scroll gesture, or <code>nil</code> .
Accessors	<code>scroll-bar-line-size</code> <code>scroll-bar-page-size</code>	
Description	<p>The class <code>scroll-bar</code> implements panes which display a scroll bar and call a callback when the user scrolls. It is not however the most usual way to add scroll bars - see the note below about <code>simple-pane</code>.</p> <p><i>line-size</i> is the logical size of a line, and is the distance moved when the user enters a scroll-line gesture, that is clicking on one of the arrow buttons at either end of the scroll bar or using a suitable arrow key. The default value of <i>line-size</i> is 1.</p> <p><i>page-size</i> is the logical size of a page, and is the distance moved when the user clicks inside the scroll bar. The default value of <i>page-size</i> is 10.</p>	

callback can be `nil`, meaning there is no callback. This is the default value. Otherwise, is a function of four arguments, the interface containing the scroll-bar, the scroll-bar itself, the mode of scrolling and the amount of scrolling. It has this signature:

`callback interface scroll-bar how where`

how can be one of `:line`, `:page`, `:move`, or `:drag`.

If *how* is `:line`, then *where* is an integer indicating how many lines were scrolled.

If *how* is `:page`, then *where* is an integer indicating how many pages were scrolled.

If *how* is `:move` or `:drag`, then *where* is an integer giving the new location of the *slug-start*, or `:start` or `:end`.

Note: the location of the slug can be found by the `range-pane` accessor `range-slug-start`.

Note: Rather than using `scroll-bar`, it is more usual to add scroll bars to a pane by the `simple-pane` initargs `:horizontal-scroll` and `:vertical-scroll`

Example

```
(defun sb-callback (interface sb how where)
  (declare (ignorable interface))
  (format t "Scrolled ~a where ~a : ~a~%"
    how where (range-slug-start sb)))

(contain
  (make-instance 'capi:scroll-bar
    :callback 'sb-callback
    :page-size 10
    :line-size 2
    :visible-min-width 200))
```

See also

`simple-pane`

search-for-item

Generic Function

Summary	The generic function <code>search-for-item</code> returns the index of an item in a collection.
Package	<code>capi</code>
Signature	<code>search-for-item</code> <i>collection</i> <i>item</i>
Description	Returns the index of <i>item</i> in the <i>collection</i> , using the <i>collection-test-function</i> to determine equality, and returns <code>nil</code> if no match is found. <code>search-for-item</code> is the counterpart function to <code>get-collection-item</code> which given an index, finds the appropriate item.
See also	<code>get-collection-item</code> <code>collection</code>

selection

Function

Summary	Returns the primary selection.
Package	<code>capi</code>
Signature	<code>selection</code> <i>self</i> &optional <i>format</i> => <i>result</i>
Arguments	<i>self</i> A displayed CAPI pane or interface. <i>format</i> A keyword.
Values	<i>result</i> A string, an <code>image</code> , a Lisp object, or <code>nil</code> .
Description	The function <code>selection</code> returns the contents of the primary selection as a string, or <code>nil</code> if there is no selection. <i>format</i> controls what kind of object is read. The following values of <i>format</i> are recognized:

<code>:string</code>	The object is a string. This the default value.
<code>:image</code>	The object is of type <code>image</code> , converted from whatever format the platform supports.
<code>:value</code>	The object is the Lisp value.

When *format* is `:image`, the image returned by `selection` is associated with *self*, so you can free it explicitly with `free-image` or it will be freed automatically when the pane is destroyed.

On Microsoft Windows there is no notion of selection, so this mechanism is internal to Lisp.

Note that X applications may or may not use the primary selection for their paste operations. For instance, Emacs is configurable by the variable `interprogram-paste-function`.

See also

- `clipboard`
- `free-image`
- `image`
- `selection-empty`
- `set-selection`

selection-empty

Function

Summary	Determines whether there is a primary selection of a particular kind.	
Package	<code>capi</code>	
Signature	<code>selection-empty self &optional format => result</code>	
Arguments	<i>self</i>	A displayed CAPI pane or interface.
	<i>format</i>	A keyword.
Values	<i>result</i>	<code>t</code> or <code>nil</code> .

Description The function `selection-empty` returns `nil` if there is a primary selection of the kind indicated by *format*, or `t` if there is no such selection.

format controls what kind of object is checked. The following values of *format* are recognized:

`:string` The object is a string. This the default value.

`:image` The object is of type `image`, converted from whatever format the platform supports.

`:value` The object is the Lisp value.

See also `image`
`selection`

set-application-interface

Function

Summary Specifies the main Cocoa application interface.

Package `capi`

Signature `set-application-interface` *interface*

Arguments *interface* An object of type `cocoa-default-application-interface`

Description The function `set-application-interface` sets *interface* as the main application interface. This interface is used to supply the application menu and receives various callbacks associated with the application.

`set-application-interface` must be called before any CAPI functions that make the `screen` object (such as `convert-to-screen` and `display`).

interface should not be displayed like a normal interface.

`set-application-interface` is only applicable when running under Cocoa.

See also `cocoa-default-application-interface`

set-button-panel-enabled-items

Generic Function

Summary Sets the enabled state of the items in a button panel.

Package `capi`

Signature `set-button-panel-enabled-items button-panel &key enable disable set test key`

Description The generic function `set-button-panel-enabled-items` sets the enabled state of the items in a button panel. If *set* is `t`, then *enable* is ignored and all items are enabled except those in the *disable* list. If *set* is `nil`, *disable* is ignored and all items are disabled except those in the *enable* list. If *set* is not given, the items in the *enable* list are enabled and the items in the *disable* list are disabled. If an item is in both lists, it is enabled. A button is in a list when the data of the button matches one of the items in the list. A match is defined as a non-`nil` return value from the test function. The default test function is `equal`.

See also `button-panel`
`redisplay-interface`

set-clipboard

Function

Summary Sets the contents of the system clipboard.

Package `capi`

Signature	<code>set-clipboard <i>self value</i> &optional <i>string plist</i> => <i>result</i></code>	
Arguments	<i>self</i>	A displayed CAPI pane or interface.
	<i>value</i>	A Lisp object (not necessarily a string) to make available within the local Lisp image.
	<i>string</i>	The string representation of <i>value</i> to export, or <code>nil</code> . If <code>nil</code> and <i>value</i> is a string, then that will be exported as the string.
	<i>plist</i>	A property list of additional format/value pairs to export. The currently supported formats are as described for <code>clipboard</code> . You can export more than one format simultaneously.
Values	<i>result</i>	A string, or <code>nil</code> .
Description	The function <code>set-clipboard</code> sets the contents of the system clipboard to be the text of <i>string</i> .	
	In Microsoft Windows applications (including LispWorks in Windows emulation mode), the contents of the system clipboard is usually accessed by the user with the <code>Ctrl+v</code> gesture.	
	The X clipboard can be accessed by the <code>Ctrl+v</code> gesture in KDE/Gnome emulation, or by running the program <code>xclipboard</code> or the Emacs function <code>x-get-clipboard</code> . The most likely explanation for apparent inconsistencies after <code>set-clipboard</code> is that the pasting application doesn't use the X clipboard.	
	In Cocoa applications (including LispWorks), the contents of the system clipboard is usually accessed by the user with the <code>Command+v</code> gesture.	
Example	To export an image: <code>(capi:set-clipboard <i>pane</i> nil nil (list :image <i>image</i>))</code>	

To export an image with a text description

```
(capi:set-clipboard pane nil nil
  (list :image image
        :string "my image"))
```

See also `clipboard`
`selection`
`text-input-pane-copy`

set-confirm-quit-flag

Function

Summary Controls the behavior of `confirm-quit`

Package `capi`

Signature `set-confirm-quit-flag flag`

Arguments *flag* One of `t`, `nil` or `:check-editor-files`

Description The function `set-confirm-quit-flag` sets a flag which controls the behavior of `confirm-quit`.
 See `confirm-quit` for the effect.

Note: on initialization, the LispWorks IDE sets the flag to the stored value of the option **Tools > Global Preferences... > Confirm Before Exiting**.

See also `confirm-quit`

set-default-editor-pane-blink-rate

Function

Summary Sets the default cursor blinking rate for editor panes.

Package `capi`

Signature	<code>set-default-editor-pane-blink-rate</code> <i>blink-rate</i>
Arguments	<i>blink-rate</i> A non-negative real number, or <code>nil</code> .
Description	<p>The function <code>set-default-editor-pane-blink-rate</code> sets the default to use for the editor pane cursor blinking rate. This default value is used when <code>editor-pane-blink-rate</code> returns <code>nil</code>.</p> <p>Initially the setting is if this call has been made:</p> <pre>(set-default-editor-pane-blink-rate nil)</pre> <p>This means that the native blink rate will be used.</p> <p>The argument <i>blink-rate</i> is interpreted as a blinking rate as described in <code>editor-pane-blink-rate</code>.</p>
See also	<code>editor-pane-blink-rate</code> <code>editor-pane-native-blink-rate</code>

set-default-interface-prefix-suffix

Function

Summary	Sets the default suffix and prefix that are added to each interface title.
Package	<code>capi</code>
Signature	<code>set-default-interface-prefix-suffix</code> &key <i>prefix suffix child-prefix child-suffix</i> => <i>prefix, suffix, child-prefix, child-suffix</i>
Arguments	<i>prefix</i> A string or <code>nil</code> . <i>suffix</i> A string or <code>nil</code> . <i>child-prefix</i> A string or <code>nil</code> . <i>child-suffix</i> A string or <code>nil</code> .
Values	<i>prefix</i> A string or <code>nil</code> .

suffix A string or `nil`.
child-prefix A string or `nil`.
child-suffix A string or `nil`.

Description

The function `set-default-interface-prefix-suffix` sets the global default suffix and prefix that are added to each `interface` title. The prefix and suffix are added by the default method of `interface-extend-title`.

If *prefix*, *suffix*, *child-prefix* or *child-suffix* are supplied, their value must be either a string or `nil`. If any of them is not passed, the corresponding previously set value is not changed.

prefix and *suffix* specify the prefix and suffix to use for interfaces that are children of a `screen` object. These values do not affect *child-prefix* and *child-suffix*.

child-prefix and *child-suffix* specify the prefix and suffix to use for interfaces that are not children of a `screen` object, such as an interface inside a Multiple Document Interface (MDI) window. These values do not affect *prefix* and *suffix*.

The return values are the settings of the prefix, suffix, child prefix and child suffix after the call.

To check the current settings, call `set-default-interface-prefix-suffix` with no arguments. This does not change the current settings.

Before setting the title on a window on the screen, the system calls `interface-extend-title` with the interface and the title of the interface, and uses the result for the actual title. The default method of `interface-extend-title` checks *prefix* and *suffix* (or *child-prefix* and *child-suffix* for MDI) as were set by `set-default-interface-prefix-suffix`, and if they are non-`nil` adds the value to the title.

`set-default-interface-prefix-suffix` can be called after some windows are displayed. It automatically updates all current interface windows as if by calling `update-all-interface-titles`.

Example If you work in an environment when it is not always obvious on which machine your image is running, you can add the name of the machine to all windows by:

```
(capi:set-default-interface-prefix-suffix
  :suffix (format nil "-- ~a" (machine-instance)))
```

See also `interface-extend-title`
`update-all-interface-titles`

set-drop-object-supported-formats

Function

Summary Sets the list of formats for a drop object

Package `capi`

Signature `set-drop-object-supported-formats` *drop-object* *formats*

Arguments *drop-object* A *drop-object*, as passed to the *drop-callback*
formats A list of format keywords

Description The function `set-drop-object-supported-formats` sets the list of formats that the drop object *drop-object* wants to receive.

The `:string` format can be used to receive a string from another application and the `:filenames-list` format can be used to receive a list of filenames from another application such as the Macintosh Finder or the Windows Explorer.

Any other keyword in *formats* is assumed to be a private format that can only be used to receive objects from with the same Lisp image.

Note: `set-drop-object-supported-formats` should only be called within a *drop-callback*. See `simple-pane` for information about drop callbacks.

Example See `examples/capi/output-panes/drag-and-drop.lisp`

See also `drop-object-provides-format`
`simple-pane`

set-geometric-hint

Function

Summary The `set-geometric-hint` function sets the hint associated with a key.

Package `capi`

Signature `set-geometric-hint` *element key value*
&optional *override*

Description Set the hint associated with *key* to *value*. If *override* is `nil`, the value is not changed when there is already a hint for this key. The default is `t`.

See also `set-hint-table`
`element`

set-hint-table

Function

Summary Modifies the hint table for an element.

Package `capi`

Signature `set-hint-table` *element plist*

Description The function `set-hint-table` modifies the hint table for the element *element* to include *plist*. All existing hints are retained for keys not in the *plist*.

This may or may not change the on-screen geometry. To change the geometry of an interface, use `set-top-level-interface-geometry`.

See also `element`
 `set-geometric-hint`
 `set-top-level-interface-geometry`

set-horizontal-scroll-parameters

Generic Function

Summary Allows programmatic control of the parameters of a horizontal scroll bar.

Package `capi`

Signature `set-horizontal-scroll-parameters` *self* &key *min-range*
 max-range *slug-position*
 slug-size *page-size* *step-size*

Description The function `set-horizontal-scroll-parameters` sets the specified parameters of the horizontal scroll bar of *self*, which should be a displayed instance of a subclass of `output-pane` (such as `editor-pane`) or `layout`.

The other arguments are:

min-range The minimum data coordinate.

max-range The maximum data coordinate.

slug-position The current scroll position.

slug-size The length of the scroll bar slug.

page-size The scroll page size.

step-size The scroll step size.

Example	See the following files: <code>examples/capi/output-panes/scroll-test.lisp</code> <code>examples/capi/output-panes/scrolling-without-bar.lisp</code>
Compatibility Note	The function <code>set-horizontal-scroll-parameters</code> supercedes the function <code>set-scroll-range</code> , which is deprecated and no longer exported. The call <pre>(set-horizontal-scroll-parameters pane :min-range 0 :max-range 42)</pre> is equivalent to <pre>(set-scroll-range pane 42 nil)</pre>
See also	<code>scroll</code> <code>get-horizontal-scroll-parameters</code> <code>simple-pane</code>

set-pane-focus

Function

Summary	Sets the input focus to a pane.
Package	<code>capi</code>
Signature	<code>set-pane-focus</code> <i>pane</i>
Arguments	<i>pane</i> An instance of a subclass of <code>simple-pane</code> or <code>choice</code> .
Description	The function <code>set-pane-focus</code> sets the input focus to <i>pane</i> or one of its children.
Example	
See also	

set-object-automatic-resize

Function

Summary	Controls automatic resizing of objects on a pinboard.	
Package	capi	
Signature	<code>set-object-automatic-resize <i>object</i> &key <i>x-align</i> <i>y-align</i> <i>x-offset</i> <i>y-offset</i> <i>x-ratio</i> <i>y-ratio</i> <i>width-ratio</i> <i>height-ratio</i> <i>aspect-ratio</i> <i>aspect-ratio-y-weight</i> <i>pinboard</i></code>	
Arguments	<i>object</i>	A <code>pinboard-object</code> or a <code>simple-pane</code> .
	<i>x-align</i>	<code>nil</code> , <code>:left</code> , <code>:center</code> or <code>:right</code> .
	<i>y-align</i>	<code>nil</code> , <code>:top</code> , <code>:center</code> or <code>:bottom</code> .
	<i>x-offset</i>	A real number, default value 0.
	<i>y-offset</i>	A real number, default value 0.
	<i>x-ratio</i>	A positive real number or <code>nil</code> .
	<i>y-ratio</i>	A positive real number or <code>nil</code> .
	<i>width-ratio</i>	A positive real number or <code>nil</code> .
	<i>height-ratio</i>	A positive real number or <code>nil</code> .
	<i>aspect-ratio</i>	A positive real number, <code>t</code> or <code>nil</code> .
	<i>aspect-ratio-y-weight</i>	A real number, default value 0.5.
	<i>pinboard</i>	A <code>pinboard-layout</code> , if supplied.
Description	<p>The function <code>set-object-automatic-resize</code> arranges for <i>object</i> to be resized and/or re-positioned automatically when <i>pinboard</i> is resized, or removes such a setting.</p> <p>The value of <i>aspect-ratio</i> can be <code>t</code>, which means use the current aspect ratio of <i>object</i> (that is, its height divided by its width).</p>	

object should be either a `pinboard-object` or a `simple-pane` which is (or will be) displayed in a `pinboard-layout`. This is, *object* will be added to the *description* of the pinboard layout by one of its `:description` initarg, (`setf capi:layout-description`) or `manipulate-pinboard`.

pinboard is the pinboard layout for *object*. If *pinboard* is already displayed with *object* in its *description*, the argument *pinboard* can be omitted.

When *pinboard* is resized, *object* is resized if either *height-ratio* or *width-ratio* are set.

The new width of *object* is calculated as follows:

- If *width-ratio*, *height-ratio* and *aspect-ratio* are all set, the new width is the width of *pinboard* multiplied by *width-ratio*, and then modified as described below.
- If *width-ratio* is set and either *height-ratio* or *aspect-ratio* is not set, the new width is the width of *pinboard* multiplied by *width-ratio*.
- If *width-ratio* is not set, and both *height-ratio* and *aspect-ratio* are set, the new width is the new height divided by *aspect-ratio*.
- Otherwise, the new width is the same as the old width.

The new height of *object* is calculated as follows:

- If *width-ratio* and *aspect-ratio* are set, the new height is the new width multiplied by the aspect ratio. Note that if *height-ratio* is set, the new width will depend on *height-ratio* too.
- If *height-ratio* is set and either *width-ratio* or *aspect-ratio* are not set, the new height is the height of *pinboard* multiplied by *height-ratio*.
- If *height-ratio* is not set, but both *width-ratio* and *aspect-ratio* are set, the new height is the new width multiplied by *aspect-ratio*.

- Otherwise, the new height is the same as the old height.

If all of *width-ratio*, *height-ratio* and *aspect-ratio* are set, the new width and height of object are calculated as follows:

1. Compute *calculated-width* as the width of *pinboard* multiplied by *width-ratio*, and *calculated-height* as the height of *pinboard* multiplied by *height-ratio*.

2. Compute *aspect-ratio-ratio* as

(/ (/ *calculated-height* *calculated-width*) *aspect-ratio*)

3. Compute *correction* as

(*expt* *aspect-ratio-ratio* *aspect-ratio-y-weight*)

4. Compute the new width as *calculated-width* multiplied by *correction*, and the new height as the new width multiplied by *aspect-ratio*.

The result is that if *aspect-ratio-y-weight* is 0, *correction* is 1 and *height-ratio* is effectively ignored, while if *aspect-ratio-y-weight* is 1, *correction* cancels the effect of *width-ratio*. With the default value of 0.5, the resulting position is in the (geometric) middle, and *object* takes a fixed fraction of the area of the pinboard.

After resizing (if needed), *object* is also positioned horizontally if *x-align* is non-*nil*, and vertically if *y-align* is non-*nil*.

The new x coordinate of *object* is calculated as follows:

- If *x-ratio* is set, the new x coordinate is the sum of *x-ratio* multiplied by the width of *pinboard* plus *x-offset*, otherwise it is simply *x-offset*.
- The actual value of the x coordinate for *object* is adjusted according to the value of *x-align* such that the left, center or right of *object* align with the new coordinate.

The new y coordinate of *object* is calculated similarly, using *y-ratio* and *y-offset*, with an adjustment such that the top, center or bottom of *object* aligns with the new coordinate according to *y-align*.

If all of *width-ratio*, *height-ratio*, *x-align* and *y-align* are `nil`, automatic resizing/re-positioning of *object* is removed.

`set-object-automatic-resize` can be called before *object* is actually displayed, and its effect persists over calls adding and removing *object* to/from `pinboard-layouts`. If *object* is to be used in another pinboard layout, `set-object-automatic-resize` must be called to remove the automatic resizing from the first pinboard layout.

Examples

Example

Put an object of fixed size at the top right corner:

```
(set-object-automatic-resize object
                          :x-ratio 1 :x-align :right)
```

Put an object in the bottom-right quadrant:

```
(set-object-automatic-resize
 object
 :x-ratio 0.5 :y-ratio 0.5
 :width-ratio 0.5 :height-ratio 0.5)
```

Put an object with a fixed aspect ratio and object width linear with the width of the pinboard in the center:

```
(set-object-automatic-resize
 object
 :x-align :center :y-align :center
 :x-ratio 0.5 :y-ratio 0.5
 :aspect-ratio 0.6 :width-ratio 0.1)
```

See also

```
manipulate-pinboard
pinboard-layout
pinboard-object
simple-pane
```

set-rich-text-pane-character-format

Function

Summary	Sets the character format.													
Package	capi													
Signature	<code>set-rich-text-pane-character-format <i>pane</i> &key <i>selection</i> <i>attributes-plist</i> <i>default</i> => <i>result</i></code>													
Arguments	<i>pane</i>	A rich-text-pane.												
	<i>selection</i>	A boolean.												
	<i>attributes-plist</i>	A plist.												
	<i>default</i>	A boolean.												
Values	<i>result</i>	A plist.												
Description	<p>The function <code>set-rich-text-pane-character-format</code> sets current character attributes for <i>pane</i>.</p> <p><i>selection</i> determines the text for which the attributes are set. If <i>selection</i> is <code>nil</code>, then the attributes are set on the next text entered in <i>pane</i>. If <i>selection</i> is <code>t</code>, then the attributes are set on the current selection. The default value of <i>selection</i> is <code>t</code>.</p> <p><i>attributes-plist</i> is a plist of keywords and values. These keywords are valid on Microsoft Windows and Cocoa:</p> <table><tr><td><code>:bold</code></td><td>A boolean.</td></tr><tr><td><code>:italic</code></td><td>A boolean.</td></tr><tr><td><code>:underline</code></td><td>A boolean.</td></tr><tr><td><code>:face</code></td><td>A string naming a font.</td></tr><tr><td><code>:color</code></td><td>A color spec or alias specifying the foreground color.</td></tr><tr><td><code>:size</code></td><td>The size of the font.</td></tr></table>		<code>:bold</code>	A boolean.	<code>:italic</code>	A boolean.	<code>:underline</code>	A boolean.	<code>:face</code>	A string naming a font.	<code>:color</code>	A color spec or alias specifying the foreground color.	<code>:size</code>	The size of the font.
<code>:bold</code>	A boolean.													
<code>:italic</code>	A boolean.													
<code>:underline</code>	A boolean.													
<code>:face</code>	A string naming a font.													
<code>:color</code>	A color spec or alias specifying the foreground color.													
<code>:size</code>	The size of the font.													

Additionally these *attributes-plist* keywords are valid on Microsoft Windows only:

- :strikeout** A boolean.
- :offset** An integer specifying the vertical offset of characters from the line (a positive value makes them superscript and a negative value makes them subscript).
- :protected** A boolean.
- :charset** A cons (*charset* . *pitch-and-family*) where *charset* has the value of a Microsoft Windows charset identifier, and *pitch-and-family* is the value of (`logior` *pitch* *family*) where *pitch* and *family* have the value of a Windows pitch and a Windows font family respectively.

Example

Note: This example uses some features which are supported only on Microsoft Windows:

```
(defun ok-to-edit-p (pane start end s)
  (declare (ignore pane))
  (capi:prompt-for-confirmation
   (format nil "Editing~:[ ~; selection ~]from ~a to ~a"
            s start end)))

(setq rtp
  (capi:contain
   (make-instance
    'capi:rich-text-pane
    :protected-callback 'ok-to-edit-p
    :character-format
    '(:size 14 :color :red)
    :visible-min-height 300
    :visible-min-width 400
    :paragraph-format
    '(:start-indent 20 :offset -15)
    :text-limit 160
    :text (format nil "First paragraph.~%Second
paragraph, a little longer.~%Another paragraph, which
should be long long enough that it spans more than one
line. ~%" ))))
```

Enter some characters in the rich text window.

Set it all to blue:

```
(capi:set-rich-text-pane-character-format
 rtp
 :attributes-plist '(:color :blue)
 :selection nil)
```

Make it all protected:

```
(capi:set-rich-text-pane-character-format
 rtp :attributes-plist '(:protected t) :selection nil)
```

Now try to delete a character, and also to delete the selection.
In both cases the `ok-to-edit-p` callback is called.

See also

```
rich-text-pane
rich-text-pane-character-format
```

set-rich-text-pane-paragraph-format*Function*

Summary	Sets the paragraph format.	
Package	<code>capi</code>	
Signature	<code>set-rich-text-pane-paragraph-format</code> <i>pane</i> <i>attributes-plist</i> => <i>result</i>	
Arguments	<i>pane</i>	A <code>rich-text-pane</code> .
	<i>attributes-plist</i>	A plist, or <code>:default</code> .
Values	<i>result</i>	A plist.
Description	<p>The function <code>set-rich-text-pane-paragraph-format</code> sets paragraph attributes for the current paragraphs in <i>pane</i>.</p> <p>The current paragraphs are those paragraphs which overlap the current selection, or the paragraph containing the insertion point if there is no selection.</p> <p>If <i>attributes-plist</i> is the symbol <code>:default</code> then the default paragraph format of the <i>pane</i> is used. Otherwise <i>attributes-plist</i> is a plist of keywords and values. These are the valid keywords on Microsoft Windows and Cocoa:</p> <ul style="list-style-type: none"> <code>:alignment</code> <code>:left</code>, <code>:right</code> or <code>:center</code>. <code>:start-indent</code> A number setting the indentation. <code>:offset-indent</code> A number modifying the indentation. <code>:offset</code> A number setting the relative indentation of subsequent lines in a paragraph. <code>:right-indent</code> A number setting the right margin. <code>:tab-stops</code> A list of numbers. <p>Additionally this <i>attributes-list</i> keyword is valid on Microsoft Windows, only:</p>	

```
:numbering      nil, t, :bullet, :arabic, :lowercase,  
                :uppercase, :lower-roman OR  
                :upper-roman.
```

numbering specifies the numbering style. Rich Edit 3.0 supports all the above values of *numbering*. Please note that the Arabic and Roman styles start numbering from zero, and that only `t` and `:bullet` work with versions of Rich Edit before 3.0 (other values of *numbering* are quietly ignored).

start-indent specifies the indentation of the first line of a paragraph. A negative value removes the indentation.

offset-indent takes effect only when *start-indent* is not passed. It specifies an increase in the current indentation. Therefore, a negative value of *offset-indent* decreases the indentation.

offset specifies the offset of the second and following lines relative to the first line of the paragraph. That is, when the indentation of the first line is *indent*, the indentation of the second and subsequent lines is *indent + offset*. When *offset* is negative, the second and subsequent lines are indented less than the first line. If *indent + offset* is negative, then these lines are not indented.

tab-stops should be a list of numbers specifying the locations of tabs. No more than 32 tabs are allowed.

Example

```
(setq rtp  
  (capi:contain  
    (make-instance  
      'capi:rich-text-pane  
      :visible-min-height 300  
      :visible-min-width 400  
      :paragraph-format  
      '(:start-indent 20 :offset -15)  
      :text (format nil "First paragraph.~%Second  
paragraph, a little longer.~%Another paragraph, which  
should be long long enough that it spans more than one  
line. ~%" )))  
  
(capi:set-rich-text-pane-paragraph-format  
  rtp '(:offset-indent 30 :numbering :lowercase))
```

See also `rich-text-pane`
`rich-text-pane-paragraph-format`

set-selection	<i>Function</i>	
Summary	Sets the primary selection.	
Package	<code>capi</code>	
Signature	<code>set-selection self value &optional string plist => result</code>	
Arguments	<i>self</i>	A displayed CAPI pane or interface.
	<i>value</i>	A Lisp object (not necessarily a string) to make available within the local Lisp image.
	<i>string</i>	The string representation of <i>value</i> to export, or <code>nil</code> . If <code>nil</code> and <i>value</i> is a string, then that will be exported as the string.
	<i>plist</i>	A property list of additional format/value pairs to export. The currently supported formats are <code>:string</code> , whose value should be a string, and <code>:image</code> whose value should be a <code>image</code> object. This allows you to export more than one format simultaneously.
Values	<i>result</i>	A string, or <code>nil</code> .
Description	<p>The function <code>set-selection</code> sets the primary selection to be the text of <i>string</i>.</p> <p>On Microsoft Windows there is no notion of selection, so this mechanism is internal to Lisp.</p> <p>Note that X applications may or may not use the primary selection for their paste operations. The most likely explanation for apparent inconsistencies after <code>set-selection</code></p>	

set-printer-options*Function*

Summary	Sets various options in the given printer.
Package	<code>capi</code>
Signature	<code>set-printer-options printer &key output-file first-page last-page orientation copies</code>
Description	<p>The function <code>set-printer-options</code> allows some printer options for the current job to be set programmatically. Note that the user can change the various printer options in the dialog displayed by <code>print-dialog</code>.</p> <p>The <i>printer</i> argument should be a printer object returned by <code>current-printer</code> or <code>print-dialog</code>. This <i>printer</i> should then be passed to <code>with-print-job</code> to print using the options specified.</p> <p>The keyword arguments control which options are set. If a keyword is not passed then the option remains unchanged.</p> <p>Values of <i>output-file</i> are:</p> <p><code>nil</code> Print directly to the device.</p> <p><code>t</code> Print to a file chosen by the user at printing time.</p> <p>A pathname Print to the file given by pathname.</p> <p>Values of <i>first-page</i> are:</p> <p><code>:all</code> Print all pages.</p> <p>A integer Print from this page to the page given by <i>last-page</i>.</p> <p>Values of <i>orientation</i> are:</p> <p><code>:landscape</code> Print in landscape mode.</p> <p><code>:portrait</code> Print in portrait mode.</p> <p>Values of <i>copies</i>:</p>

A integer The number of copies to print.

Example

```
;; Print two copies to the current printer.
(let ((printer (capi:current-printer)))
  (capi:set-printer-options printer :copies 2)
  (capi:with-print-job (port :printer printer)
    (print-my-document port)))
```

See also `print-dialog`
 `current-printer`
 `with-print-job`

set-text-input-pane-selection

Function

Summary Sets the selection in a `text-input-pane`.

Package `capi`

Signature `set-text-input-pane-selection pane start end`

Arguments *pane* A `text-input-pane`.
start, end Bounding indexes for a subsequence of the
 text of *pane*.

Description The function `set-text-input-pane-selection` sets the
 selection in *pane* to be the text bounded by the indexes *start*
 (inclusive) and *end* (exclusive).

See also `text-input-pane-selection`
 `text-input-pane`

set-top-level-interface-geometry

Function

Summary Sets the geometry of a top level interface.

Package	<code>capi</code>
Signature	<code>set-top-level-interface-geometry</code> <i>interface</i> &key <i>x y</i> <i>width height</i>
Arguments	<i>interface</i> A CAPI interface. <i>x, y, width, height</i> Integers specifying the new geometry.
Description	The coordinates of <i>interface</i> are modified according to the keyword arguments passed. The value of <i>interface</i> should be a top level interface. If a keyword is omitted then that part of the coordinates is not changed.
Example	<pre>(setf ii (capi:element-interface (capi:contain (make-instance 'capi:text-input-pane)))) (multiple-value-bind (x y width height) (capi:top-level-interface-geometry ii) (capi:execute-with-interface ii 'capi:set-top-level-interface-geometry ii :x (round (+ x (/ width 4))) :y y :width (round (* 0.75 width)) :height height))</pre>
See also	<code>top-level-interface-p</code> <code>top-level-interface-geometry</code> <code>top-level-interface-display-state</code> <code>interface</code>

set-vertical-scroll-parameters*Generic Function*

Summary Allows programmatic control of the parameters of a vertical scroll bar.

Package	<code>capi</code>
Signature	<code>set-vertical-scroll-parameters</code> <i>self</i> &key <i>min-range</i> <i>max-range</i> <i>slug-position</i> <i>slug-size</i> <i>page-size</i> <i>step-size</i>
Description	<p>The function <code>set-vertical-scroll-parameters</code> sets the specified parameters of the vertical scroll bar of <i>self</i>, which should be a displayed instance of a subclass of <code>output-pane</code> (such as <code>editor-pane</code>) or <code>layout</code>.</p> <p>The other arguments are:</p> <p><i>min-range</i> The minimum data coordinate.</p> <p><i>max-range</i> The maximum data coordinate.</p> <p><i>slug-position</i> The current scroll position.</p> <p><i>slug-size</i> The length of the scroll bar slug.</p> <p><i>page-size</i> The scroll page size.</p> <p><i>step-size</i> The scroll step size.</p>
Example	<p>See the following CAPI example files:</p> <pre>examples/capi/output-panes/scroll-test.lisp examples/capi/output-panes/scrolling-without-bar.lisp</pre>
Compatibility Note	<p>The function <code>set-vertical-scroll-parameters</code> supersedes the function <code>set-scroll-range</code>, which is deprecated and no longer exported.</p> <p>The call</p> <pre>(set-vertical-scroll-parameters pane :min-range 0 :max-range 42)</pre> <p>is equivalent to</p> <pre>(set-scroll-range pane nil 42)</pre>

See also `scroll`
`set-horizontal-scroll-parameters`
`simple-pane`

shell-pane

Class

Summary A pane allowing the user to interact with a subprocess.

Package `capi`

Superclasses `interactive-pane`

Initargs `:command` The command which is run as a subprocess.

Accessors `shell-pane-command`

Description The class `shell-pane` creates an editor in which a subprocess runs.

User input is interpreted as input to the subprocess. In particular, when the user enters `return` in the last line, the line is sent to the subprocess. The output of the subprocess is displayed in the pane.

The default value of `command` is `nil`, which means that the actual command is determined as follows:

On Unix/Linux and Mac OS X, the value of the environment variable `ESHELL` is used if set, and otherwise the environment variable `SHELL` is consulted. If that is not set, then `/bin/csh` (`/bin/sh` on SVR4 platforms) is run.

On Microsoft Windows 95/98/ME, `command` is run.

On Windows NT/2000/XP, `cmd` is run.

Examples This function emulates user input on *pane*:

```
(defun send-keys-to-pane-aux (pane string newline-p)
  (loop for char across string
        do (capi:call-editor pane char))
  (if newline-p
    (capi:call-editor pane #\Return)))
```

This function trampolines to `send-keys-to-pane-aux` on the right process:

```
(defun send-keys-to-pane (pane string newline-p)
  (capi:apply-in-pane-process pane
    'send-keys-to-pane-aux
    pane string newline-p))

(setq sp (capi:contain
  (make-instance 'capi:shell-pane
    :visible-min-width
    '(character 60)
    :visible-min-height
    '(character 30))))
```

This call emulates the user typing `dir` followed by `Return`:

```
(send-keys-to-pane sp "dir" t)
```

show-interface

Function

Summary	The <code>show-interface</code> function brings the interface containing a specified pane back onto the screen.
Package	<code>capi</code>
Signature	<code>show-interface</code> <i>pane</i>
Description	This brings the interface containing <i>pane</i> back onto the screen. To hide it again, use <code>hide-interface</code> .
See also	<code>hide-interface</code> <code>activate-pane</code> <code>interface</code>

show-pane*Function*

Summary	Restores the specified pane to the screen.
Package	<code>capi</code>
Signature	<code>show-pane pane => pane</code>
Arguments	<code>pane</code> An instance of <code>simple-pane</code> or a subclass.
Description	The function <code>show-pane</code> restores the pane <code>pane</code> to the screen if it is hidden (for instance by <code>hide-pane</code>) or iconified.
See also	<code>hide-pane</code> <code>show-interface</code>

simple-layout*Class*

Summary	A <code>simple-layout</code> is a layout with a single child, and the child is resized to fill the space (where possible).
Package	<code>capi</code>
Superclasses	<code>x-y-adjustable-layout</code>
Description	A simple layout's description can be either a single child, or a list containing just one child. The simple layout then adopts the size constraints of its child, and lays the child out inside itself.
Example	<pre>(capi:contain (make-instance 'capi:simple-layout :description (list (make-instance 'capi:text-input-pane))))</pre>

See also `layout`
`row-layout`
`column-layout`

simple-network-pane

Class

Summary A graph pane which arranges its nodes in a grid.

Package `capi`

Superclasses `graph-pane`

Initargs :`x-gap` The horizontal node spacing.
 :`y-gap` The vertical node spacing.

Description The class `simple-network-pane` provides a graph which lays out its nodes in a rectangular grid by a simple algorithm.

 The default values of *x-gap* and *y-gap* are 200 and 100 respectively.

`simple-network-pane` is a subclass of `choice`, so for details of its selection handling, see `choice`.

Examples See the file `examples/capi/graphics/network.lisp`.

simple-pane

Class

Summary The class `simple-pane` is the superclass for any elements that actually appear as a native window, and is itself an empty window.

Package `capi`

Superclasses `element`

Subclasses

- `display-pane`
- `interface`
- `title-pane`
- `button-panel`
- `list-panel`
- `option-pane`
- `output-pane`
- `progress-bar`
- `slider`
- `text-input-pane`
- `tree-view`
- `toolbar`
- `layout`
- `button`

Initargs

- `:enabled` A boolean controlling whether the pane is enabled.
- `:background` The background color of the pane.
- `:foreground` The foreground color of the pane.
- `:font` The default font for the pane.
- `:horizontal-scroll`
 - `t`, `:without-bar`, or `nil`. If true the pane can scroll horizontally.
- `:vertical-scroll`
 - `t`, `:without-bar`, or `nil`. If true the pane can scroll vertically.
- `:visible-border`
 - A boolean or a keyword controlling whether the pane has a border, for some pane classes.
- `:internal-border`
 - A non-negative integer, or `nil`. Controls the width of the internal border.
- `:cursor` A keyword naming a built-in cursor, or a cursor object, or `nil`.

`:pane-menu` Specifies a menu to be raised by the
`:post-menu` gesture.

`:drop-callback`
 Specifies a drop callback for `output-pane` or
`interface`.

Accessors

`simple-pane-enabled`
`simple-pane-background`
`simple-pane-foreground`
`simple-pane-font`
`simple-pane-cursor`
`simple-pane-scroll-callback`
`simple-pane-drop-callback`

Readers

`simple-pane-horizontal-scroll`
`simple-pane-vertical-scroll`
`simple-pane-visible-border`

Description

enabled determines whether the pane is enabled. The default value is `t`. Note that changing the enabled state of a visible pane changes its appearance.

background and *foreground* are colors specified using the Graphics Ports color system. Additionally on Cocoa, the special value `:transparent` is supported, which makes the pane's background match that of its parent.

font should be `font`, a font description, or `nil`.

The value for *visible-border* can be any of the following, with the stated meanings where applicable:

`nil` Has no border.

`t` Has a border.

`:default` Use the default for the window type.

`:outline` Add an outline border.

There are various platform/pane class combinations which do not respond to all values of *visible-border*. For instance, on Windows XP with the default theme, `text-input-choice`

and `option-pane` always have a visible border regardless of the value of `visible-border`, while other classes including `display-pane`, `text-input-pane`, `list-panel`, `editor-pane` and `graph-pane` have three distinct border styles, with `visible-border :default` meaning the same as `visible-border t`.

If `internal-border` is non-`nil`, it should be a non-negative integer specifying the width of an empty region around the edge of the pane.

Any simple pane can be made scrollable by specifying `t` to `:horizontal-scroll` or `:vertical-scroll`. By default these values are `nil`, but some subclasses of `simple-pane` default them to `t` where appropriate (for instance `editor-panes` always default to having a vertical scroll-bar).

For a pane which is scrollable but does not display a scroll bar, pass the value `:without-bar` for `:horizontal-scroll` or `:vertical-scroll`. See the example in `output-panes/scrolling-without-bar.lisp`.

The height and width of a scrollable simple pane can be specified by the initargs `:scroll-height` and `:scroll-width`, which have the same meaning as `:internal-min-height` and `:internal-min-width`. See the *LispWorks CAPI User Guide* for more information about height and width initargs.

`cursor` specifies a cursor for the pane. `nil` means use the default cursor, and this is the default value. `cursor` can also be a cursor object as returned by `load-cursor`. The other allowed values are keywords naming built-in cursors which are supported on each platform as shown in the table below.

<i>cursor</i>	Cocoa	Windows	Motif
<code>:busy</code>	No	Yes	Yes
<code>:i-beam</code>	Yes	Yes	Yes

Table 1.2

<i>cursor</i>	Cocoa	Windows	Motif
<code>:top-left-arrow</code>	Yes	Yes	Yes
<code>:h-double-arrow</code>	Yes	Yes	Yes
<code>:v-double-arrow</code>	Yes	Yes	Yes
<code>:left-side</code>	Yes	Yes	Yes
<code>:right-side</code>	Yes	Yes	Yes
<code>:top-side</code>	Yes	Yes	Yes
<code>:bottom-side</code>	Yes	Yes	Yes
<code>:wait</code>	No	Yes	Yes
<code>:crosshair</code>	Yes	Yes	Yes
<code>:gc-notification</code>	No	Yes	Yes
<code>:top-left-corner</code>	No	Yes	Yes
<code>:top-right-corner</code>	No	Yes	Yes
<code>:bottom-left-corner</code>	No	Yes	Yes
<code>:bottom-right-corner</code>	No	Yes	Yes
<code>:hand</code>	Yes	Yes	Yes
<code>:fleur</code>	Yes	Yes	Yes
<code>:move</code>	Yes	Yes	Yes
<code>:closed-hand</code>	Yes	No	No
<code>:open-hand</code>	Yes	No	No
<code>:disappearing-item</code>	Yes	No	No

Table 1.2

Note: On Cocoa in Mac OS X 10.2, only `:i-beam` is supported.

`pane-menu` can be used to specify or create a menu to be displayed when the `:post-menu` gesture is received by the pane. It has the default value `:default` which means that `make-pane-popup-menu` is called to create the menu. For a full description of `pane-menu`, see the section "Popup menus for panes" in the *LispWorks CAPI User Guide*.

drop-callback can be specified for a pane that is an instance of `output-pane`, `interface` or a subclass of one of these. When the user drags an object over a window, the CAPI first tries to call the *drop-callback* of any `output-pane` under the mouse and otherwise calls the *drop-callback* of the top-level interface. The default value of *drop-callback* is `nil`, which means that there is no support for dropping into the pane.

For `editor-pane`, *drop-callback* can be `:default`, which provides support for dropping a string into the pane and inserting the string into the pane's editor buffer.

If *drop-callback* is any other non-`nil` value, it should be a function designator with this signature:

```
drop-callback pane drop-object stage
```

The function *drop-callback* is called by the CAPI at various times such as when the pane is displayed and when the user attempts to drop data into the pane. *pane* is the pane itself, *drop-object* is an object used to communicate information about the current dropping operation (see below) and *stage* is a keyword. *drop-callback* should handle these values of *stage*:

- :formats** This might occur when the pane is being displayed or might occur each time the user drags or drops an object over the pane. It should call `SET-DROP-OBJECT-SUPPORTED-FORMATS` with the *drop-object* and a list of formats that the pane wants to receive. Each format is a keyword. The list of the formats must be the same each time it is called.
- :enter** This occurs when the user drags an object over the pane. It can query the *drop-object* using `drop-object-provides-format` and `drop-object-allows-drop-effect-p` to discover what the user is dragging. It can also use `drop-object-pane-x` and `drop-object-`

`pane-y` to query the mouse position relative to the pane. It should call `(setf drop-object-drop-effect)` with an effect if it wants to allow the object to be dropped. If this is not called, then the object cannot be dropped into the pane.

`:drag` This occurs when the user continues to drag an object over the pane. It should behave as for `stage :enter` and should call `(setf drop-object-drop-effect)` if it wants to allow the object to be dropped. It might also want to update the pane to indicate where the object will be dropped.

`:drop` This occurs when the user drops an object over the pane. It can query the `drop-object` as for `:enter` but can also obtain the object itself using `drop-object-get-object` for one of the formats in the list returned by `drop-object-provides-format`. Once the object is received, it should call `(setf drop-object-drop-effect)` with the effect that has been used by the callback.

In order to display a simple pane, it needs to be contained within an interface. The two convenience functions `make-container` and `contain` are provided to create an interface with enough support for that pane. The function `make-container` just returns a container for an element, and the function `contain` displays an interface created for the pane using `make-container`.

Examples

```
(capi:contain (make-instance 'capi:output-pane
                             :background :red
                             :scroll-width 300
                             :horizontal-scroll t))

(setf ep
      (capi:contain
        (make-instance 'capi:editor-pane
                       :visible-border t)))

(setf (capi:simple-pane-cursor ep) :crosshair)
```

See also `contain`

simple-pane-handle*Function*

Summary Returns the window handle of a pane.

Package `capi`

Signature `simple-pane-handle pane => handle`

Values *handle* An integer, or `nil`.

Description The function `simple-pane-handle` returns the handle of *pane* in the system that displays it, if there is an underlying window.

On Microsoft Windows *handle* is the `hwnd` of *pane*.

On X11/Motif, *handle* is the `windowid` of the main part of *pane* (type `Window` in the X library).

If *pane* is not displayed, or if *pane* does not have an underlying window, then *handle* is `nil`. Note that layouts do not always have an underlying window.

Use this function with caution: in general, drawing and moving of CAPI windows should be done through the CAPI.

See also `current-dialog-handle`

simple-pane-visible-height

Generic Function

Summary Gets the visible height of a pane.

Package `capi`

Signature `simple-pane-visible-height pane => result`

Arguments *pane* A simple pane.

Values *result* The height of the visible part of *pane*, or `nil`.

Description The generic function `simple-pane-visible-height` returns the height in pixels of the visible part of *pane*, that is the height of the viewport, not including any borders or scroll bars. If *pane* is not displayed the function returns `nil`.

See the *LispWorks CAPI User Guide* for a description of the visible size of a pane.

See also `simple-pane-visible-size`
`simple-pane-visible-width`
`with-geometry`

simple-pane-visible-size

Generic Function

Summary Gets the visible size of a pane.

Package `capi`

Signature `simple-pane-visible-size pane => width, height`

Arguments *pane* A simple pane.

Values	<i>width</i>	The width of the visible part of <i>pane</i> , or <code>nil</code> .
	<i>height</i>	The height of the visible part of <i>pane</i> , or <code>nil</code> .
Description	<p>The generic function <code>simple-pane-visible-size</code> returns the size in pixels of the visible part of <i>pane</i>, that is the width and height of the viewport, not including any borders or scroll bars. If <i>pane</i> is not displayed the return values are <code>nil</code>.</p> <p>See the <i>LispWorks CAPI User Guide</i> for a description of the visible size of a pane.</p>	
See also	<p><code>simple-pane-visible-height</code> <code>simple-pane-visible-width</code> <code>with-geometry</code></p>	

simple-pane-visible-width

Generic Function

Summary	Gets the visible width of a pane.	
Package	<code>capl</code>	
Signature	<code>simple-pane-visible-width <i>pane</i> => <i>result</i></code>	
Arguments	<i>pane</i>	A simple pane.
Values	<i>result</i>	The width of the visible part of <i>pane</i> , or <code>nil</code> .
Description	<p>The generic function <code>simple-pane-visible-width</code> returns the width in pixels of the visible part of <i>pane</i>, that is the width of the viewport, not including any borders or scroll bars. If <i>pane</i> is not displayed the function returns <code>nil</code>.</p> <p>See the <i>LispWorks CAPI User Guide</i> for a description of the visible size of a pane.</p>	

See also `simple-pane-visible-height`
`simple-pane-visible-size`
`with-geometry`

simple-pinboard-layout

Class

Summary A `simple-pinboard-layout` is a `pinboard-layout` that can contain just one pinboard object or pane as its child, and it adopts the size constraints of that child.

Package `capi`

Superclasses `pinboard-layout`
`simple-layout`

Subclasses `graph-pane`

Initargs `:child` The child of the pinboard layout.

Description The class `simple-pinboard-layout` is normally used to place pinboard objects in a layout by placing the layout inside a `simple-pinboard-layout`, thus displaying the pinboard objects. It inherits all of its layout behavior from `simple-layout`.

Example

```
(setq column
  (make-instance
    'capi:column-layout
    :description
    (list
      (make-instance
        'capi:image-pinboard-object
        :image
        (sys:lispworks-file
         "examples/capi/graphics/lwsplash.bmp"))
      (make-instance
        'capi:item-pinboard-object
        :text "LispWorks")))
  :x-adjust :center))
```

```
(capi:contain (make-instance
               'capi:simple-pinboard-layout
               :child column))
```

See also `pinboard-object`

simple-print-port

Function

Summary Prints the contents of an output pane to a printer.

Package `capi`

Signature `simple-print-port port &key jobname scale dpi printer
interactive background`

Description The `simple-print-port` function prints the output pane specified by *port* to the default printer, unless specified otherwise by *printer*. The arguments of *scale* and *dpi* are used to determine how to transform the output pane's coordinate space to physical units. Their meaning here is the same as in `get-page-area`, except that *scale* may also take the value `:scale-to-fit`, in which case the pane is printed as large as possible on a single sheet.

The background color of the pane is ignored, and the value given by *background* is used instead. This defaults to `:white`.

If *interactive* is `t`, a print dialog is displayed. This is the default. If *interactive* is `nil`, then the document is printed to the current printer without prompting the user.

See also `print-dialog`

slider

Class

Summary A pane with a sliding marker, which allows the user to control a numerical value within a specified range.

Package	<code>capi</code>
Superclasses	<code>range-pane</code> <code>titled-object</code> <code>simple-pane</code>
Initargs	<code>:show-value-p</code> A generalized boolean. <code>:start-point</code> A keyword.
Readers	<code>slider-show-value-p</code> <code>slider-start-point</code>
Description	<p>The <code>slider</code> class allows the user to enter a number by moving a marker on a sliding scale to the desired value. <i>show-value-p</i> determines whether the slider displays the current value. The default value is <code>t</code>.</p> <p>Note: <i>show-value-p</i> is ignored on Microsoft Windows.</p> <p><i>start-point</i> specifies which end of the slider is the start point in the range. The values allowed depend on the <i>orientation</i> of the slider. For horizontal sliders, <i>start-point</i> can take these values:</p> <p><code>:left</code> The start point is on the left. <code>:right</code> The start point is on the right. <code>:default</code> The start point is at the default side (the left).</p> <p>For vertical sliders, <i>start-point</i> can take these values:</p> <p><code>:top</code> The start point is at the top. <code>:bottom</code> The start point is at the bottom. <code>:default</code> The start point is at the default position, which is the top on Microsoft Windows and Motif, and the bottom on Cocoa.</p>

sort-object-items-by*Function*

Summary	Sorts items according to a <code>sorted-object</code> .	
Package	<code>capi</code>	
Signature	<code>sort-object-items-by</code> <i>sorted-object</i> <i>items</i> => <i>result</i>	
Arguments	<i>sorted-object</i>	An instance of <code>sorted-object</code> or a subclass.
	<i>items</i>	A list.
Values	<i>result</i>	A permutation of <i>items</i> .
Description	The function <code>sort-object-items-by</code> sorts <i>items</i> according to the current sort type of <i>sorted-object</i> , as set by <code>sorted-object-sort-by</code> .	
	Note: if the sort type is reversed, <i>items</i> will be sorted in reverse order.	
See also	<code>sorted-object</code> <code>sorted-object-sort-by</code>	

sorted-object*Class*

Summary	Defines sorting operations.	
Package	<code>capi</code>	
Superclasses	<code>standard-object</code>	
Subclasses	<code>list-panel</code>	

Initargs	<code>:sort-descriptions</code> A list.
Description	The <code>sorted-object</code> class defines sorting operations. Each element of <i>sort-descriptions</i> is a sort description object, as returned by <code>make-sorting-description</code> . These define various sorting options and are used by <code>sorted-object-sort-by</code> and <code>sort-object-items-by</code> .
See also	<code>make-sorting-description</code> <code>sort-object-items-by</code> <code>sorted-object-sort-by</code>

sorted-object-sort-by

Generic Function

Summary	Sets the sorting type of a <code>sorted-object</code> .
Package	<code>capi</code>
Signature	<code>sorted-object-sort-by</code> <i>pane</i> <i>new-sort-type</i> &key <i>allow-reverse</i>
Arguments	<i>pane</i> An instance of <code>sorted-object</code> or a subclass. <i>new-sort-type</i> The sort type to set. <i>allow-reverse</i> A boolean.
Description	The generic function <code>sorted-object-sort-by</code> sets the sort type of <i>pane</i> to <i>new-sort-type</i> . <i>new-sort-type</i> must match the type of one of the sort descriptions of <i>pane</i> . If <i>allow-reverse</i> is non- <code>nil</code> and the sort type already matches <i>new-sort-type</i> , then the sort reverses the order of the <i>items</i> . The default value of <i>allow-reverse</i> is <code>t</code> .

If *pane* is a `list-panel`, then `sorted-object-sort-by` also calls `sort-object-items-by` to sort the items with the new sort type. For your own subclasses of `sorted-object` which are not subclasses of `list-panel`, if you need this behavior define an `:after` method that calls `sort-object-items-by`. You can also define `:after` methods on subclasses of `list-panel` to perform other tasks each time the items are sorted.

See also `sort-object-items-by`
`sorted-object`

start-gc-monitor

Function

Summary Starts a Lisp Monitor window.

Package `capi`

Signature `start-gc-monitor screen => result`

Arguments *screen* A screen.

Values *result* A boolean.

Description The function `start-gc-monitor` starts a Lisp Monitor window (otherwise known as the GC or Garbage Collector monitor) on the screen *screen*.

result is `t` if it started a Lisp monitor, and `nil` if a Lisp monitor was already running on *screen*.

Note that this works only on Motif.

On these platforms, `start-gc-monitor` is called automatically when the LispWorks IDE starts, but you can call `stop-gc-monitor` and `start-gc-monitor` any time.

See also `stop-gc-monitor`

stop-gc-monitor

Function

Summary	Stop a Lisp Monitor.	
Package	<code>capi</code>	
Signature	<code>stop-gc-monitor screen => result</code>	
Arguments	<code>screen</code>	A screen.
Values	<code>result</code>	A boolean.
Description	<p>The function <code>stop-gc-monitor</code> stops the Lisp Monitor window on the screen <code>screen</code>.</p> <p><code>result</code> is <code>t</code> if it stopped a Lisp monitor, and <code>nil</code> if there was no Lisp monitor running on <code>screen</code>.</p> <p>Note that this works only on Motif. The Lisp monitor can be restarted with <code>start-gc-monitor</code>.</p>	
See also	<code>start-gc-monitor</code>	

switchable-layout

Class

Summary	A subclass of <code>simple-layout</code> that displays only one of its children at a time, and provides functionality for switching the displayed child to one of the other children.	
Package	<code>capi</code>	
Superclasses	<code>simple-layout</code>	
Initargs	<code>:visible-child</code>	The currently visible pane from the children.
	<code>:combine-child-constraints</code>	A generalized boolean.

Readers	<code>switchable-layout-visible-child</code> <code>switchable-layout-combine-child-constraints</code>
Description	<p>The <code>switchable-layout</code> has a <i>description</i> which is its list of children. The argument <i>visible-child</i> specifies the initially visible child (which defaults to the first of the children).</p> <p><code>switchable-layout</code> inherits most of its layout behavior from <code>simple-layout</code> as it only ever lays out one child at a time.</p> <p><i>combine-child-constraints</i> influences the initial size of the layout. When <i>combine-child-constraints</i> is <code>nil</code> the constraints of the switchable layout depend only on its currently visible child pane. Switching to a different child pane might cause the layout to resize. When <i>combine-child-constraints</i> is non-<code>nil</code>, the constraints depend on all of the child panes, including those that are not visible. This might increase the time taken to create the switchable layout initially, but can prevent unexpected resizing later. The default value of <i>combine-child-constraints</i> is <code>nil</code>.</p>
Example	<pre>(setq children (list (make-instance 'capi:push-button :text "Press Me") (make-instance 'capi:list-panel :items '(1 2 3 4 5)))) (setq layout (capi:contain (make-instance 'capi:switchable-layout :description children))) (capi:apply-in-pane-process layout #'(setf capi:switchable-layout-visible-child) (second children) layout) (capi:apply-in-pane-process layout #'(setf capi:switchable-layout-visible-child) (first children) layout)</pre> <p>There is a further example in the file <code>examples/capi/layouts/switchable.lisp</code>.</p>

See also `layout`
`switchable-layout-switchable-children`

switchable-layout-switchable-children *Generic Function*

Summary Finds the switchable children of a `switchable-layout`.

Package `capi`

Signature `switchable-layout-switchable-children` *switchable-layout*
`=>` *result*

Arguments *switchable-layout*
An instance of `switchable-layout` or a subclass.

Values *result* A list of panes.

Description The generic function `switchable-layout-switchable-children` returns as a list all the children of *switchable-layout* that could be made visible by calling the `switchable-layout` accessor (`setf switchable-layout-visible-child`).

See also `switchable-layout`

tab-layout *Class*

Summary The class `tab-layout` has two distinct modes. Switchable mode lays a number of panes in a switchable layout. Each pane has an associated tab which, when clicked on, pulls the pane to the front. In callback mode the tabs are linked to a *selection-callback* as for `button-panel`.

Package `capi`

Superclasses	<code>choice</code> <code>layout</code>
Initargs	<p><code>:description</code> The main layout description.</p> <p><code>:items</code> Specifies the tabs of the tab layout.</p> <p><code>:visible-child-function</code> Returns the visible child for a given selection in switchable mode.</p> <p><code>:combine-child-constraints</code> A generalized boolean which influences the initial size of the layout.</p> <p><code>:key-function</code> Specifies a function to use in referring to items in the <i>items</i> list.</p> <p><code>:print-function</code> The function used to print a name on each tab.</p> <p><code>:callback-type</code> The type of data passed to the callback function in callback mode.</p> <p><code>:selection-callback</code> The function called when a tab is selected, in callback mode.</p>
Accessors	<code>tab-layout-visible-child-function</code>
Readers	<code>tab-layout-combine-child-constraints</code>
Description	<p>A <code>tab-layout</code> has one of two distinct modes. It is in switchable mode if <i>visible-child-function</i> is supplied and non-<code>nil</code>. It is in callback mode otherwise.</p> <p>In switchable mode, the tab layout consists of a number of panes, each with its own tab. Clicking on a tab pulls the corresponding pane to the front. In this mode the tab layout is</p>

like a `switchable-layout` with the switching performed by the user selecting a tab. In this mode the *visible-child-function* is used to specify which child to make visible for a given tab selection.

In callback mode the tab layout does not work as a switchable layout, and the result of any selection is specified using a callback specified by *selection-callback*, in a similar way to a `button-pane` callback. In this mode the *description* slot is used to describe the main layout of the tab pane.

In either mode *combine-child-constraints* influences the initial size of the layout. When *combine-child-constraints* is `nil` the constraints of the tab layout depend only on its currently visible tab. Switching to a different tab might cause the layout to resize. When *combine-child-constraints* is non-`nil`, the constraints depend on all of the tabs, including those that are not visible. This might increase the time taken to create the tab layout initially, but can prevent unexpected resizing later. The default value of *combine-child-constraints* is `nil`.

Examples

The following example shows the use of the switchable mode of `tab-layout`. Each tab is linked to an output pane by pairing them in the *items* list.

```
(defun switchable-tab-layout ()
  (let* ((red-pane (make-instance
                   'capi:output-pane
                   :background :red))
        (blue-pane (make-instance
                    'capi:output-pane
                    :background :blue))
        (tl (make-instance
             'capi:tab-layout
             :items
             (list (list "Red" red-pane)
                  (list "Blue" blue-pane))
             :print-function 'car
             :visible-child-function 'second)))
    (capi:contain tl)))

(switchable-tab-layout)
```

Here is an example of the callback mode of `tab-layout`, which uses the selection of a tab to change the nodes of a graph pane through the *selection-callback*.

```
(defun non-switchable-tab-layout (tabs)
  (let* ((gp (make-instance
              'capi:graph-pane))
         (tl (make-instance
              'capi:tab-layout
              :description (list gp)
              :items tabs
              :visible-child-function nil
              :key-function nil
              :print-function
              (lambda (x)
                (format nil "~R" x))
              :callback-type :data
              :selection-callback
              #'(lambda (data)
                  (setf (capi:graph-pane-roots gp)
                        (list data))))))
    (capi:contain tl)))

(non-switchable-tab-layout '(1 2 4 5 6))
```

See also

`callbacks`
`simple-layout`
`switchable-layout`
`tab-layout-panes`
`tab-layout-visible-child`

tab-layout-panes

Function

Summary	Returns the panes in a <code>tab-layout</code> .
Package	<code>capi</code>
Signature	<code>tab-layout-panes</code> <i>tab-layout</i> => <i>panes</i>
Arguments	<i>tab-layout</i> A <code>tab-layout</code> .

Values	<i>panes</i>	A list.
Description	The function <code>tab-layout-panes</code> returns the panes in a <code>tab-layout</code> . Note that this is not necessarily the same as the items of <code>tab-layout</code> , since <i>visible-child-function</i> and/or <i>key</i> may be specified.	
See also	<code>tab-layout</code>	

tab-layout-visible-child

Function

Summary	Returns the visible child in a <code>tab-layout</code> .	
Package	<code>capi</code>	
Signature	<code>tab-layout-visible-child tab-layout => result</code>	
Arguments	<i>tab-layout</i>	A <code>tab-layout</code> .
Values	<i>result</i>	A pane.
Description	The function <code>tab-layout-visible-child</code> returns the currently-visible pane in a <code>tab-layout</code> .	
See also	<code>tab-layout</code>	

text-input-choice

Class

Summary	This pane consists of a text input area, and a button. Clicking on the button displays a drop-down list of strings, and selecting one of the strings automatically pastes it into the text input area.	
Package	<code>capi</code>	

Superclasses	<code>choice</code> <code>text-input-pane</code>
Initargs	<code>:visible-items-count</code> An integer specifying the maximum length of the drop-down list, or the symbol <code>:default</code> . <code>:popup-callback</code> A function called just before the drop-down list appears, or <code>nil</code> .
Description	The <code>text-input-choice</code> class behaves in the same way as a <code>text-input-pane</code> , but has additional functionality. The element inherits from <code>choice</code> , and the choice <i>items</i> are used as the items to display when the user clicks on the button. The <i>callback</i> is called when the user presses the <code>Return</code> key. The <i>selection-callback</i> is called when the user selects an item using the drop-down list.
See also	<code>choice</code> <code>text-input-pane</code>

text-input-pane

Class

Summary	The class <code>text-input-pane</code> is a pane for entering a single line of text.
Package	<code>capi</code>
Superclasses	<code>titled-object</code> <code>simple-pane</code>
Subclasses	<code>multi-line-text-input-pane</code> <code>password-pane</code> <code>text-input-choice</code>

Initargs	:text	The text in the pane.
	:caret-position	The position of the caret in the text (from 0).
	:max-characters	The maximum number of characters allowed.
	:enabled	Controls the enabled state of the pane.
	:completion-function	A function called to complete the text.
	:callback-type	The type of arguments to <i>callback</i> .
	:callback	A function usually called when the user presses Return.
	:change-callback-type	The type of arguments to <i>callback</i> .
	:change-callback	A function called when a change is made.
	:confirm-change-function	A function called to validate a change. Note: Implemented for Motif only, not Microsoft Windows or Mac OS X.
	:navigation-callback	A function called when certain keyboard gestures occur in the pane.
	:editing-callback	A function called when editing starts or stops.
	:buttons	A plist specifying buttons to add, or <code>t</code> or <code>nil</code> .

Accessors

```

text-input-pane-text
text-input-pane-max-characters
text-input-pane-completion-function
text-input-pane-callback
text-input-pane-confirm-change-function
text-input-pane-change-callback
text-input-pane-navigation-callback
text-input-pane-editing-callback
text-input-pane-enabled
text-input-pane-buttons-enabled

```

Readers

```

text-input-pane-caret-position

```

Description

The class `text-input-pane` provides a great deal of flexibility in its handling of the text being entered. It starts with the initial text and *caret-position* specified by the arguments *text* and *caret-position* respectively. It limits the number of characters entered with the *max-characters* argument (which defaults to `nil`, meaning there is no maximum).

If *enabled* is `nil`, the pane is disabled. If *enabled* is `:read-only`, then the pane shows the text and allows it to be selected without it being editable. In this case the visual appearance varies between window systems, but often the text can be copied and the caret position altered. If *enabled* is any other true value, then the pane is fully enabled. The default value of *enabled* is `t`.

A *completion-function* can be specified which will get called when the completion gesture is made by the user (or pressing the `Tab` key) or when `text-input-pane-complete-text` is called. The *completion-function* is called with the pane and the text to complete and should return either `nil`, the completed text or a list of candidate completions. In the latter case, the CAPI will prompt the user for the completion they wish, and this will become the new text.

callback, if non-`nil`, is called when the user presses `Return`, unless *navigation-callback* is non-`nil`, in which case *navigation-callback* is called instead.

When the *text* or *caret-position* is changed, the callback *change-callback* is called with the *text*, the pane itself, the interface and the *caret-position*. The arguments that are passed to the *change-callback* can be altered by specifying the *change-callback-type* (see the `callbacks` class for details of possible values).

Note: the *change-callback* is potentially called more than once for each user gesture.

With the Motif implementation it is possible to check changes that the user makes to the `text-input-pane` by providing a *confirm-change-function* which gets passed the new text, the pane itself, its interface and the new caret position, and which should return `non-nil` if it is OK to make the change. If `nil` is returned, then the pane will be unaltered and a beep will be signalled to indicate that the new values were invalid.

navigation-callback, if `non-nil`, is a function that will be called when certain navigation gestures are used in the `text-input-pane`. The function is called with two arguments, the pane itself, and one of the following keywords:

`:tab-forward`

Tab was pressed.

`:tab-backward` Tab Backwards (usually `shift+Tab`) was pressed.

`:return` Return was pressed.

`:shift-return` Shift+Return was pressed.

`:enter` Enter was pressed.

`:shift-enter`

Shift+Enter was pressed.

Note: `Enter` is the key usually found on the numeric keypad.

When *navigation-callback* is `non-nil`, it is called instead of *callback* when `Return` is pressed. *callback* is still called via an OK button if there is one (see *buttons* below).

navigation-callback is implemented only on Microsoft Windows and Cocoa.

editing-callback, if non-`nil`, is a function of two arguments:

`editing-callback` *pane* *type*

pane is the `text-input-pane` and *type* is a keyword. *editing-callback* is called with *type* `:start` when the user starts editing and *type* `:end` when the user stops editing. In general, this occurs when the focus changes, but on Cocoa *type* `:start` is passed when the first change is made to the text.

buttons specifies toolbar buttons which appear next to the pane and facilitate user actions on it. It also specifies the position of the buttons relative to the pane. This feature appears in the Common LispWorks IDE, for example the Class box of the Class Browser.

The allowed keys and values of the plist *buttons* are:

- `:ok` A boolean or a plist, default value `t`. If true, a button which calls *callback* appears. If the value is a plist then this plist supplies details for the button, as described below.
- `:cancel` A boolean or a plist, default value `nil`. If true, a button which calls *cancel-function* appears. A plist value is interpreted as for `:ok` and can also contain the key `:accelerator` which specifies an accelerator used for the button. There is no default accelerator.
- `:completion` A boolean or a plist. If true, a button which calls *completion-function* appears. The default value is `t` if *completion-function* is non-`nil`, and `nil` otherwise. A plist value is interpreted as for `:ok`.

:browse-file

A keyword or a plist. If true, a button which invokes `prompt-for-file` appears. If the value is `:save` or `:open` then it is passed as the operation argument to `prompt-for-file`, replacing the text in the pane if successful. If the value is a plist, then it supplies details for the button, as described below, and can also contain the keywords `:message` to specify a message for the file prompter; `:pathname` to specify the default pathname of the file prompter (defaults to the text in the `text-input-pane`) or any of the keywords `:ok-check`, `:filter`, `:filters`, `:if-exists`, `:if-does-not-exist`, `:operation`, `:owner`, `:pane-args` or `:popup-args` which are passed directly to `prompt-for-file`.

:cancel-function

A function that expects the pane as its single argument. The default is a function which sets *text* to the empty string.

:help

Specifies a help button. The value must be a plist containing either keys `:function` and optionally `:arguments`, or the keys `:title`, `:message` and optionally `:dialog-p`.

If *function* is supplied, when the user presses the help button it calls

`(apply function pane arguments)`

where *pane* is the `text-input-pane`. *title*, *message* and *dialog-p* are ignored in this case.

Otherwise when the user presses the help button it opens a window with title *title* displaying the string *message* in a `display-pane`. The message can be long, and can include newlines. The window is owned by

the pane, but is not modal, so the user can interact with the pane while the help window is displayed. If *dialog-p* is non-`nil`, the help window is raised as a dialog. The default value for *dialog-p* is `nil`. *function* and *arguments* are ignored in this case.

The plist can contain other keys as described below.

:orientation

The value is either `:horizontal` or `:vertical`. *orientation* controls the orientation of the toolbar. This is useful for `multi-line-text-input-pane`. The default value is `:horizontal`.

:adjust

The value is `:top`, `:center`, `:centre` or `:bottom`. *adjust* controls how the buttons are adjusted vertically relative to the text input pane. This is useful for `multi-line-text-input-pane`. The default value is `:center`.

:position

The value is `:top`, `:bottom`, `:left` or `:right`. *position* determines whether the buttons appear above, below, left or right of the text input pane. If *position* is not supplied, then the buttons appear to the right of the pane.

The value `nil` for *buttons* means there are no buttons - this is the default. When *buttons* is true the buttons appear or not according to their specified values or their default values.

All of the button plists (for `:ok`, `:cancel`, `:help` and so on) can contain the following keys and values in addition to those mentioned above:

:enabled

A value that controls whether the button is enabled. (See the reader `text-input-pane-buttons-enabled`).

- `:image` The image to use for the button. This should be either a pathname or string naming an image file to load, a symbol giving the id of an image registered with `register-image-translation`, an `image` object as returned by `load-image` or an `external-image`. The default image is one of the symbols `ok-button`, `cancel-button` or `complete-button`, which are preregistered image identifiers corresponding to each button.
- `:help-key` The *help-key* used to find a tooltip for the button.

The `text-input-pane-buttons-enabled` reader returns a list containing keywords such as `:ok`, `:cancel` and `:completion`, one for each corresponding button (as specified by *buttons*) that is currently enabled.

The `(setf text-input-pane-buttons-enabled)` writer takes a list of keywords as described for the reader and sets the enabled state of the buttons, enabling each button if it appears in the list and disabling it otherwise. The value `t` can also be passed: this enables all the buttons.

For more than one line of input, use `multi-line-text-input-pane`.

Compatibility Note The *confirm-change-function* was called *before-change-callback* in LispWorks 3.1. Both the old initarg `before-change-callback` and the old accessor `text-input-pane-before-change-callback` are still supported, but may not be in future releases.

Examples

```
(capi:contain (make-instance 'capi:text-input-pane
                             :text "Hello world"))
```

```
(setq tip (capi:contain
           (make-instance
            'capi:text-input-pane
            :enabled nil)))

(capi:apply-in-pane-process
 tip #'(setf capi:text-input-pane-enabled) t tip)

(capi:apply-in-pane-process
 tip #'(setf capi:text-input-pane-enabled) nil tip)

(capi:apply-in-pane-process
 tip #'(setf capi:text-input-pane-text) "New text" tip)

(capi:contain (make-instance
               'capi:text-input-pane
               :text "Hello world"
               :callback #'(lambda (text interface)
                            (capi:display-message
                             "Interface ~S's text: ~S"
                             interface text))))
```

This example uses a plist value for the *buttons* key `:cancel` to specify that the Cancel button is initially disabled:

```
(capi:contain
 (make-instance 'capi:text-input-pane
                :buttons
                '(:ok t :cancel (:enabled nil))))
```

This example shows how to specify a Help button which displays a help message:

```
(defvar *help-message* "A long help message.")

(capi:contain
 (make-instance 'capi:text-input-pane
                :buttons
                `(:help
                  (:title "help window"
                           :message ,*help-message*))))
```

There is a further example in the file `examples/capi/elements/text-input-pane.lisp`

See also `display-pane`
`editor-pane`
`multi-line-text-input-pane`
`text-input-choice`
`text-input-pane-complete-text`
`text-input-range`
`title-pane`

text-input-pane-complete-text

Function

Summary Calls the *completion-function* in a `text-input-pane`.

Package `capi`

Signature `text-input-pane-complete-text pane => result`

Arguments `pane` A `text-input-pane`.

Values `result` A string, or `nil`.

Description The function `text-input-pane-complete-text` calls the *completion-function* of `pane` with the current *text*. If this call is successful, then the *text* of `pane` is set to the result, and `text-input-pane-complete-text` returns this result. Otherwise, *result* is `nil`.

Note: the *completion-function* may return a list of completion candidates, in which case `text-input-pane-complete-text` prompts the user to select one of the candidates.

See also `text-input-pane`

text-input-pane-copy*Function*

Summary	Copies the selected text in a <code>text-input-pane</code> to the clipboard
Package	<code>capi</code>
Signature	<code>text-input-pane-copy</code> <i>text-input-pane</i>
Arguments	<i>text-input-pane</i> An instance of <code>text-input-pane</code> or a subclass.
Description	The function <code>text-input-pane-copy</code> performs the clipboard copy operation on the selected text in <i>text-input-pane</i> . It does nothing if there is no selection.
See also	<code>clipboard</code> <code>text-input-pane</code> <code>text-input-pane-selection</code> <code>text-input-pane-cut</code> <code>text-input-pane-delete</code> <code>text-input-pane-paste</code>

text-input-pane-cut*Function*

Summary	Cuts the selected text in a <code>text-input-pane</code> to the clipboard
Package	<code>capi</code>
Signature	<code>text-input-pane-cut</code> <i>text-input-pane</i>
Arguments	<i>text-input-pane</i> An instance of <code>text-input-pane</code> or a subclass.

Description The function `text-input-pane-cut` performs the clipboard cut operation on the selected text in *text-input-pane*. It does nothing if there is no selection.

See also `clipboard`
 `text-input-pane`
 `text-input-pane-selection`
 `text-input-pane-copy`
 `text-input-pane-delete`
 `text-input-pane-paste`

text-input-pane-delete

Function

Summary Deletes the selected text in a `text-input-pane`.

Package `capi`

Signature `text-input-pane-delete` *text-input-pane*

Arguments *text-input-pane* An instance of `text-input-pane` or a subclass.

Description The function `text-input-pane-delete` deletes the selected text in *text-input-pane*. It does nothing if there is no selection.

See also `clipboard`
 `text-input-pane`
 `text-input-pane-selection`
 `text-input-pane-cut`
 `text-input-pane-copy`
 `text-input-pane-paste`

text-input-pane-paste

Function

Summary Pastes the clipboard text into a `text-input-pane`.

Package	<code>capi</code>
Signature	<code>text-input-pane-paste</code> <i>text-input-pane</i>
Arguments	<i>text-input-pane</i> An instance of <code>text-input-pane</code> or a subclass.
Description	The function <code>text-input-pane-paste</code> performs the clipboard paste operation on <i>text-input-pane</i> , replacing any selected text.
See also	<code>clipboard</code> <code>text-input-pane</code> <code>text-input-pane-selection</code> <code>text-input-pane-cut</code> <code>text-input-pane-copy</code> <code>text-input-pane-delete</code>

text-input-pane-selected-text

Function

Summary	Returns the selected text in a <code>text-input-pane</code> .
Package	<code>capi</code>
Signature	<code>text-input-pane-selected-text</code> <i>text-input-pane</i> => <i>result</i>
Arguments	<i>text-input-pane</i> An instance of <code>text-input-pane</code> or a subclass.
Values	<i>result</i> A string or <code>nil</code> .
Description	The function <code>text-input-pane-selected-text</code> returns the selected text in <i>text-input-pane</i> , or <code>nil</code> if there is no selection.

See also `text-input-pane`
`text-input-pane-selection`
`text-input-pane-selection-p`

text-input-pane-selection

Function

Summary Returns the bounds of the selection in a `text-input-pane`.

Package `capi`

Signature `text-input-pane-selection pane => start, end`

Arguments `pane` A `text-input-pane`.

Values `start, end` Non-negative integers.

Description The function `text-input-pane-selection` returns as multiple values the bounding indexes of the selection in `pane`. That is, `start` is the inclusive index of the first selected character, and `end` is one greater than the index of the last selected character.

If there is no selection, then both `start` and `end` are the caret position in `pane`.

See also `set-text-input-pane-selection`
`text-input-pane`
`text-input-pane-selected-text`
`text-input-pane-selection-p`

text-input-pane-selection-p

Function

Summary Returns true if there is selected text in a `text-input-pane`.

Package `capi`

Signature	<code>text-input-pane-selection-p</code> <i>pane</i> => <i>selectionp</i>
Arguments	<i>pane</i> A <code>text-input-pane</code> .
Values	<i>selectionp</i> A boolean.
Description	The function <code>text-input-pane-selection-p</code> returns <code>t</code> if there is a selected region in <i>pane</i> and <code>nil</code> otherwise.
See also	<code>set-text-input-pane-selection</code> <code>text-input-pane</code> <code>text-input-pane-selected-text</code> <code>text-input-pane-selection</code>

text-input-range

Class

Summary	The class <code>text-input-range</code> is a pane for entering a number in a given range. Typically there are up and down buttons at the side which can used to quickly adjust the value.	
Package	<code>capi</code>	
Superclasses	<code>titled-object</code> <code>simple-pane</code>	
Initargs	<code>:start</code>	An integer specifying the lowest possible value in the range.
	<code>:end</code>	An integer specifying the highest possible value in the range.
	<code>:wraps-p</code>	A generalized boolean.
	<code>:value</code>	An integer specifying the current value in the pane.
	<code>:callback</code>	A function called when the value is changed by the user.

`:callback-type` The type of arguments passed to the callback.

Accessors

- `text-input-range-start`
- `text-input-range-end`
- `text-input-range-wraps-p`
- `text-input-range-value`
- `text-input-range-callback`
- `text-input-range-callback-type`

Description

The class `text-input-range` provides numeric input of integers in a given range (some systems refer to this a spinner or spin-box).

The range is controlled by the `:start` and `:end` initargs. `start` defaults to 0 and `end` defaults to 10. The initial value is set with the argument `value` (which defaults to 0).

`wraps-p` controls what happens if the user presses the up or down button until the start or end is reached. If `wraps-p` is `nil`, then it stops at the limit. If `wraps-p` is true then it wraps around to the other end. The default value of `wraps-p` is `nil`.

`callback` provides a function to be called whenever the value is changed by the user. The arguments to this function are specified by `callback-type` (see the `callbacks` class for details of possible values, noting that the "data" is the value and the "item" is the pane itself). The default `callback-type` is `(:item :data)`.

Examples

```
(capi:contain
  (make-instance 'capi:text-input-range
                 :start 0
                 :end 100
                 :value 42))
```

See also

- `text-input-pane`
- `text-input-choice`
- `option-pane`

title-pane*Class*

Summary	This class provides a pane that displays a single line of text.
Package	<code>capi</code>
Superclasses	<code>titled-object</code> <code>simple-pane</code>
Subclasses	<code>message-pane</code>
Initargs	<code>:text</code> The text to appear in the title pane.
Accessors	<code>title-pane-text</code>
Description	<p>The most common use of title panes is as a title decoration for a pane, and so the class <code>titled-object</code> is provided as a class that supports placing title panes around itself.</p> <p>A <code>title-pane</code> with <i>text</i> "Title" is created automatically when a <code>titled-object</code> is created with <i>title</i> "Title".</p> <p>By default, a <code>title-pane</code> is constrained so that it cannot resize (that is, the values of <i>visible-max-width</i> and <i>visible-max-height</i> are <code>t</code>). This can be overridden by passing <code>:visible-max-width nil</code> or <code>:visible-max-height nil</code>.</p>
Examples	<pre>(setq title-pane (capi:contain (make-instance 'capi:title-pane :text "This is a title pane"))) (capi:apply-in-pane-process title-pane #'(setf capi:title-pane-text) "New title" title-pane)</pre>
See also	<code>display-pane</code> <code>text-input-pane</code> <code>editor-pane</code>

titled-menu-object

Class

Summary	The class <code>titled-menu-object</code> is a subclass of <code>menu-object</code> which supports titles, and it is used by menus, menu components and menu items.
Package	<code>capi</code>
Superclasses	<code>menu-object</code>
Subclasses	<code>menu</code> <code>menu-component</code> <code>menu-item</code>
Initargs	<code>:title</code> The title for the object. <code>:title-function</code> A setup callback which returns the title for the object, and optionally a mnemonic for the title.
Accessors	<code>menu-title</code> <code>menu-title-function</code>
Description	<p>The simplest way to give a title to a <code>titled-menu-object</code> is to just supply a <i>title</i> string, and this will then appear as the title of the object.</p> <p>Alternatively, a <i>title-function</i> can be provided which will be called when the menu is about to appear and which should return the title to use. By default <i>title-function</i> is called on the interface of the <code>titled-menu-object</code>, but this argument can be changed by passing the <code>menu-object</code> initarg <i>setup-callback-argument</i>.</p> <p>To specify a mnemonic in the title returned by <i>title-function</i>, make <i>title-function</i> return the mnemonic as a second value. This value is interpreted in the same way as the <i>mnemonic</i> argument for <code>menu</code>.</p>

```

Examples  (capi:contain (make-instance 'capi:menu-item
                                     :title "Press Me"))

          (capi:contain (make-instance
                         'capi:menu-item
                         :title-function #'(lambda (item)
                                             (princ-to-string
                                              (random 5))))))

```

titled-object*Class*

Summary The class `titled-object` is a mixin class which provides support for decorating a pane with a title (a piece of text positioned next to the pane) and with a message (a piece of text below the pane).

Package `capi`

Subclasses

```

interface
layout
title-pane
display-pane
text-input-pane
toolbar
button-panel
list-panel
option-pane
progress-bar
output-pane
slider

```

Initargs

```

:title           A title string for the pane (or nil).
:title-args      Initargs to the title make-instance.
:title-font      The font used for the title.
:title-position  The position of the title.
:title-adjust    How to adjust the title relative to the pane.
:title-gap       The gap between the title and the pane.

```

:message A message string for the pane (or `nil`).

:mnemonic-title
 A string specifying the title and a
 mnemonic. Applies only to the subclasses
 specified below.

:message-gap
 The gap between the message and the pane.

Accessors `titled-object-title`
 `titled-object-title-font`
 `titled-object-message`
 `titled-object-message-font`

Description The titled pane makes its title decoration from a `title-pane`
 and the message decoration from a `message-pane`.

The *text* of the `title-pane` is passed via the `titled-object`
 initarg *title* and the *text* of the `message-pane` is passed via the
`titled-object` initarg *message*.

The initargs and font for the `title-pane` are passed via the
`titled-object` initargs *title-args* and *title-font* respectively.

title-gap specifies the size in pixels of the gap between the title
 and the pane. The default value of *title-gap* is 3.

For subclasses other than `interface`, the font used for the
message can be found by `titled-object-message-font` and
 set by `(setf titled-object-message-font)`.

message-gap specifies the size in pixels of the gap between the
 message and the pane. The default value of *message-gap* is 3.

The message is always placed below the pane, but the title's
 position can be adjusted by specifying *title-position* which can
 be any of the following.

:left Place the title to the left of the pane.

:right Place the title to the right of the pane.

:top Place the title above the pane.

`:bottom` Place the title below the pane.
`:frame` Place the title in a frame (like a groupbox) around the pane.

The *title-adjust* slot is used to adjust the title so that it is left justified, right justified or centered. The value of *title-adjust* can be any of the values accepted by the function `pane-adjusted-offset`, which are `:left`, `:right`, `:top`, `:bottom`, `:center` and `:centre`.

mnemonic-title offers an alternate way to provide the pane's title, and with a mnemonic. It takes effect only for `button-panel`, `list-panel`, `list-view`, `option-pane`, `output-pane`, `progress-bar`, `scroll-bar`, `slider`, `text-input-pane`, `text-input-range`, `tree-view` and their subclasses, and is interpreted as described for `menu`.

Note: titles and mnemonic titles can now be added in a `grid-layout`.

Examples

Try each of these examples to see some of the effects that titled panes can produce. Note that `text-input-pane` is a subclass of `titled-object`, and that it has a default *title-position* of `:left`.

```
(capi:contain (make-instance 'capi:text-input-pane))

(capi:contain (make-instance 'capi:text-input-pane
                             :title "Enter some text:"))

(capi:contain (make-instance
               'capi:text-input-pane
               :title "Enter some text:"
               :title-position :top))

(capi:contain (make-instance 'capi:text-input-pane
                             :title "Enter some text:"
                             :title-position :top
                             :title-adjust :center))

(capi:contain (make-instance 'capi:text-input-pane
                             :title "Enter some text:"
                             :title-position :top
                             :title-adjust :right))
```

```
(capi:contain (make-instance 'capi:text-input-pane
                             :message "A message"))

(capi:contain (make-instance 'capi:text-input-pane
                             :message "A message"
                             :title "Enter some text:"))

(capi:contain (make-instance 'capi:text-input-pane
                             :title "Enter some text:"
                             :title-args
                             '(:foreground :red)))
```

Compatibility Note `titled-object` corresponds to the LispWorks 4.1 class `titled-pane`. For backwards compatibility the accessors `titled-pane-title` and `titled-pane-message`, including `setf` methods, are provided. These simply trampoline to `titled-object-title` and `titled-object-message`, and may not be supported in future releases.

See also `message-pane`
`title-pane`

titled-pinboard-object

Class

Summary A pinboard object with a title.

Package `capi`

Superclasses `pinboard-object`
`titled-object`

Subclasses `image-pinboard-object`

Description The class `titled-pinboard-object` provides a pinboard object with a title. The title is regarded as part of the object in geometry calculations.

Note: `titled-pinboard-object` does not allow the value `:frame` for the `titled-object` initalarg *title-position*. The values `:top`, `:bottom`, `:left` and `:right` are allowed.

Examples

This example creates three instances of `titled-pinboard-object` and one of `item-pinboard-object`, all with with a yellow background. Note that:

1. The title does not have the yellow background in the `titled-pinboard-object`, as opposed to the `item-pinboard-object`. To specify the title background, we pass it in the *title-args*.
2. The width of the title area is determined by the title, but passing `:visible-min-width` (and other geometric hints) can be used to override this.
3. Setting the `titled-object-title` of the `titled-pinboard-object` does not reset its width.

```

(setq tpo1 (make-instance 'capi:titled-pinboard-object
                          :graphics-args
                          '(:background :yellow)
                          :x 10 :y 10
                          :width 150 :height 20
                          :title "Short"
                          :title-position :left
                          :title-args
                          '(:background :red ))
  tpo2 (make-instance 'capi:titled-pinboard-object
                      :graphics-args
                      '(:background :yellow)
                      :x 10 :y 40
                      :width 150 :height 20
                      :title "Long title"
                      :title-position :left)
  tpo3 (make-instance 'capi:titled-pinboard-object
                      :graphics-args
                      '(:background :yellow)
                      :x 10 :y 70
                      :width 150 :height 20
                      :title "Short"
                      :title-position :left
                      :title-args
                      '(:visible-min-width 100))
  ipo (make-instance 'capi:item-pinboard-object
                    :graphics-args
                    '(:background :yellow)
                    :x 10 :y 100
                    :width 150 :height 20
                    :text "Item Pinboard" ))

(setq pl (capi:contain
         (make-instance 'capi:pinboard-layout
                       :visible-min-width 200
                       :visible-min-height 200
                       :description
                       (list tpo1 tpo2 tpo3 ipo))))

(capi:apply-in-pane-process
 pl
 #'(lambda()
     (setf (capi:titled-object-title tpo1)
           "Longer...")))

```

See also `item-pinboard-object`

toolbar*Class*

Summary	This class provides a pane containing toolbar buttons and panes.	
Package	<code>capi</code>	
Superclasses	<code>collection</code> <code>simple-pane</code> <code>titled-object</code> <code>toolbar-object</code>	
Initargs	<code>:dividerp</code>	If <code>t</code> , a divider line is drawn above the toolbar, to separate it from the menu bar. The default value is <code>nil</code> .
	<code>:images</code>	A list of images.
	<code>:callbacks</code>	A list of callback functions.
	<code>:tooltips</code>	A list of tooltip strings used on Microsoft Windows.
	<code>:button-width</code>	The width of the toolbar buttons.
	<code>:button-height</code>	The height of the toolbar buttons.
	<code>:stretch-text-p</code>	A generalized boolean.
	<code>:image-width</code>	The width of images in the toolbar.
	<code>:image-height</code>	The height of images in the toolbar.
	<code>:default-image-set</code>	An optional <code>image-set</code> object which can be used to specify images. See <code>toolbar-button</code> and <code>image-set</code> for more details.
	<code>:flatp</code>	A generalized boolean.
Readers	<code>toolbar-flat-p</code>	

Description The class `toolbar` inherits from `collection`, and therefore has a list of *items*. It behaves in a similar manner to `push-button-panel`, which inherits from `choice`.

The *items* argument may be used to specify a mixture of `toolbar-buttons` and `toolbar-components`, or it may contain arbitrary objects as items. The list may also contain CAPI panes, which will appear within the toolbar. This is typically used with `text-input-pane`, `option-pane`, and `text-input-choice`.

For items that are not toolbar buttons or toolbar components, a toolbar button is automatically created, using the appropriate elements of the *images*, *callbacks* and *tooltips* lists. If no image is specified, the item itself is used as the image. For more information on acceptable values for images, see `toolbar-button`.

Each of the *images*, *callbacks* and *tooltips* lists should be in one-to-one correspondence with the items. Elements of these lists corresponding to `toolbar-button` items or `toolbar-component` items are ignored.

Note: `:tooltips` is now deprecated. Use the interface `help-callback` with `help-key :tooltip` instead.

All toolbar buttons within the item list behave as push buttons. However, toolbar button components may have `:single-selection` or `:multiple-selection` interaction. See `toolbar-component` for further details.

button-width and *button-height* specify the size of each button in the toolbar. If a button contains text and *stretch-text-p* is true, then the button stretches to the width of the toolbar if needed.

images, if supplied, must specify images all of the same size.

image-width and *image-height* must match the sub-image dimensions in *default-image-set* or the dimensions of the *images*.

flatp specifies whether the toolbar is 'flat' on Cocoa. If *flatp* is true, then the buttons do not have a visible outline until the user moves the mouse over them. *flatp* is only implemented on Cocoa. (On Microsoft Windows, all toolbars are flat. On Motif, no toolbar is flat.) The default value of *flatp* is `:default`.

See also `collection`
`image-set`
`push-button-panel`
`toolbar-component`

toolbar-button

Class

Summary This class is used to create instances of toolbar buttons.

Package `capi`

Superclasses `item`
`toolbar-object`

Initargs

- `:callback` A function that is called when the user presses the toolbar button and *popup-interface* is non-`nil`.
- `:image` Specifies the image to use for the toolbar button.
- `:selected-image` Specifies the image to use for the toolbar button when it is selected.
- `:tooltip` An optional string which is displayed, on Microsoft Windows, when the mouse moves over the button. `:tooltip` is deprecated.
- `:help-key` An object used for lookup of help. Default value `t`.

`:remapped` Links the button to a menu item.

`:dropdown-menu`
A menu or `nil`.

`:dropdown-menu-function`
A function of no arguments, or `nil`.

`:dropdown-menu-kind`
One of the keywords `:button`, `:only` and `:delayed`.

`:popup-interface`
An interface or `nil`.

Accessors

`toolbar-button-image`
`toolbar-button-selected-image`
`toolbar-button-dropdown-menu`
`toolbar-button-dropdown-menu-function`
`toolbar-button-dropdown-menu-kind`
`toolbar-button-popup-interface`

Readers

`help-key`

Description

Toolbar buttons may be placed within toolbars and toolbar components. However, there is usually no need to create toolbar buttons explicitly; instead, the *callbacks* and *images* arguments to `toolbar` or `toolbar-component` can be used. To add tooltips, use the interface *help-callback* with `help-key :tooltip`.

image and *selected-image* may each be one of the following:

A pathname or string

This specifies the filename of a file suitable for loading with `load-image`. Currently this must be a bitmap file.

A symbol

The symbol must either have been previously registered by means of a call to `register-image-translation`, or be one of the following symbols, which map to

standard images: `:std-cut`, `:std-copy`,
`:std-paste`, `:std-undo`, `:std-redo`,
`:std-delete`, `:std-file-new`,
`:std-file-open`, `:std-file-save`,
`:std-print`, `:std-print-pre`,
`:std-properties`, `:std-help`, `:std-find`
and `:std-replace`

An image object, as returned by `load-image`.

An image locator object

This allows a single bitmap to be created which contains several button images side by side. See `make-image-locator` for more information. On Microsoft Windows, this also allows access to bitmaps stored as resources in a DLL.

An integer This is a zero-based index into the *default-image-set* of the toolbar or toolbar component in which the toolbar button is used.

Each image should be of the correct size for the toolbar. By default, this is 16 pixels wide and 16 pixels high.

help-key is interpreted as described for `element`.

remapped, if non-`nil`, should match the *name* of a `menu-item` in the same interface as the button. Then, the action of pressing the button is remapped to selecting that `menu-item` and calling its *callback*. The default value of *remapped* is `nil`.

Toolbar buttons can be made with an associated dropdown menu by passing the `:dropdown-menu` or `:dropdown-menu-function` initargs.

If *dropdown-menu* is non-`nil` then it should be a `menu` object to display for the button.

If *dropdown-menu-function* is non-`nil` then it should be a function which will be called with no arguments and should return a `menu` object to display for the button.

dropdown-menu-kind can have the following values:

- :button** There is a separate smaller button for the dropdown menu next to the main button.
- :only** There is no main button, only the smaller button for the dropdown.
- :delayed** There is only one button and the menu is displayed when the user holds the mouse down over the button for some short delay. If the user clicks on the button then the normal *callback* is called.

popup-interface, if non-`nil`, should be an `interface`. When the user clicks on the toolbar button, the interface *popup-interface* is displayed near to the button. The normal *callback* is not called, but you can detect when the interface appears by using its *activate-callback*. *popup-interface* is useful for popping up windows with more complex interaction than a menu can provide. The default value of *popup-interface* is `nil`.

Toolbar buttons can display text, which should be in the *data* or *text* slot inherited from `item`.

Note: display of text in toolbar buttons is implemented only on Motif and Cocoa.

Example

A callback function:

```
(defun do-redo (data interface)
  (declare (ignorable data interface))
  (capi:display-message "Doing Redo"))
```

A simple interface:

```

(capi:define-interface redo ()
  ()
  (:panes
   (toolbar
    capi:toolbar
    :items
    (list
     (make-instance
      'capi:toolbar-component
      :items
      (list (make-instance
              'capi:toolbar-button
              ;; remap it to the menu item
              :remapped 'redo-menu-item
              :image :std-redo))))))
   (:menu-bar a-menu)
   (:menus
    (a-menu
     "A menu"
     (("Redo" :name 'redo-menu-item
              :selection-callback 'do-redo
              :accelerator #\control-\y))))
   (:layouts
    (main
     capi:row-layout
     '(toolbar)))
   (:default-initargs
    :title "Redo"))

```

In this interface, pressing the toolbar button invokes the menu item callback:

```
(capi:display (make-instance 'redo))
```

This last example illustrates the use of `:selected-image`.

```
(capi:contain
  (make-instance
    'capi:toolbar
    :items
    (list
      (make-instance
        'capi:toolbar-component
        :interaction :multiple-selection
        :items
        (list (make-instance 'capi:toolbar-button
                           :image 0
                           :selected-image 1))
      )))
  )))
```

See also `item`
`make-image-locator`
`menu-item`
`toolbar`
`toolbar-component`

toolbar-component

Class

Summary A toolbar component is used to group several toolbar buttons together. Each component is separated from the surrounding components and buttons.

Toolbar components are choices, and may be used to implement toolbars on which groups of button have single-selection or multiple-selection functionality.

Package `capi`

Superclasses `toolbar-object`
`choice`

Initargs `:images` A list of images, in one-to-one correspondence with the items. Elements corresponding to `toolbar-button` items or `toolbar-component` items are ignored

- `:callbacks` A list of callback functions, in one-to-one correspondence with the items. Elements corresponding to `toolbar-button` items or `toolbar-component` items are ignored
- `:tooltips` A list of tooltip strings, in one-to-one correspondence with the items. Elements corresponding to `toolbar-button` items or `toolbar-component` items are ignored
- `:default-image-set`
An optional `image-set` object which can be used to specify images. See `toolbar-button` and `image-set` for more details.

Description

The class `toolbar-component` inherits from `choice`, and hence has a list of *items*. Its behavior is broadly similar to `button-panel`.

The *items* argument may be used to specify a mixture of `toolbar-buttons` and `toolbar-components`, or may contain arbitrary objects as items. The list may also contain CAPI panes, which will appear within the toolbar. This is typically used with `text-input-pane`, `option-pane`, and `text-input-choice`.

For items that are not toolbar buttons or toolbar components, a toolbar button is automatically created, using the appropriate elements of the *images*, *callbacks* and *tooltips* lists. If no image is specified, the item itself is used as the image. For more information on acceptable values for images, see `toolbar-button`.

See also

- `toolbar`
- `toolbar-button`

toolbar-object

Class

Summary	This is a common superclass of all toolbar objects.
Package	<code>capi</code>
Superclasses	None
Subclasses	<code>toolbar</code> <code>toolbar-button</code> <code>toolbar-component</code>
Initargs	<code>:enabled</code> If <code>t</code> , the toolbar object is enabled. <code>:enabled-function</code> A function determining the enabled state.
Accessors	<code>simple-pane-enabled</code> <code>toolbar-object-enabled-function</code>
Description	Any toolbar object may be disabled, by setting its <i>enabled</i> slot to <code>nil</code> . Disabling a toolbar or toolbar component prevents the user from interacting with any buttons contained in it. All toolbar objects may also have an <i>enabled-function</i> specified. This is called whenever <code>update-toolbar</code> is called. If it returns <code>t</code> , the toolbar object will be enabled; if it returns <code>nil</code> , the object will be disabled.
See also	<code>toolbar</code> <code>toolbar-button</code> <code>toolbar-component</code> <code>update-toolbar</code>

top-level-interface

Generic Function

Summary	Returns the top level interface containing a specified pane.
---------	--

Package	<code>capi</code>
Signature	<code>top-level-interface</code> <i>pane</i>
Description	Returns the top level interface that contains <i>pane</i> .
See also	<code>top-level-interface-p interface element</code>

top-level-interface-display-state

Function

Summary	Returns a value which indicates how the top level interface is displayed.
Package	<code>capi</code>
Signature	<code>top-level-interface-display-state</code> <i>interface</i>
Arguments	<i>interface</i> A top level interface or dialog window
Description	<p>Top level interfaces and dialogs can be manipulated by the user, such as being iconified or maximized. The program can manipulate these windows too. The function <code>top-level-interface-display-state</code> returns a value that indicates the current state of the interface <i>interface</i>. The following values can be returned:</p> <ul style="list-style-type: none"> <code>:normal</code> The window is visible and has its normal size. <code>:maximized</code> The window is visible and has been maximized. <code>:iconic</code> The window is visible as an icon. <code>:hidden</code> The window is not visible.

These values can also be passed as the `:display-state` initarg when making a top level interface.

In addition, the function (`setf top-level-interface-display-state`) can be used to change the state of a top level interface. The value can be set to one of the above, or to `:restore` if the current state is `:iconic` or `:hidden`. When set to `:restore`, the state will become `:normal` or `:maximized` depending on how the interface was visible in the past.

See also `top-level-interface-p`
`top-level-interface-geometry`
`set-top-level-interface-geometry`
`interface`

top-level-interface-geometry

Function

Summary	Returns the geometry of the top level interface.
Package	<code>capi</code>
Signature	<code>top-level-interface-geometry</code> <i>interface</i>
Description	<code>top-level-interface-geometry</code> returns the coordinates of the given interface in a form suitable for use as the <code>:best-x</code> , <code>:best-y</code> , <code>:best-width</code> and <code>:best-height</code> initargs to <code>interface</code> . The value of <i>interface</i> should be a top level interface.
Example	<pre>;; Define and display an interface. (capi:define-interface test () () (:panes (panel capi:list-panel))) (setq int (capi:display (make-instance 'test))) ;; Now manually position the interface somewhere. ;; Find where the interface is. (multiple-value-setq (tx ty twidth theight) (capi:top-level-interface-geometry int))</pre>

```
;; Now manually close the interface.

;; Create a new interface in the same place.
(setq int
  (capi:display
    (make-instance
      'test
      :best-x tx
      :best-y ty
      :best-width twidth
      :best-height theight)))
```

See also `top-level-interface-p`
`top-level-interface-display-state`
`set-top-level-interface-geometry`
`interface`

top-level-interface-geometry-key

Generic Function

Summary	Determines where the geometry of an interface is saved.	
Package	<code>capi</code>	
Signature	<code>top-level-interface-geometry-key</code> <i>interface</i> => <i>key</i> , <i>product-name</i>	
Arguments	<code>interface</code>	A top level interface.
Values	<i>key</i>	A symbol.
	<i>product-name</i>	A symbol, a string or a list of strings.
Description	Returns as multiple values a key and a product name, which determine where the geometry of <i>interface</i> is saved. The saved geometry is used when displaying a future instance.	
	The supplied method on <code>top-level-interface-geometry-key</code> returns the class name of <i>interface</i> as the <i>key</i> , and <code>nil</code> as the <i>product-name</i> . You can define methods for your interfaces and products.	

key must be a symbol.

product-name is used to derive the *product-registry-path*.

product-name can be a symbol which was previously defined to have a registry path by

```
(setf sys:product-registry-path).
```

product-name can alternatively be a string, which is taken directly as *product-registry-path*.

product-name can alternatively be a list of strings, denoting multiple path components. These are concatenated together with the appropriate separator for the platform to give *product-registry-path*.

The geometry of *interface* is saved at the path which is constructed by concatenating (with appropriate separators) these values:

```
user-path product-registry-path "Environment" (symbol-package key) (symbol-name key)
```

where *user-path* is the registry branch

HKEY_CURRENT_USER on Microsoft Windows and the home directory on Unix/Linux and Mac OS X.

Note: for your interface classes for which you want the geometry to be saved, define a method on `top-level-interface-save-geometry-p`.

Note: in an image delivered at delivery level 5, symbol names are removed by default. This breaks the saved geometry mechanism as the registry path is constructed using `symbol-name`. To make this work in a level 5 delivered image, explicitly keep the *key* symbol. See the *LispWorks Delivery User Guide* for details.

See also

`top-level-interface-save-geometry-p`

top-level-interface-p*Generic Function*

Package	<code>capi</code>
Signature	<code>top-level-interface-p <i>pane</i></code>
Description	Returns non- <code>nil</code> if <i>pane</i> is a top level interface.
See also	<code>top-level-interface</code> <code>top-level-interface-geometry</code> <code>top-level-interface-display-state</code> <code>interface</code> <code>element</code>

top-level-interface-save-geometry-p*Generic Function*

Package	<code>capi</code>
Signature	<code>top-level-interface-save-geometry-p <i>interface</i> => <i>result</i></code>
Description	The generic function <code>top-level-interface-save-geometry-p</code> returns true if the geometry of <i>interface</i> should be saved for use by a future instance. The default method (on <code>interface</code>) returns <code>nil</code> .
See also	<code>top-level-interface-geometry-key</code>

tracking-pinboard-layout*Class*

Summary	A pinboard with automatic highlighting.
Package	<code>capi</code>
Superclasses	<code>pinboard-layout</code>

Description The class `tracking-pinboard-layout` provides a pinboard which tracks mouse movement by highlighting its objects as the mouse cursor moves over them.

This functionality is implemented via a `:motion` specification in the *input-model*. Therefore, you may not specify `:motion` in the *input-model* of a `tracking-pinboard-layout`. See `output-pane` for a description of *input-model*.

Example

```

(defclass my-ellipse (capi:drawn-pinboard-object)
  ((color :initarg :color
          :initform :red
          :accessor my-ellipse-color)))

(defun draw-my-ellipse
  (output-pane self x y width height)
  (let ((x-radius (floor width 2))
        (y-radius (floor height 2)))
    (gp:draw-ellipse output-pane
      (+ x x-radius) (+ y y-radius)
      x-radius y-radius
      :foreground
      (my-ellipse-color self)
      :filled t)))

(defun change-ellipse-color (pinboard x y)
  (let ((ellipse
        (capi:pinboard-object-at-position
         pinboard x y)))
    (when ellipse
      (let ((color
            (capi:prompt-for-color
             "New color"
             :color
             (my-ellipse-color ellipse)
             :owner
             (capi:convert-to-screen))))
        (when color
          (setf (my-ellipse-color ellipse) color)
          (capi:with-geometry ellipse
            (gp:invalidate-rectangle
             pinboard
             capi:%x%
             capi:%y%
             capi:%width%
             capi:%height%)))))))

(capi:contain
 (make-instance
  'capi:tracking-pinboard-layout
  :description
  (loop for i below 20
        collect
        (make-instance 'my-ellipse
                       :x (+ 5 (random 290))
                       :y (+ 5 (random 290))

```

```

:height (+ 10 (random 50))
:width (+ 10 (random 50))
:color
(apply 'color:make-rgb
(loop for i below 3
      collect (random 1.0)))
:display-callback
'draw-my-ellipse))


```

tree-view

Class

Summary A tree view is a pane that displays a hierarchical list of items. Each item may optionally have an image and a checkbox.

Package `capi`

Superclasses `choice`
`titled-object`
`simple-pane`

Initargs

- `:roots` A list of the root nodes.
- `:children-function`
Returns the children of a node.
- `:image-function`
Returns an image for a node.
- `:state-image-function`
Returns a state image for a node.
- `:image-lists`
A plist of keywords and `image-list` objects.

:leaf-node-p-function

Optional function which determines whether a node is a leaf node (that is, has no children). This is useful if it can be computed faster than the *children-function*.

:retain-expanded-nodes

Specifies if the tree view remembers whether hidden nodes were expanded.

:expandp-function

Optional function which is called to decide whether a node should be displayed in expanded form. If not specified, all nodes are displayed collapsed, so only the root nodes are visible.

:use-images Flag to specify whether items have images. Defaults to `t`.

:use-state-images

Flag to specify whether items have state images. Defaults to `nil`.

:image-width Defaults to 16.

:image-height Defaults to 16.

:state-image-width

Defaults to *image-width*.

:state-image-height

Defaults to *image-height*.

:action-callback-expand-p

A boolean. The default value is `nil`.

:right-click-extended-match

Controls the area within which selection by the mouse right button occurs. Default `t`.

:has-root-line

Controls whether the line and expanding boxes of the root nodes are drawn. Default `t`.

:checkbox-status

Controls whether the tree has checkboxes. If non-`nil`, the value should be a non-negative integer less than the length of the image-list. This integer specifies the default initial status. The default is `nil`, meaning no checkboxes.

:checkbox-next-map

Controls the change in status when the user clicks on a checkbox. Can be an array, a function or an integer. Default `#{2 2 0}`.

:checkbox-parent-function

Controls the changes in the ancestors when the status of an item is changed.

:checkbox-child-function

Controls the changes in the descendents when the status of an item is changed.

:checkbox-change-callback

A function called when the status of an item is changed interactively.

:checkbox-initial-status

Specifies the initial status of specific items.

Accessors

- `tree-view-roots`
- `tree-view-children-function`
- `tree-view-image-function`
- `tree-view-state-image-function`
- `tree-view-leaf-node-p-function`
- `tree-view-retain-expanded-nodes`
- `tree-view-expandp-function`
- `tree-view-action-callback-expand-p`
- `tree-view-right-click-extended-match`
- `tree-view-has-root-line`
- `tree-view-checkbox-next-map`
- `tree-view-checkbox-parent-function`
- `tree-view-checkbox-status`
- `tree-view-checkbox-child-function`
- `tree-view-checkbox-change-callback`
- `tree-view-checkbox-initial-status`

Readers

- `tree-view-checkbox-status`

Description

The tree view pane allows the user to select between items displayed in a hierarchical list. Although it is a choice, only single selection interaction is supported.

Initially, only the items specified by the *roots* argument are displayed (unless an *expandp-function* is used, in which case further items may also be displayed).

Any item which has children has a small expansion button next to it to indicate that it can be expanded. When the user clicks on this button, the children nodes (as determined by the children function) are displayed.

If *action-callback-expand-p* is true, then the activate gesture expands a collapsed node, and collapses an expanded node. This expansion and contraction of the node is additional to any supplied *action-callback*.

The *image-function* is called on an item to return an image associated with the item. It can return one of the following:

A pathname or string

This specifies the filename of a file suitable for loading with `load-image`. Currently this must be a bitmap file.

A symbol

The symbol must have been previously registered by means of a call to `register-image-translation`.

An image object, as returned by `load-image`.

An image locator object

This allowing a single bitmap to be created which contains several button images side by side. See `make-image-locator` for more information. On Microsoft Windows, it also allows access to bitmaps stored as resources in a DLL.

An integer

This is a zero-based index into the tree-view's image list. This is generally only useful if the image list is created explicitly. See `image-list` for more details.

The *state-image-function* is called on an item to determine the state image: an additional optional image used to indicate the state of an item. It can return one of the above, or `nil` to indicate that there is no state image. See also *checkbox-status*, which overrides the *state-image-function*.

If *image-lists* is specified, it should be a plist containing the following keywords as keys. The corresponding values should be `image-list` objects.

`:normal`

Specifies an `image-list` object that contains the item images. The *image-function* should return a numeric index into this `image-list`.

`:state` Specifies an `image-list` object that contains the state images. The `state-image-function` should return a numeric index into this `image-list`.

If `right-click-extended-match` is `nil`, the mouse right button gesture within the tree view selects an item only when the cursor is on the item. Otherwise, this gesture also selects an item to the left or right of the cursor. The default for `right-click-extended-match` is `t`.

If `has-root-line` is `nil`, the vertical root line and expanding boxes of the root nodes are not drawn. This is useful in two cases:

- When the tree view needs to be neater. Note that the user does not have a mouse gesture to expand the root node. Normally the programmer would compensate for this by making some other gesture call `(setf tree-view-expanded-p)`.
- If a `children-function` is not supplied, this can be used to create a pane like a list view with checkboxes (see below for details of checkboxes). This pane can be handled as if it is a typical choice, except that setting the items is done by `(setf tree-view-roots)` or by passing `:roots` to `make-instance`. In a typical choice, you would do `(setf collection-items)` or pass `:items` to `make-instance`.

The default for `has-root-line` is `t`.

If the `checkbox-status` is non-`nil` then the tree view provides an automatic way of using the state images as checkboxes. The `state-image` is defaulted to a set of images containing checkboxes and the `state-image-function` is ignored, but each *item* has a status that is a non-negative integer no greater than the number of images in `state-image-list`. The status specifies which image is displayed alongside *item*.

When *item* is expanded in the tree for the first time, the status of each child is set to *item*'s status. The status can be changed interactively by the user:

- Left mouse button on a checkbox changes its status
- Space changes the status of all selected items.

The status can also be read and set programmatically (see `tree-view-item-checkbox-status`).

When the status of an item changes:

- The statuses of its ancestors may change if a *checkbox-parent-function* was supplied.
- The statuses of an items descendents may change if a *checkbox-child-function* was supplied.
- A callback given by *checkbox-callback-function* will be called, if this was supplied.

By default checkboxes have three statuses indicated by images: un-checked(0), grey-checked(1) and checked(2). If an item is checked or un-checked, then all its descendents have the same status. If an item is grey-checked, then the its descendents have various statuses. When the status of an item changes, all the descendents of that item change to to the same status, and all its ancestors change to grey-checked.

For non-default status-changing behavior, specify *checkbox-next-map*. The value can be

- An array of statuses. When the user clicks on *item*'s checkbox, the status of *item* is used to index into *checkbox-next-map*, and the status at that index becomes the new status of *item*. For example, with the default *checkbox-next-map*, checked(0) changes to un-checked(2), grey-checked(1) changes to un-checked(2), and un-checked(2) changes to checked(0).

- A function of two arguments. The first argument is a list of items and the second argument is their current status (and if the items have various statuses, the most common is used). *checkbox-next-map* should return the new status to use.
- An integer: the status is increased by 1, until this integer is reached, at which point the status becomes 0 again.

When the status of an item is changed, the statuses of items above and below it in the tree may also be changed: the system recurses up and down the tree using *checkbox-parent-function* and *checkbox-child-function* respectively.

To recurse upwards, *checkbox-parent-function* is called on the parent with five arguments: the parent, the parent's status, the item, the item's status and an flag which is non-`nil` if all the items at the same level as the item now have the same status:

```
checkbox-parent-function parent parent-status item item-status  
all-items-same-p => new-parent-status, recurse-up, recurse-down
```

If *new-parent-status* differs from *parent-status*, then the status of *parent* is set to *new-parent-status*. If *recurse-up* is non-`nil`, then the system recurses up from *parent*, and if *recurse-down* is non-`nil`, the system recurses down. The default *checkbox-parent-function* returns (`values new-item-status t nil`) where *new-item-status* is *item-status* if *all-items-same-p* is non-`nil` and `1` otherwise.

To recurse downwards, *checkbox-child-function* is called on each child with four arguments and the results are used similarly to those of *checkbox-parent-function*:

```
checkbox-child-function child child-status item item-status =>  
new-child-status, recurse-up, recurse-down
```

The default *checkbox-child-function* returns (`values parent-status nil t`).

Note: if an item has never been expanded, then it has no children. If an item has been collapsed, then it has children even though they are not currently visible.

checkbox-parent-function and *checkbox-child-function* should not modify the tree in any way.

checkbox-change-callback takes three arguments: the tree, a list of items and their new status:

```
checkbox-change-callback tree items new-status
```

This is called after the new statuses of *items* and their ancestors and descendents have been resolved.

checkbox-initial-status is used the first time that each specified item, which can be anywhere in the tree, appears. The value is a list of conses of items and their initial statuses, for example `((item1. 2) (item2. 0))`. When *item* is displayed, its status is set from this list or, if item is not specified, from *checkbox-status*. Items are removed from the list when they are displayed and setting the list does not affect the checkbox status of items that have already been displayed.

The default value of *vertical-scroll* in a *tree-view* is `t`.

Note: Since the items of a tree view are not computed until display time, the `choice` `initarg :selected-item` has no effect. See the examples in `interface-display` for a way to set the selected item in a tree view.

See also

```
choice  
tree-view-ensure-visible  
tree-view-expanded-p  
tree-view-item-checkbox-status  
tree-view-item-children-checkbox-status  
tree-view-update-item
```

tree-view-ensure-visible*Function*

Summary	Ensures that an item in a <code>tree-view</code> is visible.	
Package	<code>capi</code>	
Signature	<code>tree-view-ensure-visible tree-view item</code>	
Arguments	<code>tree-view</code>	A tree view.
	<code>item</code>	A displayed item of <code>tree-view</code> .
Description	The function <code>tree-view-ensure-visible</code> ensures that an item in a tree view is visible, scrolling the tree view if necessary.	
	Note that <code>item</code> must be an item that is displayed in <code>tree-view</code> .	
See also	<code>tree-view</code>	

tree-view-expanded-p*Generic Function*

Summary	Gets and sets the expanded state of an item in a <code>tree-view</code> .	
Package	<code>capi</code>	
Signature	<code>tree-view-expanded-p tree-view item</code>	
Signature	<code>(setf tree-view-expanded-p) on tree-view item</code>	
Arguments	<code>tree-view</code>	A <code>tree-view</code> .
	<code>item</code>	An item.
	<code>on</code>	A boolean.
Description	The generic function <code>tree-view-expanded-p</code> is the predicate for whether <code>item</code> is expanded in <code>tree-view</code> . If item is not in <code>tree-view</code> , the function returns <code>nil</code> .	

`(setf tree-view-expanded-p)` sets the expanded state of *item* in *tree-view* to *on*. If *item* is not in *tree-view*, the function does nothing.

See also `tree-view`

tree-view-item-checkbox-status

Function

Summary Gets and sets the checkbox status of an item in a `tree-view`.

Package `capi`

Signature `tree-view-item-checkbox-status tree-view item => status`

Signature `(setf tree-view-item-checkbox-status) status tree-view item`

Arguments

<i>tree-view</i>	A tree view.
<i>item</i>	An item.
<i>status</i>	A non-negative integer.

Description The function `tree-view-item-checkbox-status` retrieves the checkbox status of *item* in *tree-view*.

`(setf tree-view-item-checkbox-status)` sets the checkbox status of *item* in *tree-view*. The *status* must be a non-negative integer smaller than the number of images in *tree-view*'s *state-image-list*.

See also `tree-view`
`tree-view-item-children-checkbox-status`

tree-view-item-children-checkbox-status

Function

Summary Gets the checkbox statuses of a `tree-view` item's children.

Package	<code>capi</code>	
Signature	<code>tree-view-item-children-checkbox-status</code> <i>tree-view item</i> <code>=> result</code>	
Arguments	<i>tree-view</i>	A <i>tree-view</i> .
	<i>item</i>	An item.
Values	<i>result</i>	A list of conses (<i>child . status</i>) where each <i>child</i> is a child of <i>item</i> and <i>status</i> is <i>child's</i> checkbox status.
Description	The function <code>tree-view-item-children-checkbox-status</code> returns <i>item's</i> children together with their checkbox statuses. Note that, if <i>item</i> has not been expanded in <i>tree-view</i> , then it has no children and <i>result</i> will be <code>nil</code> .	
See also	<code>tree-view</code> <code>tree-view-item-checkbox-status</code>	

tree-view-update-an-item*Generic Function*

Summary	Updates an item in a <i>tree-view</i> .	
Package	<code>capi</code>	
Signature	<code>tree-view-update-an-item</code> <i>tree-view item in-parent</i>	
Description	The generic function <code>tree-view-update-an-item</code> is a synonym for <code>tree-view-update-item</code> . Note: <code>tree-view-update-an-item</code> is deprecated. Please use <code>tree-view-update-item</code> instead.	

See also `tree-view`
`tree-view-update-item`

tree-view-update-item

Generic Function

Summary Updates an item in a `tree-view`.

Package `capi`

Signature `tree-view-update-item tree-view item in-parent`

Arguments `tree-view` A `tree-view`.
`item` An item.
`in-parent` A boolean.

Description The generic function `tree-view-update-item` updates the item `item` in `tree-view`. This includes recomputing the text, images and children of `item`. This is useful when the data in `tree-view` changes, but the entire tree does not need recomputing.

When `in-parent` is non-`nil`, `tree-view-update-item` updates the children of the parent of `item`. This is useful when `item` is actually removed from `tree-view`, causing the children of its parent to be re-positioned.

See also `tree-view`

undefine-menu

Macro

Package `capi`

Signature `undefine-menu function-name &rest args`

Description This function undefines a menu created with `define-menu`.

See also `define-menu`
`menu`

unhighlight-pinboard-object

Generic Function

Summary Removes the highlighting from a `pinboard-object`.

Package `capi`

Signature `unhighlight-pinboard-object` *pinboard object* *&key redisplay*

Description This removes the highlighting from a pinboard object if necessary, and then if *redisplay* is non-`nil` it redisplay it. The default value of *redisplay* is `t`.

To highlight a pinboard object use `highlight-pinboard-object`.

See also `highlight-pinboard-object`
`pinboard-object`

uninstall-postscript-printer

Function

Summary Uninstalls a Postscript printer definition.

Package `capi`

Signature `uninstall-postscript-printer` *name*
&key if-does-not-exist deletep

Arguments *name* A string.
if-does-not-exist One of `nil` or `:error`.
deletep A boolean.

Description Uninstalls a PostScript printer definition for the given device *name*.
if-does-not-exist controls what happens if the named printer does not exist. The default value is `:error`.
deletep, if true, causes the printer to be removed for subsequent sessions as well as the current session, by deleting the file on the disk. The default value of *deletep* is `nil`.

See also `install-postscript-printer`

unmap-typeout

Function

Package `capi`

Signature `unmap-typeout collector-pane`

Description This switches the *collector-pane* out from its switchable layout, and brings back the pane that was there before `map-typeout` was called.

See also `map-typeout`
 `with-random-typeout`
 `collector-pane`

update-all-interface-titles

Function

Summary Updates interface window titles.

Package `capi`

Signature `update-all-interface-titles`

Description The function `update-all-interface-titles` can be used to update all the `interface` window titles when needed.

This is useful when `interface-extend-title` may return a new, different, value.

`update-all-interface-titles` calls `update-screen-interface-titles` on all the screens.

See also `interface-extend-title`
`update-screen-interface-titles`

update-interface-title

Generic Function

Summary Updates the title of an interface window.

Package `capi`

Signature `update-interface-title` *interface*

Arguments *interface* A CAPI *interface*.

Description The generic function `update-interface-title` updates the title of interface *interface*. This is useful when `interface-extend-title` may return a new, different, value.

You can specialize `update-interface-title` if needed.

To update all the interface titles, use `update-all-interface-titles` OR `update-screen-interface-titles`.

See also `interface-extend-title`
`update-all-interface-titles`
`update-screen-interface-titles`

update-pinboard-object

Function

Package `capi`

Signature `update-pinboard-object` *object*

Description This function checks the *object*'s constraints, and adjusts the *object*'s size as necessary. It then forces the layout to redisplay the *object* at its new size. Finally, it returns `⊔` if a resize was necessary.

See also `redraw-pinboard-object`
`pinboard-object`

update-screen-interface-titles

Function

Summary Updates interface window titles.

Package `capi`

Signature `update-screen-interface-titles screen`

Arguments `screen` A CAPI `screen`.

Description The function `update-screen-interface-titles` can be used to update the titles of all the interface windows on the screen `screen` when needed.

This is useful when `interface-extend-title` may return a new, different, value.

`update-screen-interface-titles` calls `update-interface-title` on all the relevant interfaces.

See also `interface-extend-title`
`update-interface-title`

update-screen-interfaces-hooks

Variable

Summary A list of functions that are called when a CAPI interface is created or destroyed.

Package	<code>capi</code>
Description	<p>Each function in the list <code>*update-screen-interfaces-hooks*</code> is called when an interface <i>interface</i> is created or destroyed.</p> <p>Each function takes two arguments: the screen and <i>interface</i>.</p> <p>You should not remove system functions from this variable so take care if setting its value. Only add or delete your own functions.</p>

update-toolbar

Function

Summary	Updates a toolbar object.
Package	<code>capi</code>
Signature	<code>update-toolbar self</code>
Description	<p>The <code>update-toolbar</code> function updates the toolbar object <i>self</i>. It computes the enabled function of <i>self</i> and the enabled functions of any toolbar components or toolbar buttons contained in it. Each toolbar object is enabled if the enabled function returns <code>t</code>, and is disabled if it returns <code>nil</code>.</p>
See also	<p><code>toolbar</code> <code>toolbar-button</code> <code>toolbar-component</code></p>

with-atomic-redisplay

Macro

Summary	The <code>with-atomic-redisplay</code> macro delays the updating of specified panes until all state changes have been performed.
Package	<code>capi</code>

Signature	<code>with-atomic-redisplay (&rest <i>panes</i>) &body <i>body</i></code>
Description	Most CAPI pane slot writers update the visual appearance of the pane at the point that their state changes, but it is sometimes necessary to cause all updates to the pane to be left until after they are all completed. The macro <code>with-atomic-redisplay</code> defers all visible changes to the state of each pane in <i>panes</i> until the end of the scope of the macro.
See also	<code>display</code> <code>simple-pane</code>

with-busy-interface

Macro

Summary	Displays an alternate cursor during the execution of some code.
Package	<code>capi</code>
Signature	<code>with-busy-interface (<i>pane</i> &key <i>cursor delay</i>) &body <i>body</i></code>
Description	<p>The macro <code>with-busy-interface</code> switches the cursor of the interface containing <i>pane</i> to be the busy cursor, evaluates <i>body</i>, and then restores the cursor. This is useful when a piece of code may take significant time to run, and visual feedback should be provided.</p> <p><i>cursor</i> specifies the cursor to use while <i>body</i> is running. The default value is <code>:busy</code>. For other allowed values, see <code>simple-pane</code>.</p> <p><i>delay</i> should be a provides a delay before the cursor is switched, so if <i>body</i> runs in less than <i>delay</i> seconds, then the cursor is not switched at all. This is usually more useful behavior than switching the cursor immediately. The default value of <i>delay</i> is 0.5.</p>

`with-busy-interface` must be called in the process of the interface containing *pane*.

See also `simple-pane`

with-dialog-results

Macro

Summary Displays a dialog and executes a body when the dialog is dismissed.

Package `capi`

Signature `with-dialog-results (&rest results) dialog-form &body body => :continuation, nil`

Arguments

<i>results</i>	Variables.
<i>dialog-form</i>	A function call form.
<i>body</i>	Forms.

Description The macro `with-dialog-results` is designed to evaluate the *dialog-form* in a special way to allow dialogs on Cocoa to use window-modal sheets. It is not needed unless you want to make code that is portable to Cocoa. The *dialog-form* should be a function call form that displays a dialog.

The overall effect is that the *body* forms are evaluated with the *results* variables bound to the values returned by the *dialog-form* when the dialog is dismissed.

The dynamic environment in which the body is evaluated varies between platforms:

- On Microsoft Windows and Motif, the `with-dialog-results` macro waits until the dialog has been dismissed and then evaluates the *body* forms.

- On Cocoa, the *dialog-form* creates a sheet attached to the active window and the `with-dialog-results` macro returns immediately. The *body* forms are evaluated when the user dismisses the sheet.

The *dialog-form* must be a cons with one of the following two formats:

- `(function-name . arguments)`
- `(apply function-name . arguments)`

The *function-name* is called with all the given *arguments*, plus an additional pair of arguments, `:continuation` and a continuation function created from *body*. In the first format, the additional arguments are placed after all the given *arguments*. In the second format, the additional arguments are placed just before the last of the given *arguments* (i.e. before the list of remaining argument to `apply`).

The continuation function binds the *results* variables to its arguments and evaluates the *body* forms. If there are more arguments than *results* variables, the extra arguments are discarded.

This macro is designed for use with *function-names* such as `popup-confirmer` or `prompt-for-string`, which take a `:continuation` keyword. You can define your own such functions provided that they call one of the CAPI functions, passing the received continuation argument.

Example

On Microsoft Windows and Motif, this displays a dialog, calls `record-label-in-database` when the user clicks OK and then returns. On Cocoa, this creates a sheet and returns; `record-label-in-database` will be called when the user clicks OK.

```
(with-dialog-results (new-label okp)
  (prompt-for-string "Enter a label")
  (when okp ; the user clicked in the OK button
    (record-label-in-database new-label)))
```

See also `display-dialog`
`popup-confirmer`

with-document-pages

Macro

Summary Executes a body of code repeatedly with a variable bound to the number of the page to be printed each iteration.

Package `capi`

Signature `with-document-pages page-var first-page last-page &body body`

Description The `with-document-pages` evaluates *body* repeatedly, with *page-var* bound to the number of the page to print on each iteration. It is used to by applications providing Page on Demand printing.

The *first-page* and *last-page* arguments are evaluated to yield the page numbers of the first and last pages in the document.

See also `with-page`
`with-print-job`

with-external-metafile

Macro

Summary Creates a metafile on disk using Graphics Ports operations.

Package `capi`

Signature `with-external-metafile (var &key pane bounds
format pathname) &body body => nil`

Arguments *var* A variable.

pane A graphics port, or `nil`.

<i>bounds</i>	A list of four integers. Can also be <code>nil</code> on Microsoft Windows.
<i>format</i>	One of <code>:enhanced</code> or <code>:windows</code> . Used only on Microsoft Windows.
<i>pathname</i>	A pathname or string.
<i>body</i>	Code containing Graphic Ports operations that draw to <i>var</i> .

Description The macro `with-external-metafile` creates a metafile at the location given by *pathname* containing records corresponding to the Graphics Ports operations in *body* that draw to *var*.

On Microsoft Windows the metafile is a device-independent format for storing pictures. For more information about metafiles, see the Microsoft documentation.

On Cocoa the metafile format is PDF as a single page.

`with-external-metafile` is not implemented on X11/Motif.

If *pane* is `nil`, the macro binds *var* to a graphics port object representing the metafile. If *pane* is non-`nil` then it must be an instance of `output-pane` or a subclass. In this case *var* is bound to *pane*, and *pane* is modified within the dynamic extent of `with-external-metafile` so all drawing operations draw to the metafile instead of *pane*. This can be useful when reusing existing redisplay code that is written expecting an `output-pane`. The default value of *pane* is `nil`.

If *bounds* is `nil` the metafile size will be computed from the drawing done within the body. This value is not allowed on Cocoa.

If *bounds* is non-`nil` (required on Cocoa), it should be a list of integers specifying the coordinate rectangle (*x y width height*) that the metafile contains.

On Microsoft Windows if *format* is `:enhanced`, an Enhanced-metafile is created. If *format* is `:windows`, a Windows-metafile is created. The default behavior on is to create an Enhanced-metafile.

On Cocoa the metafile format is always PDF as a single page, and the *format* argument is ignored.

pathname specifies the filename of the metafile. If its *pathname-type* is `nil`, then the file extension `"EMF"` is used for an Enhanced-metafile, or `"WMF"` for a Windows-metafile.

See also `draw-metafile`
`with-internal-metafile`

with-geometry

Macro

Summary The `with-geometry` macro is used for defining layouts and for creating new `pinboard-object` subclasses, by binding a set of variables to a pane's geometry.

Package `capi`

Signature `with-geometry pane &body body`

Description The main uses of the macro `with-geometry` are defining layouts and for creating new `pinboard-object` subclasses.

`with-geometry` binds the following variables across the forms in *body* to slots in the pane's geometry in much the same way as the Common Lisp macro `with-slots`:

<code>%x%</code>	The x position of the pane.
<code>%y%</code>	The y position of the pane.
<code>%object%</code>	The object whose geometry this is.
<code>%child%</code>	The same as <code>%object%</code> (kept for 3.1 compatibility).

`%ratio%` Ratio information.

The following variables give the external size and external constraints (see the *LispWorks CAPI User Guide* for a description of width and height constraints):

`%width%` The width in pixels of the pane.

`%height%` The height in pixels of the pane.

`%min-width%` The minimum width of the pane.

`%min-height%` The minimum height of the pane.

`%max-width%` The maximum width of the pane.

`%max-height%` The maximum height of the pane.

The following variables are also bound but apply only to classes with internal scrolling, such as `editor-pane`. They can be retrieved by `get-horizontal-scroll-parameters` and `get-vertical-scroll-parameters`. They can be set by `set-horizontal-scroll-parameters` and `set-vertical-scroll-parameters`.

`%scroll-width%`
 The extent of the horizontal scroll range.

`%scroll-height%`
 The extent of the vertical scroll range.

`%scroll-horizontal-page-size%`
 The horizontal scroll page size.

`%scroll-horizontal-slug-size%`
 The width of the scroll bar slug.

`%scroll-horizontal-step-size%`
 The horizontal scroll step size.

`%scroll-start-x%`
 The start of the horizontal scroll range.

`%scroll-start-y%`
 The start of the vertical scroll range.

`with-internal-metfile` is not implemented on X11/Motif.

`with-internal-metfile` behaves like `with-external-metfile` except that an object representing the metfile is returned, and no file is created on disk.

metfile must be freed after use, by calling `free-metfile`.

See also `draw-metfile`
`free-metfile`
`with-external-metfile`

with-output-to-printer

Macro

Summary	Binds a stream variable and prints its output.	
Package	<code>capi</code>	
Signature	<code>with-output-to-printer</code> (<i>stream</i> &key <i>printer</i> <i>tab-spacing</i> <i>interactive</i> <i>jobname</i>) &body <i>body</i> => <i>result</i>	
Arguments	<i>stream</i>	A variable.
	<i>printer</i>	A printer or <code>nil</code> .
	<i>tab-spacing</i>	An integer.
	<i>interactive</i>	A boolean.
	<i>jobname</i>	A string.
Values	<i>result</i>	The result of evaluating <i>body</i> .
Description	The macro <code>with-output-to-printer</code> binds the variable <i>stream</i> to a stream object, and prints everything is that is written to it in the code of <i>body</i> .	

If *interactive* is `t` then `print-dialog` is called to select the printer to use. If *interactive* is `nil` then *printer* is used unless it is `nil` in which case the `current-printer` is used. The default value of *interactive* is `t` and the default value of *printer* is `nil`.

The values of *jobname* and *tab-spacing* are passed to `print-text`, which is used to actually do the printing. The default value of *tab-spacing* is 8 and the default value of *jobname* is "Text".

See also `current-printer`
`print-dialog`
`print-text`

with-page

Macro

Summary Binds a variable to either `t` or `nil`, and executes a body of code to print a page only if the variable is `t`.

Package `capi`

Signature `with-page (printp) &body body`

Description The `with-page` macro binds *printp* to `t` if a page is to be printed, or `nil` if it is to be skipped. The *body* is executed once, and is expected to draw the document only if *printp* is `t`.

Each call to `with-page` contributes a new page to the document.

Note: `with-page` does not work on Cocoa.

See also `with-document-pages`
`with-page-transform`

with-page-transform

Macro

Summary	Defines a rectangular region within the coordinate space of an output pane or printer port.
Package	<code>capi</code>
Signature	<code>with-page-transform (x y width height) &body body</code>
Description	<p>The <code>with-page-transform</code> macro evaluates <i>x</i>, <i>y</i>, <i>width</i> and <i>height</i> to define a rectangular region within the coordinate space of an output pane or printer port. Within <i>body</i> the region is mapped onto the printable area of the page. If the specified rectangle does not have the same aspect ratio as the printable area of the page, then non-isotropic scaling will occur.</p> <p>Any number of calls to <code>with-page-transform</code> can occur during the printing of a page; for example, it is sometimes convenient to use a different page transform from that used to print the main body of the page when printing headers and footers.</p>
See also	<code>get-printer-metrics</code>

with-print-job

Macro

Summary	Creates a print job that prints to the specified printer.
Package	<code>capi</code>
Signature	<code>with-print-job (var &key pane jobname printer) &body body</code>

Description	<p>The <code>with-print-job</code> macro creates a print job which prints to <i>printer</i>. If <i>printer</i> is not specified, the default printer is used. The macro binds <i>var</i> to a graphics port object, and printing is performed by using graphics port operations to draw the object.</p> <p>If <i>pane</i> is specified it must be an instance of <code>output-pane</code> or a subclass. In this case <i>var</i> is bound to <i>pane</i>, and <i>pane</i> is modified within the dynamic extent of the <code>with-print-job</code> so all drawing operations draw to the printer instead of <i>pane</i>. This can be useful when implementing printing by modifying existing redisplay code that is written expecting an <code>output-pane</code>.</p> <p><i>jobname</i> is the name of the print job. The default value is <code>nil</code>, meaning that the name "Document" is used.</p>
See also	<p><code>printer-port-handle</code> <code>printer-port-supports-p</code> <code>set-printer-options</code> <code>with-document-pages</code> <code>with-page</code> <code>with-page-transform</code></p>

with-random-typeout*Macro*

Summary	Binds a stream variable to a collector pane.
Package	<code>capi</code>
Signature	<code>with-random-typeout</code> (<i>stream-variable</i> <i>pane</i>) &body <i>body</i>
Description	The macro <code>with-random-typeout</code> binds the variable <i>stream-variable</i> to a collector pane stream associated with <i>pane</i> for the scope of the macro. The collector pane is automatically

mapped and unmapped around the body. If the body exits normally, the timeout is not unmapped until the space bar is pressed or the mouse is clicked.

See also `map-timeout`
`unmap-timeout`
`collector-pane`

wrap-text

Function

Summary Wraps text for a given character width.

Package `capi`

Signature `wrap-text text width &key start end => strings`

Arguments `text` A string.
`width` A positive integer.
`start, end` Bounding index designators of `text`.

Values `strings` A list of strings.

Description The function `wrap-text` takes a string `text` and returns a list of strings, each of which is no longer than `width`. Together the strings in `strings` contain all the non-whitespace characters of `text` between `start` and `end` and are suitable for displaying this text on multiple lines of length `width`.

See also `wrap-text-for-pane`

wrap-text-for-pane

Function

Summary Wraps text for a given pane.

Package	<code>capi</code>
Signature	<code>wrap-text-for-pane pane text &key external-width visible-width font start end => strings</code>
Arguments	<p><i>text</i> A string.</p> <p><i>pane</i> A displayed CAPI pane.</p> <p><i>external-width</i> An integer or <code>nil</code>.</p> <p><i>visible-width</i> An integer or <code>nil</code>.</p> <p><i>font</i> A font object.</p> <p><i>start</i> An integer.</p> <p><i>end</i> An integer or <code>nil</code>.</p>
Values	<i>strings</i> A list of strings.
Description	<p>The function <code>wrap-text-for-pane</code> takes a string <i>text</i> and returns a list of strings. Together the strings in <i>strings</i> contain all the non-whitespace characters of <i>text</i> and are suitable for displaying this text on <i>pane</i>. That is, each string has a display width no greater than the width of <i>pane</i> when drawn using the font of <i>pane</i>. The arguments <i>start</i> and <i>end</i> are used as bounding index designators for <i>text</i> and characters outside these bounds are ignored.</p> <p>If <i>visible-width</i> is non-<code>nil</code> then text is wrapped to that width. Otherwise, if <i>external-width</i> is non-<code>nil</code> then text is wrapped as if the pane had that external width (that is, taking account of any borders in the pane). If both <i>visible-width</i> and <i>external-width</i> are <code>nil</code>, then the text is wrapped to the current visible width of the pane. The default value of both <i>visible-width</i> and <i>external-width</i> is <code>nil</code>.</p> <p>The <i>font</i> is used to perform the wrapping calculations. If it is <code>nil</code> (the default), then the <code>graphics-state-font</code> is used for panes such as <code>output-pane</code> that have a <code>graphics-state</code> and the <code>simple-pane-font</code> is used for other panes.</p>

See also `wrap-text`

x-y-adjustable-layout

Class

Summary	The class <code>x-y-adjustable-layout</code> provides functionality for positioning panes in a space larger than themselves (for example, it is used to choose whether to center them, or left justify them).	
Package	<code>capi</code>	
Superclasses	<code>layout</code>	
Subclasses	<code>simple-layout</code> <code>grid-layout</code>	
Initargs	<code>:x-adjust</code>	The adjust value for the x direction.
	<code>:y-adjust</code>	The adjust value for the y direction.
Accessors	<code>layout-x-adjust</code> <code>layout-y-adjust</code>	
Description	The values <i>x-adjust</i> and <i>y-adjust</i> of the slots are used by layouts to decide what to do when a pane is smaller than the space in which it is being laid out. Typically the values will be a keyword or a list of the form (<i>keyword n</i>) where <i>n</i> is an integer. These values of <i>adjust</i> are interpreted as by <code>pane-adjusted-position</code> . <code>:top</code> is the default for <i>y-adjust</i> and <code>:left</code> is the default for <i>x-adjust</i> .	
Example	Note: <code>column-layout</code> is a subclass of <code>x-y-adjustable-layout</code> .	

```
(setq column (capi:contain
              (make-instance
               'capi:column-layout
               :description (list
                            (make-instance
                             'capi:push-button
                             :text "Ok")
                            (make-instance
                             'capi:list-panel
                             :items '(1 2 3 4 5)
                            )
                           )))
              (capi:apply-in-pane-process
               column #'(setf capi:layout-x-adjust) :right column)
              (capi:apply-in-pane-process
               column #'(setf capi:layout-x-adjust) :center column)
```

See also `pane-adjusted-position`

2

GP Reference Entries

The following chapter provides reference entries for the functions and macros exported from the `graphics-ports` package. You can use these functions to draw graphics in CAPI output panes, which are a kind of graphics port. See the Graphics Ports chapter in the *LispWorks CAPI User Guide* for more information on graphics ports and their associated types.

analyze-external-image

Function

Summary	Gets the properties of DIB data in an external image.	
Package	<code>graphics-ports</code>	
Signature	<code>analyze-external-image external-image => width height color-table number</code>	
Arguments	<i>external-image</i>	An <code>external-image</code> .
Values	<i>width</i>	An integer.
	<i>height</i>	An integer.
	<i>color-table</i>	A color table.

number An integer.

Description The `analyze-external-image` function returns the width, height, color-table, and number of important colors for the external image *external-image*.

The image data in *external-image* must be in Device Independent Bitmap (DIB) format.

apply-rotation *Function*

Summary Modifies a transform such that a rotating of a given number of radians is performed on any points multiplied by the transform.

Package `graphics-ports`

Signature `apply-rotation transform theta`

Arguments *transform* A transform.
theta A real number.

Description The `apply-rotation` function modifies *transform* such that a rotation of *theta* radians is performed on any points multiplied by the transform. Any operations already contained in the transform occur before the new rotation.

apply-scale *Function*

Summary Modifies a transform such that a scaling occurs on any points multiplied by the transform.

Package `graphics-ports`

Signature `apply-scale transform sx sy`

Arguments	<i>transform</i>	A transform.
	<i>sx</i>	A real number.
	<i>sy</i>	A real number.
Description	The <code>apply-scale</code> function modifies <i>transform</i> such that a scaling of <i>sx</i> in <i>x</i> and <i>sy</i> in <i>y</i> is performed on any points multiplied by the transform. Any operations already contained in the transform occur before the new scaling.	

apply-translation

Function

Summary	Modifies a transform such that a translation is performed on any points multiplied by the transform.	
Package	<code>graphics-ports</code>	
Signature	<code>apply-translation transform dx dy</code>	
Arguments	<i>transform</i>	A transform.
	<i>dx</i>	A real number.
	<i>dy</i>	A real number.
Description	The <code>apply-translation</code> function modifies <i>transform</i> such that a translation of (<i>dx dy</i>) is performed on any points multiplied by the transform. Any operations already contained in the transform occur before the new translation.	

augment-font-description

Function

Summary	Returns a font description combining the attributes of a given font description with a set of font attributes.	
Package	<code>graphics-ports</code>	

Signature	<code>augment-font-description <i>fdesc</i> &rest <i>font-attribute*</i> => <i>return</i></code>	
Arguments	<code><i>fdesc</i></code>	A font description.
	<code><i>font-attribute</i></code>	A font attribute.
Values	<code><i>return</i></code>	A font description.
Description	<p>The <code>augment-font-description</code> function returns a font description that contains all the attributes of <code><i>fdesc</i></code> combined with the extra <code><i>font-attributes</i></code>. The <code>:stock</code> attribute is handled specially: it is omitted from <code><i>return</i></code>, unless it is the only attribute specified.</p> <p>If an attribute appears in both <code><i>fdesc</i></code> and a <code><i>font-attribute</i></code>, the value in the <code><i>font-attribute</i></code> is used. The contents of <code><i>fdesc</i></code> are not modified.</p>	
See also	<code>make-font-description</code>	

clear-external-image-conversions

Function

Summary	Clears external image conversions for a port.	
Package	<code>graphics-ports</code>	
Signature	<code>clear-external-image-conversions <i>external-image gp-or-null</i> &key <i>free-image all errorp</i></code>	
Arguments	<code><i>external-image</i></code>	An external image.
	<code><i>gp-or-null</i></code>	A graphics port or nil.
	<code><i>free-image</i></code>	A boolean.
	<code><i>all</i></code>	A boolean.
	<code><i>errorp</i></code>	A boolean.

Description The `clear-external-image-conversions` function clears the external image conversions for a port. If *gp-or-null* is `nil` all conversions are cleared using the `image-color-users`. If *all* is non-`nil` all conversions for all ports are cleared using *gp-or-null*. Conversions are also freed if *free-image* is non-`nil`. By default, *free-image* is `t`, *all* is `(null gp-or-null)`, and *errorp* is `t`.

clear-graphics-port

Function

Summary Draws a filled rectangle covering the entire port in the port's background color.

Package `graphics-ports`

Signature `clear-graphics-port port`

Arguments *port* A graphics port.

Description The `clear-graphics-port` function draws a filled rectangle covering the entire port in the port's background. All other graphics state parameters are ignored.

clear-graphics-port-state

Function

Summary Sets the graphics state of a port back to its default values.

Package `graphics-ports`

Signature `clear-graphics-port-state port`

Arguments *port* A graphics port.

Description The `clear-graphics-port-state` function sets the graphics state of *port* back to its default values, which are the ones it possessed immediately after creation.

clear-rectangle*Function*

Summary	Draws a rectangle in the port's background color.	
Package	<code>graphics-ports</code>	
Signature	<code>clear-rectangle port x y width height</code>	
Arguments	<i>port</i>	A graphics port.
	<i>x</i>	A real number.
	<i>y</i>	A real number.
	<i>width</i>	A real number.
	<i>height</i>	A real number.
Description	The <code>clear-rectangle</code> function draws the rectangle specified by <i>x</i> , <i>y</i> , <i>width</i> , and <i>height</i> in <i>port</i> 's background color. All other graphics state parameters are ignored.	

compress-external-image*Function*

Summary	Compresses DIB data in an external image.	
Package	<code>graphics-ports</code>	
Signature	<code>compress-external-image external-image => result</code>	
Arguments	<i>external-image</i>	An <code>external-image</code> .
Values	<i>result</i>	The difference in bytes between size of the original image and the size of the compressed version.
Description	The <code>compress-external-image</code> function converts the <i>external-image</i> data into compressed DIB format.	

The image data in *external-image* must be in Device Independent Bitmap (DIB) format.

compute-char-extents

Function

Summary	Returns the <i>x</i> coordinates of the end of each of the characters in a string if the string was printed to a graphics port.	
Package	<code>graphics-ports</code>	
Signature	<code>compute-char-extents port string &optional font => extents</code>	
Arguments	<i>port</i>	A CAPI pane.
	<i>string</i>	A string.
	<i>font</i>	A font.
Values	<i>extents</i>	An array of integers.
Description	Returns the <i>extents</i> of the characters in <i>string</i> in the font associated with <i>port</i> , or the <i>font</i> given. The extents are an array, one element per character, which gives the ending <i>x</i> coordinate of that character if the string was drawn to <i>port</i> . Note: To compute the extents of the entire string for a given port or font, use <code>port-string-width</code> or <code>get-string-extent</code> .	
See also	<code>get-string-extent</code> <code>port-string-width</code>	

convert-external-image

Function

Summary	Returns an image derived from an external image format.	
Package	<code>graphics-ports</code>	

Signature	<code>convert-external-image</code> <i>gp external-image</i> <i>&key cache force-new => image</i>	
Arguments	<i>gp</i>	A CAPI pane.
	<i>external-image</i>	An external image.
	<i>cache</i>	A boolean.
	<i>force-new</i>	A boolean.
Values	<i>image</i>	An image.
Description	<p>The <code>convert-external-image</code> function returns an image derived from <i>external-image</i>. The image is ready for drawing to the given graphics port.</p> <p>If <i>cache</i> is non-<code>nil</code> image conversions are cached in the <i>external-image</i>. The default value of <i>cache</i> is <code>nil</code>.</p> <p>If <i>force-new</i> is non-<code>nil</code> a new image is always created, and put in the cache. The default value of <i>force-new</i> is <code>nil</code>.</p>	

convert-to-font-description*Function*

Summary	Converts a font-spec to a font description.	
Package	<code>graphics-ports</code>	
Signature	<code>convert-to-font-description</code> <i>port font-spec => fdesc</i>	
Arguments	<i>port</i>	A graphics port
	<i>font-spec</i>	A font description object, font or symbol
Values	<i>fdesc</i>	A font-description
Description	<p>The function <code>convert-to-font-description</code> converts <i>font-spec</i> to a font description object <i>fdesc</i> for the graphics port <i>port</i>. If <i>font-spec</i> is a font, then its description is returned. If</p>	

font-spec is a font description object, then it is returned. If *font-spec* is a symbol naming a font alias, then `convert-to-font-description` converts this alias to a font and returns its font description. Other platform-specific values of *font-spec* are also accepted.

See also `font-description`
`make-font-description`

copy-external-image

Function

Summary Returns a copy of an external image.

Package `graphics-ports`

Signature `copy-external-image external-image
&key new-color-table => new-external-image`

Arguments *external-image* An external image.
new-color-table A color table.

Values *new-external-image*
An external image.

Description The `copy-external-image` function returns a copy of the *external-image*, optionally supplying a *new-color-table*. An error is signalled if this is a different size from the existing color-table.

copy-pixels

Function

Summary Copies a rectangular area from one port to another.

Package `graphics-ports`

Signature	<code>copy-pixels</code> <i>to-port from-port to-x to-y width height from-x from-y &rest args</i>	
Arguments	<i>to-port</i>	A graphics port.
	<i>from-port</i>	A graphics port.
	<i>to-x</i>	A real number.
	<i>to-y</i>	A real number.
	<i>width</i>	A real number.
	<i>height</i>	A real number.
	<i>from-x</i>	A real number.
	<i>from-y</i>	A real number.
Description	The <code>copy-pixels</code> function copies a rectangular area from one port to another. The <i>transform</i> , <i>mask</i> , <i>mask-x</i> and <i>mask-y</i> from the <i>to-port</i> 's graphics state are used. The (<i>to-x to-y</i>) is transformed according to <i>to-port</i> 's transform, but the image is not scaled or rotated. The <i>to-port</i> and <i>from-port</i> need not be the same depth and can be the same object. The <i>from-x</i> and <i>from-y</i> values are interpreted as pixel positions in the window coordinates of <i>from-port</i> , that is, they are not transformed by <i>from-port</i> 's transform.	

copy-transform*Function (inline)*

Summary	Returns a copy of a transform.	
Package	<code>graphics-ports</code>	
Signature	<code>copy-transform</code> <i>transform</i> => <i>result</i>	
Arguments	<i>transform</i>	A transform.
Values	<i>result</i>	A transform.

Description The `copy-transform` function returns a copy of *transform*.

create-pixmap-port

Function

Summary Creates a pixmap port and its window system representation.

Package `graphics-ports`

Signature `create-pixmap-port` *pane width height &key background collect
relative clear => pixmap-port*

Arguments *pane* A graphics port for a window.

width An integer.

height An integer.

background A color designator.

collect A boolean.

relative A boolean.

clear A list or `τ`.

Values *pixmap-port* A pixmap graphics port.

Description The `create-pixmap-port` function creates a pixmap-port and its window system representation. The *pane* argument specifies the color-user, used for color conversions, and its representation may also be used by the library to match the pixmap port properties. The value of *background* is used to initialize the `graphics-state-background`.

If *clear* is `τ`, the pixmap is cleared to its background color, otherwise the initial pixel values will be non-deterministic. If *clear* is a list of the form *(x y width height)*, only that part of the pixmap is cleared initially. The default value is `nil`.

If *relative* is non-`nil`, the pixmap graphics port collects pixel coordinates corresponding to the left, top, right, and bottom extremes of the drawing operations taking place within the body forms, and if these extend beyond the edges of the pixmap (into negative coordinates for example) the entire drawing is offset by an amount which ensures it remains within the port. It is as if the port moves its relative origin in order to accommodate the drawing. If the drawing size is greater than the screen size, then some of it is lost. The default value is `nil`.

If *collect* is non-`nil`, this causes the drawing extremes to be collected but without having the pixmap shift to accommodate the drawing, as *relative* does. The extreme values can be read using the `get-bounds` function, and `make-image-from-port`.

default-image-translation-table

Variable

Summary	The default image translation table.
Package	<code>graphics-ports</code>
Description	The <code>*default-image-translation-table*</code> variable contains the default image translation table. It is used if no image translation table is specified in calls to image translation table functions.
See also	<code>load-image</code>

define-font-alias

Function

Summary	Defines an alias for a font.
Package	<code>graphics-ports</code>

Signature	<code>define-font-alias</code> <i>keyword font</i>	
Arguments	<i>keyword</i>	A keyword.
	<i>font</i>	A font.
Description	The function <code>define-font-alias</code> defines <i>keyword</i> as an alias for <i>font</i> .	

destroy-pixmap-port

Function

Summary	Destroys a pixmap port, thereby freeing any window system resources it used.	
Package	<code>graphics-ports</code>	
Signature	<code>destroy-pixmap-port</code> <i>pixmap-port</i>	
Arguments	<i>pixmap-port</i>	A pixmap port.
Description	The <code>destroy-pixmap-port</code> function destroys a <code>pixmap-port</code> , freeing any window system resources.	

dither-color-spec

Function

Summary	Returns <code>t</code> if the color specification for a given pixel should result in a pixel that is on in a 1 bit dithered bitmap.	
Package	<code>graphics-ports</code>	
Signature	<code>dither-color-spec</code> <i>rgb-color-spec y x</i>	
Arguments	<i>rgb-color-spec</i>	An RGB specification.
	<i>y</i>	An integer.
	<i>x</i>	An integer.

Values	<i>result</i>	A boolean.
Description	The <code>dither-color-spec</code> returns <code>t</code> if <i>rgb-color-spec</i> should result in a pixel that is on in a 1-bit dithered bitmap. The current set of dithers is used in the decision.	
	Note: dithers do not affect drawing or the antialiasing that occurs when drawing in Cocoa.	
See also	<code>initialize-dithers</code> <code>make-dither</code> <code>with-dither</code>	

draw-arc*Function*

Summary	Draws an arc.	
Package	<code>graphics-ports</code>	
Signature	<code>draw-arc</code> <i>port x y width height start-angle sweep-angle &rest args &key filled</i>	
Arguments	<i>port</i>	A graphics port.
	<i>x</i>	A real number.
	<i>y</i>	A real number.
	<i>width</i>	A real number.
	<i>height</i>	A real number.
	<i>start-angle</i>	A real number.
	<i>sweep-angle</i>	A real number.
	<i>filled</i>	A boolean.
	<i>args</i>	General graphics port drawing arguments.

Description The `draw-arc` function draws an arc contained in the rectangle from $(x\ y)$ to $(x+width\ y+width)$ from *start-angle* to *start-angle+sweep-angle*. Both angles are specified in radians. Currently, arcs are parts of ellipses whose major and minor axes are parallel to the screen axes. If the port has rotation in its transform, the enclosing rectangle is modified to be the external enclosing orthogonal rectangle of the rotated rectangle. The start angle is rotated. The *transform*, *foreground*, *background*, *operation*, *thickness*, *scale-thickness*, and *mask* from the port's graphics state (see `make-graphics-state`) are all used. Additionally on Unix only, *stipple*, *pattern*, *mask-x*, *mask-y* are used. When *filled* is non-`nil`, a sector is drawn.

See also `draw-arcs`

draw-arcs

Function

Summary Draws several arcs.

Package `graphics-ports`

Signature `draw-arcs port description &rest args &key filled`

Arguments

<i>port</i>	A graphics port.
<i>description</i>	A description sequence.
<i>filled</i>	A boolean.
<i>args</i>	General graphics port drawing arguments.

Description The `draw-arcs` function draws several arcs as specified by the *description* sequence. This is usually more efficient than making several calls to `draw-arc`. The *description* argument is a sequence of values of the form $x\ y\ width\ height\ start-angle\ sweep-angle$. See `draw-arc` for more information.

See also `draw-arc`

draw-character*Function*

Summary	Draws a character in a given graphics port.	
Package	<code>graphics-ports</code>	
Signature	<code>draw-character port character x y &rest args &key block</code>	
Arguments	<i>port</i>	A graphics port.
	<i>character</i>	A character.
	<i>x</i>	A real number.
	<i>y</i>	A real number.
	<i>block</i>	A boolean.
	<i>args</i>	General graphics port drawing arguments.
Description	<p>The <code>draw-character</code> function draws the character at (<i>x y</i>) on the port. The <i>transform</i>, <i>foreground</i>, <i>background</i>, <i>operation</i>, <i>stipple</i>, <i>pattern</i>, <i>mask</i>, <i>mask-x</i>, <i>mask-y</i> and <i>font</i> from the <i>port</i>'s graphics state (see <code>make-graphics-state</code>) are all used. The (<i>x y</i>) specifies the leftmost point of the character's baseline. <i>block</i>, if non-<code>nil</code>, causes the character to be drawn in a character cell filled with the port's graphics state background.</p> <p>Note: The Graphics State slot <i>operation</i> is not supported for drawing text on Windows.</p>	

draw-circle*Function*

Summary	Draws a circle.	
Package	<code>graphics-ports</code>	
Signature	<code>draw-circle port x y radius &rest args &key filled</code>	

Arguments	<i>port</i>	A graphics port.
	<i>x</i>	A real number.
	<i>y</i>	A real number.
	<i>radius</i>	A real number.
	<i>args</i>	General graphics port drawing arguments.
	<i>filled</i>	A boolean.
Description	The <code>draw-circle</code> function draws a circle of the given radius centered on $(x\ y)$. The <i>transform</i> , <i>foreground</i> , <i>background</i> , <i>operation</i> , <i>thickness</i> , <i>scale-thickness</i> , and <i>mask</i> from the port's graphics state (see <code>make-graphics-state</code>) are all used. When <i>filled</i> is non- <code>nil</code> , the circle is filled with the foreground color.	
Examples	<pre>(gp:draw-circle port 100 100 20) (gp:draw-circle port 100 100 50 :filled t :foreground :green)</pre>	

draw-ellipse

Function

Summary	Draws an ellipse.	
Package	<code>graphics-ports</code>	
Signature	<code>draw-ellipse</code> <i>port x y x-radius y-radius</i> &rest <i>args</i> &key <i>filled</i>	
Arguments	<i>port</i>	A graphics port.
	<i>x</i>	A real number.
	<i>y</i>	A real number.
	<i>x-radius</i>	A real number.

<i>y-radius</i>	A real number.
<i>radius</i>	A real number.
<i>args</i>	General graphics port drawing arguments.
<i>filled</i>	A boolean.

Description The `draw-ellipse` function draws an ellipse of the given radii centered on $(x\ y)$. The *transform*, *foreground*, *background*, *operation*, *thickness*, *scale-thickness*, and *mask* from the port's graphics state (see `make-graphics-state`) are all used. When *filled* is non-`nil`, the ellipse is filled with the foreground color.

Examples

```
(gp:draw-ellipse port 100 100 20 40)

(gp:draw-ellipse port 100 100 50 10
 :filled t
 :foreground :green)
```

draw-image*Function*

Summary	Displays an image on a graphics port at a given position.	
Package	<code>graphics-ports</code>	
Signature	<code>draw-image port image to-x to-y &rest args &key from-x from-y to-width to-height from-width from-height global-alpha</code>	
Arguments	<i>port</i>	A graphics port.
	<i>image</i>	An image.
	<i>to-x</i>	A real number.
	<i>to-y</i>	A real number.
	<i>args</i>	General graphics port drawing arguments.
	<i>from-x</i>	A real number.
	<i>from-y</i>	A real number.

<i>to-width</i>	A real number.
<i>to-height</i>	A real number.
<i>from-width</i>	A real number.
<i>from-height</i>	A real number.
<i>global-alpha</i>	A real number in the inclusive range [0,1], or <code>nil</code> .

Description The `draw-image` function displays *image* on the port at *to-x to-y*. Graphics state translation is guaranteed to be supported. The default value of *from-x* and *from-y* is 0. The *width* and *height* arguments default to the size of the image.

Support for scaling and rotation are library dependent. Specifically, scaling is supported in the Windows and Cocoa implementations, but not on X11/Motif.

global-alpha, if non-`nil`, is a blending factor that applies to the whole image, in the Windows and Cocoa implementations, but not on X11/Motif. The value 0 means use only the target (that is, do not draw anything) and the value 1 means use only the source (that is, normal drawing). Intermediate real values mean use proportions of both the target and source. The value `nil` also means normal drawing, and this is the default value.

draw-line

Function

Summary	Draws a line between two given points.	
Package	<code>graphics-ports</code>	
Signature	<code>draw-line port from-x from-y to-x to-y &rest args</code>	
Arguments	<i>port</i>	A graphics port.
	<i>from-x</i>	A real number.

from-y A real number.
to-x A real number.
to-y A real number.
args General graphics port drawing arguments.

Description The `draw-line` function draws a line from (*from-x from-y*) to (*to-x to-y*). The graphics state parameters *transform*, *foreground*, *background*, *operation*, *thickness*, *scale-thickness*, *dashed*, *dash*, *line-end-style* and *mask* are used. Additionally on Unix only, *stipple*, *pattern*, *mask-x*, *mask-y* are used.

See also `draw-lines`

draw-lines

Function

Summary Draws several lines between pairs of two given points.

Package `graphics-ports`

Signature `draw-lines port description &rest args`

Arguments *port* A graphics port.
 description A description sequence.
 args General graphics port drawing arguments.

Description The `draw-lines` function draws several lines as specified by the *description* sequence. This is usually more efficient than making several calls to `draw-line`. The *description* argument is a sequence of values of the form *x1 y1 x2 y2*. See `draw-line` for more information.

See also `draw-line`

draw-point

Function

Summary	Draws a pixel at a given point.	
Package	<code>graphics-ports</code>	
Signature	<code>draw-point <i>port x y</i> &rest <i>args</i></code>	
Arguments	<i>port</i>	A graphics port.
	<i>x</i>	A real number.
	<i>y</i>	real number.
	<i>args</i>	General graphics port drawing arguments.
Description	The <code>draw-point</code> function draws a single-pixel point at (<i>x y</i>). The <i>transform</i> , <i>foreground</i> , <i>background</i> , <i>operation</i> and <i>mask</i> slots of the graphics state are used. Additionally on Unix only, <i>stipple</i> , <i>pattern</i> , <i>mask-x</i> , <i>mask-y</i> are used.	
See also	<code>draw-points</code> <code>set-graphics-state</code>	

draw-points

Function

Summary	Draws pixels at given points.	
Package	<code>graphics-ports</code>	
Signature	<code>draw-points <i>port description</i> &rest <i>args</i></code>	
Arguments	<i>port</i>	A graphics port.
	<i>description</i>	A description sequence.
	<i>args</i>	General graphics port drawing arguments.

Description The `draw-points` function draws several single-pixel points as specified by the *description* argument, which is a sequence of *x y* pairs. It is usually faster than several calls to `draw-point`. See `draw-point` for more information.

See also `draw-point`

draw-polygon

Function

Summary Draws a polygon.

Package `graphics-ports`

Signature `draw-polygon port points &rest args &key filled closed fill-rule`

Arguments

<i>port</i>	A graphics port.
<i>points</i>	A description sequence.
<i>filled</i>	A boolean.
<i>closed</i>	A boolean.
<i>fill-rule</i>	A keyword.
<i>args</i>	General graphics port drawing arguments.

Description The `draw-polygon` function draws a polygon using alternating *x* and *y* values in the *points* argument as the vertices. When *closed* is non-`nil` the edge from the last vertex to the first to be drawn. When *filled* is non-`nil` a filled, closed polygon is drawn; the *closed* argument is ignored if *filled* is non-`nil`. *transform*, *foreground*, *background*, *operation*, *thickness*, *scale-thickness*, *dashed*, *dash*, *line-end-style*, *line-joint-style* and *mask* from the *port*'s graphics state (see `make-graphics-state`) are all used. Additionally on Unix only, *stipple*, *pattern*, *mask-x*, *mask-y* are used. The *fill-rule* specifies how overlapping regions are filled. Possible values are `:even-odd` and `:winding`.


```
(gp:draw-polygons oo
  '((150 100 200 100 235 150 200
     200 150 200 115 150)
    (140 90 210 90 250 150
     210 210 140 210 100 150))
 :closed t)
```

See also `draw-polygon`

draw-rectangle

Function

Summary `Draws a rectangle.`

Package `graphics-ports`

Signature `draw-rectangle port x y width height &rest args &key filled`

Arguments

<i>port</i>	A graphics port.
<i>x</i>	A real number.
<i>y</i>	A real number.
<i>width</i>	A real number.
<i>height</i>	A real number.
<i>filled</i>	A boolean.
<i>args</i>	General graphics port drawing arguments.

Description The `draw-rectangle` function draws a rectangle whose corners are $(x\ y)$, $(x+width\ y)$, $(x+width\ y+height)$ and $(x\ y+height)$. The *filled* keyword, if non-`nil`, causes a filled rectangle to be drawn. While the exact results are host-specific, it is intended that a filled rectangle does not include the lines $(x = x+width)$ and $(y = y+height)$ while a non-filled rectangle does. This function works correctly if the *port*'s transform includes rotation. The graphics state parameters *transform*, *foreground*,

background, operation, thickness, scale-thickness, dashed, dash, line-joint-style and *mask* are used. Additionally on Unix only, *stipple, pattern, mask-x, mask-y* are used.

See also `draw-rectangles`

draw-rectangles

Function

Summary Draws several rectangles.

Package `graphics-ports`

Signature `draw-rectangles port description &rest args &key filled`

Arguments *port* A graphics port.
 description A description sequence.
 filled A boolean.
 args General graphics port drawing arguments.

Description The `draw-rectangles` function draws several rectangles as specified in *description* which consists of a group of values given as *x y width height*. The *filled* keyword if non-`nil` causes filled rectangles to be drawn. While the exact results are host-specific, it is intended that a filled rectangle does not include the lines ($x = x+width$) and ($y = y+height$) while a non-filled rectangle does. This function works correctly if the *port*'s transform includes rotation. The graphics state parameters *transform, foreground, background, operation, thickness, scale-thickness, dashed, dash, line-joint-style* and *mask* are used. Additionally on Unix only, *stipple, pattern, mask-x, mask-y* are used.

See also `draw-rectangle`

draw-string *Function*

Summary Draws a string with the baseline positioned at a given point.

Package `graphics-ports`

Signature `draw-string port string x y &rest args &key start end block`

Arguments

<i>port</i>	A graphics port.
<i>string</i>	A string.
<i>x</i>	A real number.
<i>y</i>	A real number.
<i>start</i>	A real number.
<i>end</i>	A real number.
<i>block</i>	A boolean.
<i>args</i>	General graphics port drawing arguments.

Description Draws the string with the baseline starting at (*x y*). The *transform*, *foreground*, *background*, *operation*, *stipple*, *pattern*, *mask*, *mask-x*, *mask-y* and *font* from the port's graphics state (see `make-graphics-state`) are all used. *start* and *end* specify which elements of the *string* to draw. *block*, if non-`nil`, causes each character to be drawn in a character cell filled with the port's graphics state *background*.

By default, *start* is 0.

Note: The Graphics State slot *operation* is not supported for drawing text on Windows.

ensure-gdiplus *Function*

Summary Ensures GDI+ is present and running, or shuts it down.
 Needed only when writing FLI graphics code on Windows.

Package	<code>graphics-ports</code>	
Signature	<code>ensure-gdiplus &key <i>event-func</i> <i>force</i> <i>shutdown</i> => <i>result</i></code>	
Arguments	<i>event-func</i>	A function, or <code>nil</code> .
	<i>force</i>	A boolean.
	<i>shutdown</i>	A boolean.
Values	<i>result</i>	A boolean.
Description	<p>The function <code>ensure-gdiplus</code> checks that the GDI+ module <code>gdiplus.dll</code> is loaded and that <code>GdiplusStartup</code> has been called, or shuts down GDI+.</p> <p>Most users will not need to call <code>ensure-gdiplus</code>. This is because when LispWorks itself uses GDI+, for instance via <code>read-external-image</code>, it calls <code>ensure-gdiplus</code> automatically, and never shuts GDI+ down.</p> <p>However, if your code uses GDI+ directly (by calling it through the Foreign Language Interface), then you should call <code>ensure-gdiplus</code> instead of using <code>GdiplusStartup</code> directly. Then, LispWorks will know that GDI+ has already started. This is the only circumstance in which you need to call <code>ensure-gdiplus</code>.</p> <p>Note: <code>ensure-gdiplus</code> is implemented only in LispWorks for Windows.</p> <p>If <i>shutdown</i> is <code>nil</code>, <code>ensure-gdiplus</code> ensures GDI+ is started, by the following steps:</p> <ol style="list-style-type: none"> 1. Load the GDI+ module <code>gdiplus.dll</code>, if it is not already loaded. 2. If <ol style="list-style-type: none"> a) GDI+ was already started by a previous call to <code>ensure-gdiplus</code>, and 	

- b) *force* is `nil`, and
- c) *event-func* was either not passed or is `eq` to the value that was passed for point a)

then `ensure-gdiplus` simply returns `nil`.

3. If GDI+ was already started, shut it down.
4. Start GDI+, and return the result of `GdiplusStartup`. This is 0 for success. For the meaning of other values, see the documentation of `gpstatus` in the MSDN.

If *shutdown* is true, then if GDI+ was started `ensure-gdiplus` shuts it down, and returns `t`, otherwise `ensure-gdiplus` returns `nil`. The default value of *shutdown* is `nil`.

The default value of both *event-func* and *force* is `nil`.

See also `read-external-image`

external-image

Class

Summary A class representing a color image.

Package `graphics-ports`

Description The class `external-image` provides a representation of a color image that is subject to `write-external-image`, `read-external-image` and `convert-external-image` operations.

See also `convert-external-image`
`read-external-image`
`write-external-image`

external-image-color-table

Function

Summary Returns a vector containing RGB color specifications of an external image.

Package	<code>graphics-ports</code>
Signature	<code>external-image-color-table</code> <i>external-image</i> => <i>color-table</i>
Arguments	<i>external-image</i> An external image.
Values	<i>color-table</i> A color table.
Description	The <code>external-image-color-table</code> function returns a vector containing RGB color specifications representing the color table as specified in the external image. If the result is <code>nil</code> , the external image is a 24-bit DIB, with the colors defined in each pixel instead of through a table.

external-image-color-table

Setf Expander

Summary	Replaces the color table in an external image.
Package	<code>graphics-ports</code>
Signature	<code>(setf external-image-color-table)</code> <i>replacement-color-table</i> <i>external-image</i>
Arguments	<i>external-image</i> An external image. <i>replacement-color-table</i> A color table.
Description	<code>(setf external-image-color-table)</code> replaces the color table in <i>external-image</i> . The color table specified by <i>replacement-color-table</i> must be the same length as the external image's original color table. It is a vector of RGB color-specifications.

externalize-image	<i>Function</i>
Summary	Returns an external image containing color information from an image.
Package	<code>graphics-ports</code>
Signature	<code>externalize-image gp image &key maximum-colors important-colors &allow-other-keys => external-image</code>
Arguments	<p><i>gp</i> A CAPI pane.</p> <p><i>image</i> An image.</p> <p><i>maximum-colors</i> An integer or <code>nil</code>. The default is <code>nil</code>.</p> <p><i>important-colors</i> An integer or <code>nil</code></p>
Values	<i>external-image</i> An external image.
Description	<p>The <code>externalize-image</code> function returns an <code>external-image</code> containing color information from <i>image</i>.</p> <p>If <i>maximum-colors</i> is <code>nil</code> or if the screen has no palette, an <code>external-image</code> using all the colors in <i>image</i> is created.</p> <p>If <i>maximum-colors</i> is an integer, the <code>external-image</code> containing image will be created using no more than that number of colors. If the image contains more than <i>maximum-colors</i> colors, the <i>maximum-colors</i> most frequently used colors will be accurately stored; the remainder will be approximated by nearest colors out of the accurate ones, using internal Color System parameters as the weighting factors for the color distance.</p> <p>The value of <i>important-color</i> is recorded in the <i>external-image</i> for later use, and specifies the number of colors required to draw a good likeness of the image. The default value is the number of colors in the image.</p>

See also `make-image-from-port`
`write-external-image`

find-best-font

Function

Summary Returns the best font for a CAPI pane.

Package `graphics-ports`

Signature `find-best-font pane fdesc => font`

Arguments *pane* A graphic port.
fdesc A font description.

Values *font* A font.

Description The `find-best-font` function returns the best font for *pane* which matches *fdesc*. When there alternative fonts available the choice of best font is operating system dependent.
When *fdesc* contains the attribute `:stock` with value `:system-font` or `:system-fixed-font`, the lookup will always find a stock font.

See also `find-matching-fonts`
`make-font-description`
`prompt-for-font`

find-matching-fonts

Function

Summary Returns a list of the font objects available for a pane.

Package `graphics-ports`

Signature `find-matching-fonts pane fdesc => fonts`

Arguments	<i>pane</i>	A CAPI pane.
	<i>fdesc</i>	A font description.
Values	<i>fonts</i>	A list of fonts.
Description	<p>The <code>find-matching-fonts</code> function returns a list of the font objects available for <i>pane</i> which match the attributes in <i>fdesc</i>. <code>nil</code> is returned if none match.</p> <p>When <i>fdesc</i> contains the attribute <code>:stock</code> with value <code>:system-font</code> or <code>:system-fixed-font</code>, the lookup will always find a stock font.</p>	
See also	<code>find-best-font</code> <code>list-all-font-names</code> <code>make-font-description</code>	

font-description

Function

Summary	Returns a font description object for a given font.	
Package	<code>graphics-ports</code>	
Signature	<code>font-description font => fdesc</code>	
Arguments	<i>font</i>	A font.
Values	<i>fdesc</i>	A font description.
Description	<p>The <code>font-description</code> function returns a font description object for <i>font</i>. Using this font description in a later call to <code>find-matching-fonts</code> or <code>find-best-font</code> on the original pane is expected to return a similar font.</p>	
See also	<code>convert-to-font-description</code> <code>make-font-description</code>	

font-description-attributes

Function

Summary Returns the attributes of a given font description.

Package `graphics-ports`

Signature `font-description-attributes fdesc => font-attributes`

Arguments *fdesc* A font description.

Values *font-attributes* A list of font attributes.

Description The `font-description-attributes` function returns the attributes of the *fdesc*. The list should not be destructively modified.

font-description-attribute-value

Function

Summary Returns the values of a given font attribute in a font description.

Package `graphics-ports`

Signature `font-description-attribute-value fdesc font-attribute => value`

Arguments *fdesc* A font description.

font-attribute A font attribute.

Values *value* A font attribute value.

Description The `font-description-attribute-value` function returns the value of *font-attribute* in *fdesc*, or `:wild` if *font-attribute* is not specified in *fdesc*.

font-fixed-width-p*Function*

Summary	Returns τ if a specified font is of a fixed width.	
Package	<code>graphics-ports</code>	
Signature	<code>font-fixed-width-p port &optional font => bool</code>	
Arguments	<i>port</i>	A CAPI pane.
	<i>font</i>	A font.
Values	<i>bool</i>	A boolean.
Description	The <code>font-fixed-width-p</code> function returns τ if the font associated with <i>port</i> , or the optionally specified <i>font</i> , is fixed width. Note: <code>editor-pane</code> supports variable width fonts, in Lisp-Works 5.0 and later.	

free-image*Function*

Summary	Frees the library resources allocated with an image.	
Package	<code>graphics-ports</code>	
Signature	<code>free-image port image</code>	
Arguments	<i>port</i>	A CAPI pane.
	<i>image</i>	An image.
Description	The <code>free-image</code> function frees the library resources associated with <i>image</i> . This should be done when an image is no longer needed.	

free-image-access

Function

Summary	Frees an Image Access object.
Package	<code>graphics-ports</code>
Signature	<code>free-image-access image-access</code>
Arguments	<i>image-access</i> An Image Access object
Description	The function <code>free-image-access</code> discards <i>image-access</i> , which should be an Image Access object returned by <code>make-image-access</code> .
See also	<code>image-access-transfer-from-image</code> <code>image-access-transfer-to-image</code> <code>image-access-pixel</code> <code>make-image-access</code>

get-bounds

Function

Summary	Returns the four values of the currently collected drawing extremes.
Package	<code>graphics-ports</code>
Signature	<code>get-bounds pixmap-port => left, top, right, bottom</code>
Arguments	<i>pixmap-port</i> A graphics port.
Values	<i>left</i> An integer. <i>top</i> An integer. <i>right</i> An integer. <i>bottom</i> An integer.

Description The `get-bounds` functions returns the four values *left*, *top*, *right*, *bottom* of the currently collected drawing extremes. The values can be used to get an image from the port.

Example

```
(with-pixmap-graphics-port (p1 pane width height
                           :relative t)
  (with-graphics-rotation (p1 0.123)
    (draw-rectangle p1 100 100 200 120 :filled t
                   :foreground :red)
    (get-bounds p1)))
```

produces the following output:

```
72
112
285
255
```

See also `make-image-from-port`

get-character-extent

Function

Summary Returns the extent of a character in pixels.

Package `graphics-ports`

Signature `get-character-extent port character &optional font =>`
 left, top, right, bottom

Arguments *port* A CAPI pane.

character A character.

font A font.

Values *left* An integer.

top An integer.

right An integer.

bottom An integer.

Description The `get-character-extent` function returns the extent in pixels of the *character* in the font associated with *port*, or the *font* given.

get-char-ascent

Function

Summary Returns the ascent of a character in pixels.

Package `graphics-ports`

Signature `get-char-ascent port character font => ascent`

Arguments *port* A CAPI pane.
 character A character.
 font A font.

Values *ascent* An integer.

Description The `get-character-ascent` function returns the *ascent* in pixels of the *character* in the font associated with *port*, or the *font* given.

get-char-descent

Function

Summary Returns the descent of a character in pixels.

Package `graphics-ports`

Signature `get-char-descent port character font => descent`

Arguments *port* A CAPI pane.
 character A character.
 font A font.

Values	<i>descent</i>	An integer.
Description	The <code>get-char-descent</code> function returns the <i>descent</i> in pixels of the <i>character</i> in the font associated with <i>port</i> , or the <i>font</i> given.	

get-char-width

Function

Summary	Returns the width of a character in pixels.	
Package	<code>graphics-ports</code>	
Signature	<code>get-char-width port character font => width</code>	
Arguments	<i>port</i>	A CAPI pane.
	<i>character</i>	A character.
	<i>font</i>	A font.
Values	<i>width</i>	An integer.
Description	The <code>get-char-width</code> function returns the <i>width</i> in pixels of the <i>character</i> in the font associated with <i>port</i> , or the <i>font</i> given.	

get-enclosing-rectangle

Function

Summary	Returns the smallest rectangle enclosing the given points.	
Package	<code>graphics-ports</code>	
Signature	<code>get-enclosing-rectangle &rest points => left, top, right, bottom</code>	
Arguments	<i>points</i>	Real numbers.

Values	<i>left</i>	A real number.
	<i>top</i>	A real number.
	<i>right</i>	A real number.
	<i>bottom</i>	A real number.
Description	The <code>get-enclosing-rectangle</code> function returns four values, describing the rectangle which exactly encloses the input points. The <i>points</i> argument must be a (possibly empty) list of alternating <i>x</i> and <i>y</i> values. If no <i>points</i> are given the function returns the null (unspecified) rectangle, which is four <code>nils</code> .	

get-font-ascent

Function

Summary	Returns the ascent of a font.	
Package	<code>graphics-ports</code>	
Signature	<code>get-font-ascent</code> <i>port</i> &optional <i>font</i> => <i>ascent</i>	
Arguments	<i>port</i>	A CAPI pane.
	<i>font</i>	A font.
Values	<i>ascent</i>	An integer.
Description	The <code>get-font-ascent</code> function returns the <i>ascent</i> in pixels of the font associated with <i>port</i> , or the <i>font</i> given.	

get-font-average-width

Function

Summary	Returns the average width of a font in pixels.	
Package	<code>graphics-ports</code>	

Signature	<code>get-font-average-width</code> <i>port</i> &optional <i>font</i> => <i>average-width</i>	
Arguments	<i>port</i>	A CAPI pane.
	<i>font</i>	A font.
Values	<i>average-width</i>	An integer.
Description	The <code>get-font-average-width</code> function returns the <i>average-width</i> in pixels of the font associated with <i>port</i> , or the <i>font</i> given.	

get-font-descent

Function

Summary	Returns the descent in pixels of a font.	
Package	<code>graphics-ports</code>	
Signature	<code>get-font-descent</code> <i>port</i> &optional <i>font</i> => <i>descent</i>	
Arguments	<i>port</i>	A CAPI pane.
	<i>font</i>	A font.
Values	<i>descent</i>	An integer.
Description	The <code>get-font-descent</code> function returns the <i>descent</i> in pixels of the font associated with <i>port</i> , or the <i>font</i> given.	

get-font-height

Function

Summary	Returns the height of a font.	
Package	<code>graphics-ports</code>	
Signature	<code>get-font-height</code> <i>port</i> &optional <i>font</i> => <i>height</i>	

Arguments	<i>port</i>	A CAPI pane.
	<i>font</i>	A font.
Values	<i>height</i>	An integer.
Description	The <code>get-font-height</code> function returns the <i>height</i> in pixels of the font associated with <i>port</i> , or the <i>font</i> given.	

get-font-width

Function

Summary	Returns the width of a font.	
Package	<code>graphics-ports</code>	
Signature	<code>get-font-width port &optional font => width</code>	
Arguments	<i>port</i>	A graphics port.
	<i>font</i>	A font.
Values	<i>width</i>	An integer.
Description	The <code>get-font-width</code> function returns the <i>width</i> in pixels of the font associated with <i>port</i> , or the <i>font</i> given.	

get-graphics-state

Function

Summary	Returns the graphics state object for a graphics port.	
Package	<code>graphics-ports</code>	
Signature	<code>get-graphics-state port => state</code>	
Arguments	<i>port</i>	A graphics port.

Values	<i>state</i>	A graphics state object.
Description	The <code>get-graphics-state</code> function returns the graphics state object for <i>port</i> . The individual slots can be accessed using the accessor functions.	
See also	<code>make-graphics-state</code>	

get-origin*Function*

Summary	Returns the coordinate origin of a pixmap graphics port.	
Package	<code>graphics-ports</code>	
Signature	<code>get-origin <i>pixmap-port</i> => x y</code>	
Arguments	<i>pixmap-port</i>	A graphics port.
Values	<i>x</i>	An integer.
	<i>y</i>	An integer.
Description	This returns two values being the coordinate origin of the pixmap graphics port. Normally this is (0 0) but after a series of drawing function calls with <code>:relative t</code> , the drawing may have been shifted. The <code>get-origin</code> values tell you by how much. The values are <i>not</i> needed when making images from the port's drawing.	
Example	<pre>(with-pixmap-graphics-port (p1 pane width height :relative t) (with-graphics-rotation (p1 0.123) (draw-rectangle p1 0 0 200 120 :filled t :foreground :red) (get-origin p1)))</pre> <p>produces:</p>	

get-string-extent

Function

Summary	Returns the extent in pixels of a string.	
Package	<code>graphics-ports</code>	
Signature	<code>get-string-extent port string &optional font => left, top, right, bottom</code>	
Arguments	<i>port</i>	A CAPI pane.
	<i>string</i>	A string.
Values	<i>left</i>	An integer.
	<i>top</i>	An integer.
	<i>right</i>	An integer.
	<i>bottom</i>	An integer.
Description	The <code>get-string-extent</code> function returns the extent in pixels of the <i>string</i> in the font associated with <i>port</i> , or the <i>font</i> given. Note: To compute the horizontal extents of each successive character in a string for a given port or font, use <code>compute-char-extents</code> .	
See also	<code>compute-char-extents</code>	

get-transform-scale

Function

Summary	Returns the overall scaling factor of a transform.	
Package	<code>graphics-ports</code>	

Signature	<code>get-transform-scale</code> <i>transform</i> => <i>result</i>	
Arguments	<i>transform</i>	A transform.
Values	<i>result</i>	A real number.
Description	The <code>get-transform-scale</code> function returns a single number representing the overall scaling factor present in the <i>transform</i> .	

graphics-port-transform*Function*

Summary	Returns the transform object of a graphics port.	
Package	<code>graphics-ports</code>	
Signature	<code>graphics-port-transform</code> <i>port</i> => <i>transform</i>	
Arguments	<i>port</i>	A graphics port.
Values	<i>transform</i>	A transform object.
Description	The <code>graphics-port-transform</code> function returns the transform object (a six-element list) associated with <i>port</i> .	

image*Class*

Summary	An abstract image object. An image can be drawn via <code>draw-image</code> .	
Package	<code>graphics-ports</code>	
Accessors	<code>image-height</code> <code>image-width</code>	

Description	The <code>image</code> class is the abstract image object class. An image can be drawn using <code>draw-image</code> . <code>image-height</code> and <code>image-width</code> return the image size in pixels.
See also	<code>convert-external-image</code> <code>draw-image</code> <code>load-image</code> <code>make-image-from-port</code> <code>make-sub-image</code> <code>read-and-convert-external-image</code>

image-access-pixel

Function

Summary	Gets and sets the pixels in an Image Access object.	
Package	<code>graphics-ports</code>	
Signature	<code>image-access-pixel</code> <i>image-access</i> <i>x</i> <i>y</i> => <i>color-rep</i> (<code>setf image-access-pixel</code>) <i>color-rep</i> <i>image-access</i> <i>x</i> <i>y</i> => <i>color-rep</i>	
Arguments	<i>image-access</i>	An Image Access object
	<i>x</i>	An integer.
	<i>y</i>	An integer.
Values	<i>color-rep</i>	A color reference.
Description	The function <code>image-access-pixel</code> returns the pixel value at position <i>x</i> , <i>y</i> in the Image Access object <i>image-access</i> . The pixel value <i>color-rep</i> is a color representation like that returned by <code>convert-color</code> . If needed, <i>color-rep</i> can be converted to an RGB value using <code>unconvert-color</code> . <i>color-rep</i> can contain an alpha value, for images with an alpha channel.	

The function (`setf image-access-pixel`) sets the value of the pixel at position *x*, *y* in the Image Access object *image-access*.

image-access must be an Image Access object returned by `make-image-access`.

Example See the file
`examples/capi/graphics/image-access.lisp`.

See also `image-access-pixels-from-bgra`
`image-access-pixels-to-bgra`
`image-access-transfer-to-image`
`image-access-transfer-from-image`
`free-image-access`
`make-image-access`

image-access-pixels-from-bgra

Function

Summary Copies a vector of pixel values into an Image Access object.

Package `graphics-ports`

Signature `image-access-pixels-from-bgra image-access vector`

Arguments *image-access* An Image Access object.
vector A vector.

Description The function `image-access-pixels-from-bgra` copies all the pixels to the Image Access object *image-access* from the vector *vector*. *vector* should contain a sequence of integer values in the range 0-255 for blue, green, red and alpha of each pixel. This function is optimized for the case where *vector* has element type (`unsigned-byte 8`).

An error is signalled if *vector* is not of the correct length for the Image Access object, that is (`* 4 width height`) where *width* and *height* represent the size of *image-access*.

Note: `image-access-pixels-to-bgra` must be called after this function (similarly to `(setf image-access-pixel)`).

image-access must be an Image Access object returned by `make-image-access`.

Example See the file
`examples/capi/graphics/image-access-bgra.lisp`.

See also `image-access-pixel`
`image-access-pixels-to-bgra`

image-access-pixels-to-bgra

Function

Summary Copies pixel values from an Image Access object into a vector.

Package `graphics-ports`

Signature `image-access-pixels-to-bgra image-access vector`

Arguments *image-access* An Image Access object.

vector A vector.

Description The function `image-access-pixels-to-bgra` copies all the pixels in the Image Access object *image-access* into the vector *vector* as a sequence of integer values in the range 0-255 for the blue, green, red and alpha components of each pixel. This function is optimized for the case where *vector* has element type `(unsigned-byte 8)`.

An error is signalled if *vector* is not of the correct length for the Image Access object, that is ($* 4$ *width height*) where *width* and *height* represent the size of *image-access*.

Note: `image-access-pixels-from-bgra` must be called before this function (similarly to `image-access-pixel`).

image-access must be an Image Access object returned by `make-image-access`.

Example	See the file <code>examples/capi/graphics/image-access-bgra.lisp</code> .
See also	<code>image-access-pixel</code> <code>image-access-pixels-from-bgra</code>

image-access-transfer-from-image

Function

Summary	Gets the pixel values from an <i>image</i> .
Package	<code>graphics-ports</code>
Signature	<code>image-access-transfer-from-image</code> <i>image-access</i>
Arguments	<i>image-access</i> An Image Access object
Description	<p>The function <code>image-access-transfer-from-image</code> gets the pixel values from an <i>image</i> object, making them accessible via a corresponding Image Access object <i>image-access</i>.</p> <p><i>image-access</i> must be an Image Access object returned by <code>make-image-access</code>.</p> <p>Notionally <code>image-access-transfer-from-image</code> transfers the pixel data from the window system into <i>image-access</i>, though it might do nothing on platforms where the window system allows direct access to the pixel data.</p> <p>The pixel data can be accessed using <code>image-access-pixel</code>.</p>

Example See the file
 `examples/capi/graphics/image-access.lisp`.

See also `image-access-transfer-to-image`
 `image-access-pixel`
 `free-image-access`
 `make-image-access`

image-access-transfer-to-image

Function

Summary Sets the pixel values in an *image*.

Package `graphics-ports`

Signature `image-access-transfer-to-image` *image-access*

Arguments *image-access* An Image Access object

Description The function `image-access-transfer-to-image` sets the pixel values in an *image* object from the values in a corresponding Image Access object *image-access*.

image-access must be an Image Access object returned by `make-image-access`.

Notionally `mage-access-transfer-to-image` transfers the pixel data from *image-access* to the window system, though it might do nothing on platforms where the window system allows direct access to the pixel data.

Example See the file
 `examples/capi/graphics/image-access.lisp`.

See also `free-image-access`
 `image-access-transfer-from-image`
 `image-access-pixel`
 `make-image-access`

image-freed-p*Function*

Summary	Determines whether an image has been freed.	
Package	<code>graphics-ports</code>	
Signature	<code>image-freed-p image => bool</code>	
Arguments	<i>image</i>	An image object.
Values	<i>bool</i>	A boolean.
Description	The <code>image-freed-p</code> function returns <code>non-nil</code> if the image has been freed, and <code>nil</code> otherwise.	

image-loader*Function*

Summary	Returns the image load function.	
Package	<code>graphics-ports</code>	
Signature	<code>image-loader image-id &key image-translation-table => loader</code>	
Arguments	<i>image-id</i>	An image identifier.
	<i>image-translation-table</i>	An image translation table.
Values	<i>loader</i>	An image load function.
Description	The <code>image-loader</code> function returns the image load function that would be called to load the image associated with <i>image-id</i> in <i>image-translation-table</i> . If the <i>image-id</i> is not registered with a load function, the default image load function is returned. The default value of <i>image-translation-table</i> is <code>*default-image-translation-table*</code> .	

See also `register-image-load-function`
`register-image-translation`

image-translation

Function

Summary Returns the translation for an image registered in its image translation table.

Package `graphics-ports`

Signature `image-translation image-id &key image-translation-table => translation`

Arguments *image-id* An image identifier.
image-translation-table
An image translation table.

Values *translation* A translation.

Description The `image-translation` function returns the translation for *image-id* registered in *image-translation-table*. The default value of *image-translation-table* is `*default-image-translation-table*`.

See also `register-image-load-function`
`register-image-translation`

initialize-dithers

Function

Summary Initialize dither objects up to a given order.

Package `graphics-ports`

Signature `initialize-dithers &optional order`

Arguments	<i>order</i>	An integer.
Description	<p>The <code>initialize-dithers</code> function initializes dither objects up to the given <i>order</i> ($\text{size} = 2 \wedge \text{order}$). By default, order is 3.</p> <p>Note: dithers do not affect drawing or the antialiasing that occurs when drawing in Cocoa.</p>	
See also	<p><code>dither-color-spec</code> <code>make-dither</code> <code>with-dither</code></p>	

inset-rectangle

Function (inline)

Summary	Moves the corners of a rectangle inwards by a given amount.	
Package	<code>graphics-ports</code>	
Signature	<code>inset-rectangle <i>rectangle dx dy</i> &optional <i>dx-right dy-bottom</i></code>	
Arguments	<i>rectangle</i>	A list of integers.
	<i>dx</i>	An integer.
	<i>dy</i>	An integer.
	<i>dx-right</i>	An integer.
	<i>dy-bottom</i>	An integer.
Description	<p>The <code>inset-rectangle</code> function moves the <i>left</i>, <i>top</i>, <i>right</i> and <i>bottom</i> elements of <i>rectangle</i> inwards towards the center by the distances <i>dx</i>, <i>dy</i>, <i>dx-right</i> and <i>dy-bottom</i> respectively.</p> <p>By default, <i>dx-right</i> is <i>dx</i>, and <i>dy-bottom</i> is <i>dy</i>.</p>	

inside-rectangle

Function

Summary	Determines if a point lies inside a rectangle.	
Package	<code>graphics-ports</code>	
Signature	<code>inside-rectangle <i>rectangle</i> <i>x</i> <i>y</i> => <i>result</i></code>	
Arguments	<i>rectangle</i>	A list of integers.
	<i>x</i>	An integer.
	<i>y</i>	An integer.
Values	<i>result</i>	A boolean.
Description	The <code>inside-rectangle</code> function returns <code>⊤</code> if the point (<i>x</i> <i>y</i>) is inside <i>rectangle</i> . The <i>rectangle</i> is expected to be ordered; if the rectangle is specified by (<i>left</i> <i>right</i> <i>top</i> <i>bottom</i>), then <i>left</i> must be less than <i>right</i> , and <i>bottom</i> must be less than <i>top</i> . The lines <code>y = bottom</code> and <code>x = right</code> are not considered to be inside the rectangle.	

invalidate-rectangle

Function

Summary	Invalidates the rectangle associated with the object, which causes it to be redisplayed.	
Package	<code>graphics-ports</code>	
Signature	<code>invalidate-rectangle <i>object</i> &optional <i>x</i> <i>y</i> <i>width</i> <i>height</i> => <i>result</i></code>	
Arguments	<i>object</i>	An instance of a subclass of <code>graphics-ports-mixin</code> or a subclass of <code>pinboard-object</code> .
	<i>x</i>	A real number.

	<i>y</i>	A real number.
	<i>width</i>	A real number.
	<i>height</i>	A real number.
Values	<i>result</i>	A boolean.
Description	By default it invalidates the whole rectangle, but this can be limited by passing the <code>&optional</code> arguments.	
See also	<code>validate-rectangle</code>	

invert-transform*Function*

Summary	Constructs the inverse of a transform.	
Package	<code>graphics-ports</code>	
Signature	<code>invert-transform <i>transform</i> &optional <i>into</i> => <i>inverse</i></code>	
Arguments	<i>transform</i>	A transform.
	<i>into</i>	A transform or <code>nil</code> .
Values	<i>inverse</i>	A transform.
Description	This function constructs the inverse of <i>transform</i> . If <i>T</i> is <i>transform</i> and <i>T'</i> is its inverse, then $TT' = I$. If <i>into</i> is non- <code>nil</code> it is modified to contain <i>T'</i> and returned, otherwise a new transform is constructed and returned.	

list-all-font-names*Function*

Summary	Finds the names of the available fonts.	
Package	<code>graphics-ports</code>	

Signature	<code>list-all-font-names <i>pane</i> => <i>fdescs</i></code>	
Arguments	<i>pane</i>	A graphics port.
Values	<i>fdescs</i>	A list of font description objects.
Description	<p>The function <code>list-all-font-names</code> returns a list of partially-specified font description objects which contain the "name" attributes for each known font that is available for <i>pane</i>.</p> <p>On Microsoft Windows and Cocoa the "name" attributes are just the <code>:family</code> attribute.</p> <p>On X11 the "name" attributes are <code>:foundry</code> and <code>:family</code>.</p>	
See also	<code>font-description-attributes</code> <code>find-matching-fonts</code>	

load-icon-image

Function

Summary	Loads a Windows icon image, and returns the image object.	
Package	<code>graphics-ports</code>	
Signature	<code>load-icon-image <i>port id</i> &<i>key width height</i> => <i>image</i></code>	
Arguments	<i>port</i>	A graphics port or CAPI object.
	<i>id</i>	A keyword, string or pathname.
	<i>width</i>	The desired width in pixels, or <code>nil</code> .
	<i>height</i>	The desired height in pixels, or <code>nil</code> .
Values	<i>image</i>	An <code>image</code> object.

Description The `load-icon-image` function loads an icon specified by *id* which should be either a keyword describing a standard icon, or a string or a pathname naming a Windows format icon (`.ico`) file. In this case, the first icon in the file is loaded.

The following keyword values of *id* are recognized:

<code>:sample</code>	A rectangle
<code>:hand</code>	A cross in a circle
<code>:ques</code>	A question mark in a bubble
<code>:bang</code>	An exclamation mark in a triangle
<code>:note</code>	An 'I' in a bubble
<code>:winlogo</code>	The Windows logo
<code>:warning</code>	Same as <code>:bang</code>
<code>:error</code>	Same as <code>:hand</code>
<code>:information</code>	Same as <code>:note</code>

`load-icon-image` returns an `image` object which can be drawn to *port* using `draw-image` and which must be freed using `free-image` when no longer needed.

If *width* and *height* are specified, then the image is scaled accordingly. If *width* and *height* are `nil` then the dimensions are taken from the icon file. *width* defaults to `nil` and *height* defaults to *width*.

Note: `load-icon-image` is defined only in LispWorks for Windows.

See also `draw-image`
 `free-image`
 `load-image`

load-image

Function

Summary	Loads an image and returns the image object.	
Package	<code>graphics-ports</code>	
Signature	<code>load-image gp id &key cache type editable image-translation-table => image</code>	
Arguments	<i>gp</i>	A graphics port.
	<i>id</i>	An image identifier, a file, an <code>external-image</code> , or an <code>image</code> .
	<i>cache</i>	A boolean.
	<i>type</i>	A keyword, or <code>nil</code> .
	<i>editable</i>	One of the keywords <code>:with-alpha</code> and <code>:without-alpha</code> , or a boolean.
	<i>image-translation-table</i>	An image translation table.
Values	<i>image</i>	An image object.
Description	<p>The <code>load-image</code> function loads an image identified by <i>id</i> via the <i>image-translation-table</i> using the image load function registered with it. It returns an <code>image</code> object with the representation slot initialized. The <i>gp</i> argument specifies a graphics port used to identify the library. It also specifies the resource in which colors are defined and if necessary allocated for the image. If <i>id</i> is in the table but the translation is not an external image, and the image loader returns an external image as the second value, that external image replaces the translation in the table. The default value of <i>image-translation-table</i> is <code>*default-image-translation-table*</code>.</p> <p><i>id</i> can be an <code>image</code>, which is simply returned if it is a Plain Image or if <i>editable</i> is <code>nil</code>. Otherwise a new Plain Image object is returned, as described below.</p>	

id can also be a string or pathname denoting a file, and in this case the image is loaded according to *type*, as described below.

The *cache* argument controls whether the image translation is cached. See the `convert-external-image` function for more details.

type tells `load-image` that the image is in a particular graphics format. Currently the only recognised value is `:bmp`, which means the image is a Bitmap. Other values of *type* cause `load-image` to load the image according to the file type of *id*, if *id* denotes a file, as described for `read-external-image`. See the Graphics Ports chapter in the *LispWorks CAPI User Guide* for a discussion of image handling. The default value of *type* is `nil`.

editable controls whether the image *image* is a Plain Image suitable for use with the Image Access API. The values of *editable* have the following effects:

<code>nil</code>	The image is not editable.
<code>:without-alpha</code>	The image is editable, but does not have an alpha channel.
<code>t</code>	The image is editable, but does not have an alpha channel if the source of the image has an alpha channel (for example, a TIFF file with alpha channel).
<code>:with-alpha</code>	The image is editable and has an alpha channel. It will be fully opaque when loading files without an alpha channel.

Given an image *my-image*, call

```
(load-image port my-image :editable t)
```

to create an image guaranteed to work with `make-image-access`. The default value of *editable* is `nil`.

Normally the image is freed automatically, when *gp* is destroyed. However there are circumstances where you need to explicitly free an image, for example when you want it to go away before the port. If the image is not freed, a memory leak occurs.

Note: *gp* must already be created at the time `load-image` is called. If you need to delay loading the image, for example if you are computing the image dynamically, then you can call `load-image` in the *create-callback* of the port or even in its first *display-callback*.

Compatibility Note In LispWorks 4.4 there is a keyword argument `:force-plain` with the same effect as `:editable`. `:force-plain` is still accepted in LispWorks 5.1 for backwards compatibility, but you should now use `:editable` instead.

See also `convert-external-image`
`*default-image-translation-table*`
`load-icon-image`
`make-image`
`make-image-access`

make-dither

Function

Summary Makes a dither matrix of a given size.

Package `graphics-ports`

Signature `make-dither size => matrix`

Arguments `size` An integer.

Values `matrix` A dither matrix.

Description The `make-dither` function makes a dither matrix of the given *size*.

Note: dithers do not affect drawing or the antialiasing that occurs when drawing in Cocoa.

See also `dither-color-spec`
`initialize-dithers`
`with-dither`

make-font-description

Function

Summary Returns a new font description object containing given font attributes.

Package `graphics-ports`

Signature `make-font-description &rest font-attribute* => fdesc`

Arguments *font-attribute* A font attribute.

Values *fdesc* A font description object.

Description The `make-font-description` function returns a new font description object containing the given font attributes. There is no error checking of the attributes at this point.

The `:stock` attribute is handled specially: it is omitted from *fdesc*, unless it is the only attribute specified.

See also `augment-font-description`
`convert-to-font-description`
`find-best-font`
`find-matching-fonts`
`font-description`
`merge-font-descriptions`

make-graphics-state

Function

Summary	Creates a graphics state object.
Package	<code>graphics-ports</code>
Signature	<code>make-graphics-state &key <i>transform foreground background operation thickness scale-thickness dashed dash line-end-style line-joint-style mask font fill-style stipple pattern mask-x mask-y</i> => <i>state</i></code>
Values	<i>state</i> A graphics state object.
Description	The <code>make-graphics-state</code> function creates a graphics state object. Each graphics port has a graphics state associated with it, but you may want to create your own individual graphics states for use in specialized drawing operations. Graphics state objects do not consume local resources beyond dynamic memory for the structure (so you can be relaxed about creating them in some number if you really need to). Such objects are used in the <code>with-graphics-state</code> macro described below and modified using the following functions:

```
graphics-state-transform  
graphics-state-foreground  
graphics-state-background  
graphics-state-operation  
graphics-state-stipple  
graphics-state-pattern  
graphics-state-thickness  
graphics-state-scale-thickness  
graphics-state-dashed  
graphics-state-dash  
graphics-state-fill-style  
graphics-state-line-end-style  
graphics-state-line-joint-style  
graphics-state-mask  
graphics-state-mask-x  
graphics-state-mask-y  
graphics-state-font
```

These are the read and write (via `setf`) accessors for the graphics state slots. See the "Graphics state" section in the *LispWorks CAPI User Guide* for valid values for slots of the graphics state.

Note: These slots are used only on Unix: *fill-style stipple pattern mask-x mask-y*.

Note: *operation* is not supported for drawing text on Windows.

See also `set-graphics-state`

make-image *Function*

Summary `make-image` Makes a new, empty, `image` object.

Package `graphics-ports`

Signature `make-image port width height &key alpha => image`

Arguments

<i>port</i>	A graphics port.
<i>width</i>	A positive integer.
<i>height</i>	A positive integer.
<i>alpha</i>	A generalized boolean.

Values *image* An `image` object.

Description The function `make-image` makes a new blank, editable `image` object associated with *port* and of the given *width* and *height*.

On Windows and Cocoa, if *alpha* is true, then the image will have an alpha channel.

The initial pixels in *image* are undefined. *image* is editable, that is, it is suitable for use with the Image Access API. To set the pixels, see `make-image-access`.

See also `load-image`
`make-image-access`

make-image-access

Generic Function

Summary Creates an Image Access object.

Package `graphics-ports`

Signature `make-image-access port image => image-access`

Arguments *port* A graphics port.
image An `image` object.

Values *image-access* An Image Access object.

Description The generic function `make-image-access` returns an Image Access object for the given `image` image.

image can be any `image` object returned by `make-image-from-port`. An `image` object returned by `load-image` is also suitable, but only if it is a Plain Image (see below).

image-access is used when reading and writing the pixel values of the image. For an overview of using Image Access objects, see the Graphics Ports chapter in the *LispWorks CAPI User Guide*.

Note: on some platforms (currently Windows) not every `image` object is a Plain Image. If needed, forcibly create a Plain Image suitable for passing to `make-image-access` as described in `load-image`.

Note: ensure that you eventually discard *image-access*, using `free-image-access`.

Example See the file
`examples/capi/graphics/image-access.lisp`.

make-transform*Function*

Summary	Returns a new transform object initialized according to a set of optional arguments.	
Package	<code>graphics-ports</code>	
Signature	<code>make-transform &optional a b c d e f => transform</code>	
Arguments	<i>a</i>	A real number.
	<i>b</i>	A real number.
	<i>c</i>	A real number.
	<i>d</i>	A real number.
	<i>e</i>	A real number.
	<i>f</i>	A real number.
Values	<i>transform</i>	A transform.
Description	<p>The <code>make-transform</code> function returns a new transform object initialized according to the optional args. The default args make the unit transform.</p> <p>Default values are as follows: <i>a</i> and <i>d</i> are 1; <i>b</i>, <i>c</i>, <i>e</i>, and <i>f</i> are 0. The transform matrix is</p> $\begin{array}{ccc} a & b & 0 \\ c & d & 0 \\ e & f & 1 \end{array}$ <p>for generalized two dimensional points of the form (<i>x y 1</i>).</p>	

merge-font-descriptions*Function*

Summary	Returns a font description containing the attributes of two specified font descriptions.
---------	--

Package	<code>graphics-ports</code>	
Signature	<code>merge-font-descriptions <i>fdesc1</i> <i>fdesc2</i> => <i>fdesc</i></code>	
Arguments	<i>fdesc1</i>	A font description.
	<i>fdesc2</i>	A font description.
Values	<i>fdesc</i>	A font description.
Description	<p>The <code>merge-font-description</code> function returns a font description containing all the attributes of <i>fdesc1</i> and <i>fdesc2</i>. If an attribute appears in both <i>fdesc1</i> and <i>fdesc2</i>, the value in <i>fdesc1</i> is used. The <code>:stock</code> attribute is handled specially: it is omitted from <i>fdesc</i>, unless it is the only attribute in <i>fdesc1</i> and <i>fdesc2</i>.</p> <p>The contents of <i>fdesc1</i> and <i>fdesc2</i> are not modified.</p>	
See also	<code>make-font-description</code>	

offset-rectangle

Function (inline)

Summary	Offsets a rectangle by a given distance.	
Package	<code>graphics-ports</code>	
Signature	<code>offset-rectangle <i>rectangle</i> <i>dx</i> <i>dy</i></code>	
Arguments	<i>rectangle</i>	A list of integers.
	<i>dx</i>	A real number.
	<i>dy</i>	A real number.
Description	<p>The <code>offset-rectangle</code> function offsets the <i>rectangle</i> by the distance (<i>dx dy</i>).</p> <p><i>rectangle</i> is a list (<i>left top right bottom</i>).</p>	

ordered-rectangle-union*Function*

Summary	Returns the union of two rectangles.	
Package	<code>graphics-ports</code>	
Signature	<code>ordered-rectangle-union</code> <i>left-1 top-1 right-1 bottom-1</i> <i>left-2 top-2 right-2 bottom-2</i> <i>=> left, top, right, bottom</i>	
Arguments	<i>left-1</i>	A real number.
	<i>top-1</i>	A real number.
	<i>right-1</i>	A real number.
	<i>bottom-1</i>	A real number.
	<i>left-2</i>	A real number.
	<i>top-2</i>	A real number.
	<i>right-2</i>	A real number.
	<i>bottom-2</i>	A real number.
Values	<i>left</i>	A real number.
	<i>top</i>	A real number.
	<i>right</i>	A real number.
	<i>bottom</i>	A real number.
Description	The <code>ordered-rectangle-union</code> function returns four values: the <i>left</i> , <i>top</i> , <i>right</i> and <i>bottom</i> of the union of the two rectangles specified in the arguments. The caller guarantees that each input rectangle is ordered, that is, the left values must be smaller or equal to the right values, and the top values must be greater than or equal to the bottom ones.	
See also	<code>rectangle-union</code>	

pixblt

Function

Summary	Copies one area of a graphics port to another area of a different graphics port.	
Package	<code>graphics-ports</code>	
Signature	<code>pixblt to-port operation from-port to-x to-y width height from-x from-y</code>	
Arguments	<i>to-port</i>	A graphics port.
	<i>operation</i>	A graphics state operation.
	<i>from-port</i>	A graphics port.
	<i>to-x</i>	A real number.
	<i>to-y</i>	A real number.
	<i>width</i>	A real number.
	<i>height</i>	A real number.
	<i>from-x</i>	A real number.
	<i>from-y</i>	A real number.
Description	<p>The <code>pixblt</code> function copies one area of <i>from-port</i> to another area of <i>to-port</i> using the specified <i>operation</i> and <i>mask</i>. Both ports should be the same depth. The graphics port transforms are not used.</p> <p>See the "Graphics state" section in the <i>LispWorks CAPI User Guide</i> for valid values for <i>operation</i>.</p>	

pixmap-port

Class

Summary	The class of pixmap graphics port objects.
Package	<code>graphics-ports</code>

Description The `Pixmap-port` class is the class of pixmap graphics port objects which can be used for drawing operations.

See also `create-pixmap-port`
 `destroy-pixmap-port`
 `with-pixmap-graphics-port`

port-height

Function

Summary Returns the pixel height of a port.

Package `graphics-ports`

Signature `port-height port => result`

Arguments `port` A graphics port.

Values `result` An integer.

Description The `port-height` function returns the pixel height of `port`.

port-string-height

Function

Summary Returns the height of a string drawn to a given port in pixels.

Package `graphics-ports`

Signature `port-string-height port string => height`

Arguments `port` A graphics port.

`string` A string.

Values `height` An integer.

Description The `port-string-height` function returns the *height* in pixels of *string* when drawn to *port*. The font used is the one currently in the port's graphics state.

port-string-width

Function

Summary Returns the width of a string drawn to a given port in pixels.

Package `graphics-ports`

Signature `port-string-width port string => width`

Arguments

<i>port</i>	A graphics port.
<i>string</i>	A string.

Values *width* An integer.

Description The `port-string-width` function returns the *width* in pixels of *string* when drawn to *port*. The font used is the one currently in the port's graphics state.

Note: To compute the horizontal extents of each successive character in a string for a given port or font, use `compute-char-extents`.

See also `compute-char-extents`

port-width

Function

Summary Returns the pixel width of a port.

Package `graphics-ports`

Signature `port-width port => width`

Arguments	<i>port</i>	A graphics port.
Values	<i>width</i>	An integer.
Description	The <code>port-width</code> function returns the pixel width of <i>port</i> .	

postmultiply-transforms*Function*

Summary	Postmultiplies two transforms.	
Package	<code>graphics-ports</code>	
Signature	<code>postmultiply-transforms transform1 transform2</code>	
Arguments	<i>transform1</i>	A transform.
	<i>transform2</i>	A transform.
Description	The <code>postmultiply-transforms</code> function postmultiplies the partial 3 x 3 matrix represented by <i>transform1</i> by the partial 3 x 3 matrix represented by <i>transform2</i> , storing the result in <i>transform1</i> . In the result, the translation, scaling and rotation operations contained in <i>transform2</i> are effectively performed <i>after</i> those in <i>transform1</i> .	
	<code>transform1 = transform1 . transform2</code>	

premultiply-transforms*Function*

Summary	Premultiplies two transforms.	
Package	<code>graphics-ports</code>	
Signature	<code>premultiply-transforms transform1 transform2</code>	
Arguments	<i>transform1</i>	A transform.

transform2 A transform.

Description The `premultiply-transforms` function premultiplies the partial 3 x 3 matrix represented by *transform1* by the partial 3 x 3 matrix represented by *transform2*, storing the result in *transform1*. In the result, the translation, scaling and rotation operations contained in *transform2* are effectively performed *before* those in *transform1*.

```
transform1 = transform2 . transform1
```

read-and-convert-external-image

Function

Summary Returns an image converted from an external image read from a file.

Package `graphics-ports`

Signature `read-and-convert-external-image gp file
 &key transparent-color-index => image, external-image`

Arguments *gp* A CAPI pane.
file A pathname designator.
transparent-color-index
 An integer or `nil`.

Values *image* An image.
external-image An external image.

Description Returns an image converted from an external image read from *file*. The external image is returned as a second value. *transparent-color-index* is interpreted as described for `read-external-image`.

See also `convert-external-image`
`external-image`
`read-external-image`

read-external-image

Function

Summary Returns an external image read from a file.

Package `graphics-ports`

Signature `read-external-image file &key transparent-color-index type`
`=> image`

Arguments *file* A pathname designator.

transparent-color-index

An integer or `nil`.

type A keyword, or `nil`.

Values *image* An external image.

Description The `read-external-image` function returns an external image read from *file*.

transparent-color-index specifies the index of the transparent color in the color map. *transparent-color-index* works only for images with a color map, that is, those with 256 colors or less. The default value is `nil`, meaning that there is no transparent color.

type tells `read-external-image` that the image is in a particular graphics format. Currently the only recognised value is `:bmp`, which means the image is read as a Bitmap. Other values of *type* cause `read-external-image` to read the image according to the file type of *file*. "bmp" or "dib" mean that the image is read as a Bitmap. Other file types are handled in

Operating System-specific ways. See the Graphics Ports chapter in the *LispWorks CAPI User Guide* for details. The default value of *type* is `nil`.

Example To see the effect of *transparent-color-index*, edit `examples/capi/graphics/images.lisp`.

Specify a non-white `:background` for the *viewer* pane. Use an image editing tool to find the transparent color index (183 in this image) and change the call to `read-external-image` like this:

```
(gp:read-external-image file :transparent-color-index
 183)
```

Then compile and run the example, click the **Change...** button and select the `lwsplash.bmp` file.

See also `external-image`

rectangle-bind

Macro

Summary Binds four variables to the corners of a rectangle across a body of code.

Package `graphics-ports`

Signature `rectangle-bind ((a b c d) rectangle) &body body => result`

Arguments

<i>a</i>	A variable.
<i>b</i>	A variable.
<i>c</i>	A variable.
<i>d</i>	A variable.
<i>rectangles</i>	A rectangle.
<i>body</i>	A body of code.

Values	<i>result</i>	The return value of the last form in <i>body</i> .
Description	The <code>rectangle-bind</code> macro binds the variables <i>a b c d</i> to <i>left top right bottom</i> of <i>rectangle</i> for the <i>body</i> of the macro.	

rectangle-bottom*Macro*

Summary	Get and sets the <i>bottom</i> element of a rectangle.	
Package	<code>graphics-ports</code>	
Signature	<code>rectangle-bottom</code> <i>rectangle</i> => <i>bottom</i>	
Signature	<code>(setf rectangle-bottom)</code> <i>bottom</i> <i>rectangle</i> => <i>bottom</i>	
Arguments	<i>rectangle</i>	A rectangle.
Values	<i>bottom</i>	A real number.
Description	Returns and via <code>setf</code> sets the <i>bottom</i> element of <i>rectangle</i> . <i>rectangle</i> is a list of numbers (<i>left top right bottom</i>).	

rectangle-height*Macro*

Summary	Returns the <i>height</i> element of a rectangle.	
Package	<code>graphics-ports</code>	
Signature	<code>rectangle-height</code> <i>rectangle</i> => <i>height</i>	
Arguments	<i>rectangle</i>	A rectangle.
Values	<i>height</i>	A real number.

Description The `rectangle-height` macro returns the difference between the *bottom* and *top* elements of *rectangle*.
rectangle is a list of numbers (*left top right bottom*).

rectangle-left

Macro

Summary Gets and set the *left* element of a rectangle.

Package `graphics-ports`

Signature `rectangle-left rectangle => left`

Signature `(setf rectangle-left) left rectangle => left`

Arguments *rectangle* A rectangle.

Values *left* A real number.

Description The `rectangle-left` macro returns and via `setf` sets the *left* element of *rectangle*.
rectangle is a list of numbers (*left top right bottom*).

rectangle-right

Macro

Summary Gets and sets the *right* element of a rectangle.

Package `graphics-ports`

Signature `rectangle-right rectangle => right`

Signature `(setf rectangle-right) right rectangle => right`

Arguments *rectangle* A rectangle.

Values	<i>right</i>	A real number.
Description	The <code>rectangle-right</code> macro returns and via <code>setf</code> sets the <i>right</i> element of <i>rectangle</i> . <i>rectangle</i> is a list of numbers (<i>left top right bottom</i>).	

rectangle-top*Macro*

Summary	Gets and sets the <i>top</i> element of a rectangle.	
Package	<code>graphics-ports</code>	
Signature	<code>rectangle-top rectangle => top</code>	
Signature	<code>(setf rectangle-top) top rectangle => top</code>	
Arguments	<i>rectangle</i>	A rectangle.
Values	<i>top</i>	A real number.
Description	The <code>rectangle-top</code> macro returns and via <code>setf</code> sets the <i>top</i> element of <i>rectangle</i> . <i>rectangle</i> is a list of numbers (<i>left top right bottom</i>).	

rectangle-union*Function*

Summary	Returns the four values representing a union of two rectangles.	
Package	<code>graphics-ports</code>	
Signature	<code>rectangle-union left-1 top-1 right-1 bottom-1 left-2 top-2 right-2 bottom-2 => left, top, right, bottom</code>	

Arguments	<i>left-1</i>	A real number.
	<i>top-1</i>	A real number.
	<i>right-1</i>	A real number.
	<i>bottom-1</i>	A real number.
	<i>left-2</i>	A real number.
	<i>top-2</i>	A real number.
	<i>right-2</i>	A real number.
	<i>bottom-2</i>	A real number.
Values	<i>left</i>	A real number.
	<i>top</i>	A real number.
	<i>right</i>	A real number.
	<i>bottom</i>	A real number.
Description	The <code>rectangle-union</code> function returns four values: the <i>left</i> , <i>top</i> , <i>right</i> and <i>bottom</i> of the union of the two rectangles specified in the arguments. The values input for the two rectangles are ordered by this function before it uses them.	
See also	<code>ordered-rectangle-union</code>	

rectangle-width

Macro

Summary	Returns the difference between the <i>left</i> and <i>right</i> elements of a rectangle.	
Package	<code>graphics-ports</code>	
Signature	<code>rectangle-width <i>rectangle</i> => <i>width</i></code>	
Arguments	<i>rectangle</i>	A rectangle

Values	<i>width</i>	A real number
Description	The <code>rectangle-width</code> macro returns the difference between <i>right</i> and <i>left</i> elements of <i>rectangle</i> . <i>rectangle</i> is a list of numbers (<i>left top right bottom</i>).	

rect-bind *Macro*

Summary	Binds four variables to the elements of a rectangle across a body of code.	
Package	<code>graphics-ports</code>	
Signature	<code>rect-bind ((x y <i>width height</i>) <i>rectangle</i>) &body <i>body</i> => <i>result</i></code>	
Arguments	<i>x</i>	A variable.
	<i>y</i>	A variable.
	<i>width</i>	A variable.
	<i>height</i>	A variable.
	<i>rectangle</i>	A rectangle.
	<i>body</i>	A body of Lisp code.
Values	<i>result</i>	The return value of the last form in <i>body</i> .
Description	The <code>rect-bind</code> macro binds <i>x y width height</i> to the appropriate values from <i>rectangle</i> and executes the <i>body</i> forms. The <i>rectangle</i> is a list of the form (<i>left top right bottom</i>).	

register-image-load-function *Function*

Summary	Registers one or more image identifiers with an image loading function.	
---------	---	--

Package	<code>graphics-ports</code>
Signature	<code>register-image-load-function</code> <i>image-id</i> <i>image-load-function</i> &key <i>image-translation-table</i>
Arguments	<i>image-id</i> An image identifier or a list of image identifiers. <i>image-load-function</i> A function. <i>image-translation-table</i> An image translation table.
Description	The <code>register-image-load-function</code> function registers one or more <i>image-ids</i> with an <i>image-load-function</i> in the <i>image-translation-table</i> . If <i>image-load-function</i> is <code>nil</code> it causes the default loader to be used in subsequent calls to <code>load-image</code> . The <i>image-id</i> argument can be a list of identifiers or a single identifier. The default value of <i>image-translation-table</i> is <code>*default-image-translation-table*</code> .
See also	<code>*default-image-translation-table*</code> <code>load-image</code>

register-image-translation

Function

Summary	Registers an image identifier and image loading function with a translation in an image translation table.
Package	<code>graphics-ports</code>
Signature	<code>register-image-translation</code> <i>image-id</i> <i>translation</i> &key <i>image-translation-table</i> <i>image-load-fn</i>
Arguments	<i>image-id</i> An image identifier. <i>translation</i> An image translation.

image-translation-table

An image translation table.

image-load-fn An image loading function.

Description The `register-image-translation` function registers *image-id* and *image-load-fn* with the *translation* in the *image-translation-table*. When `load-image` is called with second argument *image-id*, the *image-load-fn* is called with *translation* as its second argument. If *image-load-fn* is `nil`, the image translation table's default image loader is used; this converts an external image object or file to an image. If *translation* is `nil` the identifier is deregistered. Returns the *image-id* and the *image-load-fn*. The default value of *image-translation-table* is `*default-image-translation-table*`.

See also `*default-image-translation-table*`
`load-image`
`reset-image-translation-table`

reset-image-translation-table*Function*

Summary Clears the image translation table hash tables.

Package `graphics-ports`

Signature `reset-image-translation-table &key image-translation-table`

Arguments *image-translation-table*
 An image translation table.

Description The `reset-image-translation-table` function clears the image translation table hash tables and set the default *image-load-fn* to `read-and-convert-external-image`. The default value of *image-translation-table* is `*default-image-translation-table*`.

See also `*default-image-translation-table*`
`read-and-convert-external-image`
`register-image-translation`

separation

Function

Summary Returns the distance between two points.

Package `graphics-ports`

Signature `separation x1 y1 x2 y2 => dist`

Arguments *x1* An integer.

y1 An integer.

x2 An integer.

y2 An integer.

Values *dist* A real number.

Description The `separation` function returns the distance between points (*x1 y1*) and (*x2 y2*).

set-default-image-load-function

Function

Summary Sets the default image load function of an image translation table.

Package `graphics-ports`

Signature `set-default-image-load-function image-load-function
&key image-translation-table`

Arguments *image-load-function*

An image load function.

image-translation-table

An image translation function.

Description The `set-default-image-load-function` function sets the default image load function of *image-translation-table*. The default image load function is `read-and-convert-external-image`. The default value of *image-translation-table* is `*default-image-translation-table*`.

set-graphics-port-coordinates*Function*

Summary Modifies the transform of a port such that the edges of the port correspond to the arguments given.

Package `graphics-ports`

Signature `set-graphics-port-coordinates` *port*
&key left top right bottom

Arguments

<i>port</i>	A graphics port.
<i>left</i>	A real number.
<i>top</i>	A real number.
<i>right</i>	A real number.
<i>bottom</i>	A real number.

Description The `set-graphics-port-coordinates` function modifies the transform of the *port* is permanently such that the edges of the port correspond to the values of the arguments.

Example The following code

```
(set-graphics-port-coordinates port :left -1.0
                                   :top 1.0
                                   :right 1.0
                                   :bottom -1.0)
```

changes the coordinates of the port so that the point (0 0) is in the exact center of the port and the edges are a unit distance away, with a right-handed coordinate system.

By default, *left* and *top* are 1.

set-graphics-state

Function

Summary Directly alters the graphics state of a graphics port according to the keyword arguments supplied.

Package `graphics-ports`

Signature `set-graphics-state` *port* &rest *args*
&key transform foreground background
operation stipple pattern fill-style thickness
scale-thickness dashed dash line-end-style
line-joint-style mask mask-x mask-y font

Arguments *port* A graphics port.

Description This directly alters the graphics state of *port* according to the values of the keyword arguments *args*. Unspecified keywords leave the associated slots unchanged.

See the "Graphics state" section in the *LispWorks CAPI User Guide* for valid values for *args*.

See also `with-graphics-state`

transform

Type

Summary The transform type, defined for transform objects.

Package `graphics-ports`

Description The `transform` type is the type defined for transform objects, which are six-element lists of numbers.

transform-area*Function*

Summary Transforms a set of points and returns the resulting rectangle.

Package `graphics-ports`

Signature `transform-area transform x y width height => rectangle`

Arguments *transform* A transform.
 x A real number.
 y A real number.
 width A real number.
 height A real number.

Values *rectangle* A rectangle.

Description The `transform-area` function transforms the points $(x\ y)$ and $(x+width\ y+height)$ and returns the transformed rectangle as $(x\ y\ width\ height)$ values.

transform-distance*Function*

Summary Transforms a distance vector by the rotation and scale of a transform.

Package `graphics-ports`

Signature `transform-distance transform dx dy => dx2, dy2`

Arguments *transform* A transform.

	<i>dx</i>	A real number.
	<i>dy</i>	A real number.
Values	<i>dx2</i>	A real number.
	<i>dy2</i>	A real number.
Description	The <code>transform-distance</code> function transforms the distance (<i>dx dy</i>) by the rotation and scale in the <i>transform</i> . The translation in the transform is ignored. Transformed (<i>dx dy</i>) is returned as two values.	

transform-distances

Function

Summary	Transforms a list of alternating distance vectors by a given transform.	
Package	<code>graphics-ports</code>	
Signature	<code>transform-distances transform distances => result</code>	
Arguments	<i>transform</i>	A transform.
	<i>distances</i>	A list of pairs of real numbers.
Values	<i>result</i>	A list of pairs of real numbers.
Description	The <code>transform-distances</code> function transforms a list of alternating (<i>dx dy</i>) pairs in <i>distances</i> by the <i>transform</i> . Transformed values are returned as a new list.	

transform-is-rotated

Function

Summary	Returns <code>⊤</code> if a given transform contains a rotation.	
Package	<code>graphics-ports</code>	

Signature	<code>transform-is-rotated transform => bool</code>	
Arguments	<code>transform</code>	A transform.
Values	<code>bool</code>	A boolean.
Description	The <code>transform-is-rotated</code> function returns <code>⊤</code> if <code>transform</code> contains any rotation.	

transform-point*Function*

Summary	Transforms a point by multiplying it by a transform.	
Package	<code>graphics-ports</code>	
Signature	<code>transform-point transform x y => xnew ynew</code>	
Arguments	<code>transform</code>	A transform.
	<code>x</code>	A real number.
	<code>y</code>	A real number.
Values	<code>xnew</code>	A real number.
	<code>ynew</code>	A real number.
Description	The <code>transform-point</code> function transforms the point $(x\ y)$ by multiplying it by <code>transform</code> . The transformed $(x\ y)$ is returned as two values.	

transform-points*Function*

Summary	Transforms a list of points by a transform.	
Package	<code>graphics-ports</code>	

Signature	<code>transform-points</code> <i>transform points</i> &optional <i>into</i> => <i>result</i>	
Arguments	<i>transform</i>	A transform.
	<i>points</i>	A list of pairs of real numbers.
	<i>into</i>	A list.
Values	<i>result</i>	A list of pairs of real numbers.
Description	The <code>transform-points</code> function transforms a list of alternating (<i>x y</i>) pairs in <i>points</i> by the <i>transform</i> . If <i>into</i> is supplied it is modified to contain the result and must be a list the same length as <i>points</i> . If <i>into</i> is not supplied, a new list is returned.	

transform-rect

Function

Summary	Returns the transform of two points representing the top-left and bottom-right of a rectangle.	
Package	<code>graphics-ports</code>	
Signature	<code>transform-rect</code> <i>transform left top right bottom</i> => <i>left2 top2 right2 bottom2</i>	
Arguments	<i>transform</i>	A transform.
	<i>left</i>	A real number.
	<i>top</i>	A real number.
	<i>right</i>	A real number.
	<i>bottom</i>	A real number.
Values	<i>left2</i>	A real number.
	<i>top2</i>	A real number.
	<i>right2</i>	A real number.

bottom2 A real number.

Description The `transform-rect` function transforms the rectangle represented by the two points (*left top*) and (*right bottom*) by *transform*.

undefine-font-alias

Function

Summary Removes a font alias.

Package `graphics-ports`

Signature `undefine-font-alias keyword`

Arguments *keyword* A keyword.

Description The `undefine-font-alias` function removes the font alias named by *keyword*.

union-rectangle

Macro

Summary Modifies a rectangle to be a union of itself and another rectangle.

Package `graphics-ports`

Signature `union-rectangle rectangle left top right bottom => rectangle`

Arguments *rectangle* A rectangle.
left A real number.
right A real number.
top A real number.
bottom A real number.

Values	<i>rectangle</i>	A rectangle.
Description	The <code>union-rectangle</code> macro modifies the <i>rectangle</i> to be the union of <i>rectangle</i> and (<i>left top right bottom</i>).	

unit-transform

Variable

Summary	The list (1 0 0 1 0 0).	
Package	<code>graphics-ports</code>	
Signature	<code>*unit-transform*</code>	
Description	The <code>*unit-transform*</code> variable holds the list (1 0 0 1 0 0) which is the unit transform I, such that $X = XI$, where X is a 3-vector. Graphics ports are initialized with the unit transform in their graphics state. This means that port coordinate axes are initially the same as the window axes.	

unit-transform-p

Function

Summary	Returns τ if a given transform is a unit transform.	
Package	<code>graphics-ports</code>	
Signature	<code>unit-transform-p transform => bool</code>	
Arguments	<i>transform</i>	A transform.
Values	<i>bool</i>	A boolean.
Description	The <code>unit-transform-p</code> returns τ if <i>transform</i> is the unit transform.	

unless-empty-rect-bind*Macro*

Summary	Binds the elements of a rectangle to four variables, and if the rectangle has a non-zero area, executes a body of code.	
Package	<code>graphics-ports</code>	
Signature	<code>unless-empty-rect-bind ((<i>x y width height</i>) <i>rectangle</i>) &body <i>body</i> => <i>result</i></code>	
Arguments	<i>x</i>	A variable.
	<i>y</i>	A variable.
	<i>width</i>	A variable.
	<i>height</i>	A variable.
	<i>rectangle</i>	A rectangle.
	<i>body</i>	A body of Lisp code.
Values	<i>result</i>	The return value of the last form executed in <i>body</i> .
Description	The <code>unless-empty-rect-bind</code> macro binds <i>x</i> , <i>y</i> , <i>width</i> , and <i>height</i> to the appropriate values from <i>rectangle</i> and if the <i>width</i> and <i>height</i> are both positive, executes the <i>body</i> forms.	

untransform-distance*Function*

Summary	Transforms a distance by the rotation and scale of the inverse of a given transform.	
Package	<code>graphics-ports</code>	
Signature	<code>untransform-distance <i>transform dx dy</i> => <i>x, y</i></code>	
Arguments	<i>transform</i>	A transform.

	<i>dx</i>	A real number.
	<i>dy</i>	A real number.
Values	<i>x</i>	A real number.
	<i>y</i>	A real number.
Description	The <code>untransform-distance</code> function transform the distance (<i>dx dy</i>) by the rotation and scale of the effective inverse of <i>transform</i> . The translation in the inverse transform is ignored. The transformed distance (<i>dx dy</i>) is returned as two values.	

untransform-distances

Function

Summary	Transforms a list of integer pairs representing distances by the inverse of a transform.	
Package	<code>graphics-ports</code>	
Signature	<code>untransform-distances transform distances => result</code>	
Arguments	<i>transform</i>	A transform.
	<i>distances</i>	A list of pairs of real numbers.
Values	<i>result</i>	A list of pairs of real numbers.
Description	The <code>untransform-distances</code> function transforms a list of alternating (<i>dx dy</i>) pairs in <i>distances</i> by the effective inverse of <i>transform</i> . Transformed values are returned as a new list.	

untransform-point

Function

Summary	Transforms a point by multiplying it by the inverse of a given transform.	
---------	---	--

Package	<code>graphics-ports</code>	
Signature	<code>untransform-point transform x y => x2, y2</code>	
Arguments	<i>transform</i>	A transform.
	<i>x</i>	A real number.
	<i>y</i>	A real number.
Values	<i>x2</i>	A real number.
	<i>y2</i>	A real number.
Description	The <code>untransform-point</code> function transform the point (<i>x y</i>) by effectively multiplying it by the inverse of <i>transform</i> . The transformed (<i>x y</i>) is returned as two values.	

untransform-points*Function*

Summary	Transforms a list of points by the inverse of a given transform.	
Package	<code>graphics-ports</code>	
Signature	<code>untransform-points transform points &optional into => result</code>	
Arguments	<i>transform</i>	A transform.
	<i>points</i>	A list of pairs of real numbers.
	<i>into</i>	A list.
Values	<i>result</i>	A list of pairs of real numbers.
Description	The <code>untransform-points</code> function transforms a list of alternating (<i>x y</i>) pairs in <i>points</i> by the effective inverse of <i>transform</i> . If <i>into</i> is supplied it must be a list the same length as <i>points</i> . If <i>into</i> is not supplied, a new list is returned.	

validate-rectangle

Function

Summary	Validates the rectangle associated with the object, marks it as already drawn.	
Package	<code>graphics-ports</code>	
Signature	<code>validate-rectangle <i>object</i> &optional <i>x y width height</i></code> <code>=> <i>result</i></code>	
Arguments	<i>object</i>	An instance of a subclass of <code>graphics-ports-mixin</code> or a subclass of <code>pinboard-object</code> .
	<i>x</i>	A real number.
	<i>y</i>	A real number.
	<i>width</i>	A real number.
	<i>height</i>	A real number.
Values	<i>result</i>	A boolean.
Description	<p>The given area of <i>object</i> is marked as not needing to be displayed. This can be useful if you want to draw that area immediately and avoid it being drawn again by the window system. By default it validates the whole rectangle, but this can be limited by passing the <code>&optional</code> arguments.</p> <p>The <i>result</i> is non-<code>nil</code> if the function succeeds and <code>nil</code> if it fails (doing nothing).</p> <p>Note: this function is not fully implemented on all platforms. On Windows, it succeeds for all valid values of <i>x</i>, <i>y</i>, <i>width</i> and <i>height</i>.</p> <p>On Cocoa, it fails if <i>x</i>, <i>y</i>, <i>width</i> and <i>height</i> are passed.</p> <p>On Motif, it fails in all cases.</p>	
See also	<code>invalidate-rectangle</code>	

with-dither*Macro*

Summary	Specifies a dither for use within a specified body of code.	
Package	<code>graphics-ports</code>	
Signature	<code>with-dither (<i>dither-or-size</i>) &body <i>body</i> => <i>result</i></code>	
Arguments	<i>dither-or-size</i>	See Description.
	<i>body</i>	A body of Lisp code.
Values	<i>result</i>	The return value of the last form executed in <i>body</i> .
Description	<p>The <code>with-dither</code> function specifies a dither for use within <i>body</i>. The <i>dither-or-size</i> argument can be a dither mask object from <code>make-dither</code> or a size, in which case a dither of that size is created.</p> <p>Note: dithers do not affect drawing or the antialiasing that occurs when drawing in Cocoa.</p>	
See also	<code>dither-color-spec</code> <code>make-dither</code> <code>initialize-dithers</code>	

with-graphics-mask*Macro*

Summary	Binds the mask slot of a graphics port to a rectangular area across the execution of a body of code.	
Package	<code>graphics-ports</code>	
Signature	<code>with-graphics-mask (<i>port mask mask-x mask-y</i> &key) &body <i>body</i> => <i>result</i></code>	
Arguments	<i>port</i>	A graphics port.

	<i>mask</i>	A list of the form (<i>x y width height</i>) or <code>nil</code> .
	<i>mask-x</i>	An integer.
	<i>mask-y</i>	An integer.
	<i>body</i>	A body of Lisp code.
Values	<i>result</i>	The return value of the last form executed in <i>body</i> .
Description	<p>The <code>with-graphics-mask</code> macro binds the <code>mask</code> slot of <i>port</i>'s graphic state to a rectangular area across the execution of <i>body</i>.</p> <p>By default, <i>mask-x</i> and <i>mask-y</i> are both 0. These values are used only on Motif.</p>	
Example	<p>For a mask value of (<i>x y width height</i>) drawing is limited to the rectangular region whose X coordinate is in the range</p> $\mathbf{mask-x + x \quad to \quad mask-x + x + width}$ <p>and whose Y coordinate is in the range</p> $\mathbf{mask-y + y \quad to \quad mask-y + y + height}$	

with-graphics-rotation

Macro

Summary	Performs a call to <code>apply-rotation</code> with a given angle for the duration of the macro's body.	
Package	<code>graphics-ports</code>	
Signature	<code>with-graphics-rotation (port angle) &body body => result</code>	
Arguments	<i>port</i>	A graphics port.
	<i>angle</i>	A real.
	<i>body</i>	A body of Lisp code.

Values	<i>result</i>	The return value of the last form executed in <i>body</i> .
Description	The <code>with-graphics-rotation</code> marco performs a call to (<code>apply-rotation</code> <i>transform</i> <i>angle</i>) on the port's transform for the duration of the body of the macro.	
See also	<code>apply-rotation</code>	

with-graphics-scale*Macro*

Summary	Performs a call to <code>apply-scale</code> with a given scale for the duration of the macro's body.	
Package	<code>graphics-ports</code>	
Signature	<code>with-graphics-scale</code> (<i>port</i> <i>sx</i> <i>sy</i>) &body <i>body</i> => <i>result</i>	
Arguments	<i>port</i>	A graphics port.
	<i>sx</i>	A real number.
	<i>sy</i>	A real number.
	<i>body</i>	A body of Lisp code.
Values	<i>result</i>	The return value of the last form executed in <i>body</i> .
Description	The <code>with-graphics-scale</code> macro performs a call to (<code>apply-scale</code> <i>transform</i> <i>sx</i> <i>sy</i>) on the port's transform for the duration of the body of the macro.	
See also	<code>apply-scale</code>	

with-graphics-state

Macro

Summary	Binds the graphics state values of a port to a list of arguments and executes a body of code.	
Package	<code>graphics-ports</code>	
Signature	<code>with-graphics-state</code> (<i>port</i> &rest <i>args</i> &key <i>transform foreground background</i> <i>operation thickness scale-thickness dashed</i> <i>dash line-end-style line-joint-style mask font</i> <i>state fill-style stipple pattern mask-x mask-y</i>) <i>body => result</i>	
Arguments	<i>port</i>	A graphics port.
	<i>body</i>	A body of Lisp code.
Values	<i>result</i>	The return value of the last form executed in <i>body</i> .
Description	The <code>with-graphics-state</code> macro binds the graphics state values for the specified port to the values specified in the <i>args</i> list. The keyword arguments <i>args</i> correspond to the slots in the graphics state, as described in <code>set-graphics-state</code> . See the "Graphics state" section in the <i>LispWorks CAPI User Guide</i> for valid values for <i>args</i> .	

For example:

```
(with-graphics-state (port :thickness 12  
                          :foreground fore-color) ...)
```

Arguments that are not supplied default to the current state of that slot in the graphics state. The arguments *fill-style*, *stipple*, *pattern*, *mask-x* and *mask-y* are used only on Unix.

An extra keyword argument `:state` can be used. The value must be a graphics state object created by a call to `make-graphics-state`. The contents of the graphics state object passed are used instead of the port's state.

Example

```
(setf gstate (make-graphics-state))

(setf (graphics-state-foreground gstate) my-color)

(with-graphics-state (port :state gstate)
  (draw-rectangle port image-1 100 100))
```

See also `set-graphics-state`

with-graphics-transform

Macro

Summary Combines a given transform with the transform of a port for the duration of the macro.

Package `graphics-ports`

Signature `with-graphics-transform (port transform) &body body`
`=> result`

Arguments

<i>port</i>	A graphics port.
<i>transform</i>	A transform.
<i>body</i>	A body of Lisp code.

Values

<i>result</i>	The return value of the last form executed in <i>body</i> .
---------------	---

Description The `with-graphics-transform` macro combines the transform associated with the graphics port *port* with *transform* during the body of the macro. The port is given a new transform obtained by pre-multiplying its current transform with *transform*. This has the effect of *preceding* any translation, scaling and rotation operations specified in the body of the macro by those operations embodied in *transform*.

with-graphics-translation

Macro

Summary	Applies a translation to a given port for the duration of the macro.	
Package	<code>graphics-ports</code>	
Signature	<code>with-graphics-translation (<i>port dx dy</i>) &body <i>body</i> => <i>result</i></code>	
Arguments	<i>port</i>	A graphics port.
	<i>dx</i>	A real number.
	<i>dy</i>	A real number.
	<i>body</i>	A body of Lisp code.
Values	<i>result</i>	The return value of the last form executed in <i>body</i> .
Description	The <code>with-graphics-translation</code> macro performs a call to <code>(apply-translation <i>transform dx dy</i>)</code> on the port's <code>transform</code> for the duration of <i>body</i> of the macro.	

with-inverse-graphics

Macro

Summary	Executes all drawing function calls to a given port within the body of the macro with foreground and background colors swapped.	
Package	<code>graphics-ports</code>	
Signature	<code>with-inverse-graphics (<i>port</i>) &body <i>body</i> => <i>result</i></code>	
Arguments	<i>port</i>	A graphic port.
	<i>body</i>	A body of Lisp code.

Values	<i>result</i>	The return value of the last form executed in <i>body</i> .
Description	The <code>with-inverse-graphics</code> macro ensures that all drawing function calls to <i>port</i> within the body of the macro are executed with the <code>foreground</code> and <code>background</code> slots of the graphics state of the port swapped around.	

without-relative-drawing*Macro*

Summary	Evaluates a body of Lisp code with the <i>relative</i> and <i>collect</i> internal variables of the port set to <code>nil</code> .	
Package	<code>graphics-ports</code>	
Signature	<code>without-relative-drawing (port) &body body => result</code>	
Arguments	<i>port</i>	A graphic port.
	<i>body</i>	A body of Lisp code.
Values	<i>result</i>	The return value of the last form executed in <i>body</i> .
Description	The <code>with-relative-drawing</code> macro evaluates the code in <i>body</i> with the <i>relative</i> and <i>collect</i> internal variables of the pixmap graphics port <i>port</i> set to <code>nil</code> to turn off the port's collecting of drawing bounds and automatic shifting of its origins. Use this macro only within a <code>with-pixmap-graphics-port</code> macro.	

with-pixmap-graphics-port*Macro*

Summary	Binds a port to a new pixmap graphics port for the duration of the macro's code body.	
---------	---	--

Package	<code>graphics-ports</code>	
Signature	<code>with-pixmap-graphics-port</code> (<i>port pane width height</i> &key <i>background collect relative clear</i>) &body <i>body</i>) => <i>result</i>	
Arguments	<i>port</i>	A graphic port.
	<i>pane</i>	An output pane.
	<i>width</i>	An integer.
	<i>height</i>	An integer.
	<i>background</i>	A color keyword.
	<i>collect</i>	A boolean.
	<i>relative</i>	A boolean.
	<i>clear</i>	A list or <code>t</code> .
	<i>body</i>	A body of Lisp code.
Values	<i>result</i>	The return value of the last form executed in <i>body</i> .
Description	The <code>with-pixmap-graphics-port</code> macro binds <i>port</i> to a new pixmap graphics-port. <i>pane</i> and the other arguments are passed to <code>create-pixmap-port</code> . The <i>body</i> is then evaluated. The port is destroyed when <i>body</i> returns.	

with-transformed-area

Macro

Summary	Transforms a rectangle using a port's transform, and binds the resulting values to a variable across the evaluation of the macro's body.
Package	<code>graphics-ports</code>

Signature	<code>with-transformed-area</code> (<i>points port left top right bottom</i>) &body <i>body</i>	
Arguments	<i>points</i>	A variable.
	<i>port</i>	A graphics port.
	<i>left</i>	A real number.
	<i>top</i>	A real number.
	<i>right</i>	A real number.
	<i>bottom</i>	A real number.
	<i>body</i>	A body of Lisp code.
Values	<i>result</i>	The return value of the last form executed in <i>body</i> .
Description	The <code>with-transformed-area</code> macro transforms a rectangle, binding the resulting four corner points to <i>points</i> for the duration of <i>body</i> . The <i>left top right bottom</i> values represent a rectangular area bounded by four points. The four points are transformed by the <i>port</i> 's transform and the list of eight values (alternating <i>x</i> and <i>y</i> values for four points) bound to the <i>points</i> variable for the duration of the macro body.	

with-transformed-point*Macro*

Summary	Binds a point transformed by a given ports transform to two variables across the body of the macro.	
Package	<code>graphics-ports</code>	
Signature	<code>with-transformed-point</code> (<i>new-x new-y port x y</i>) &body <i>body</i> => <i>result</i>	
Arguments	<i>new-x</i>	A variable.
	<i>new-y</i>	A variable.

	<i>port</i>	A graphics port.
	<i>x</i>	A real number.
	<i>y</i>	A real number.
	<i>body</i>	A body of Lisp code.
Values	<i>result</i>	The return value of the last form executed in <i>body</i> .
Description	The <code>with-transformed-point</code> macro transforms the point given by (<i>x y</i>) using the <i>port</i> 's transform and the resulting values are bound to the <i>new-x</i> and <i>new-y</i> variables. The <i>body</i> of the macro is then evaluated with this binding.	

with-transformed-points

Macro

Summary	Binds a list of transformed points in a port to a list across the execution of the macro's body.	
Package	<code>graphics-ports</code>	
Signature	<code>with-transformed-points (<i>points port</i>) &body <i>body</i> => <i>result</i></code>	
Arguments	<i>points</i>	A list of real numbers.
	<i>port</i>	A graphics port.
Values	<i>result</i>	The return value of the last form executed in <i>body</i> .
Description	The <code>with-transformed-points</code> macro binds <i>points</i> to a new list of <i>x</i> and <i>y</i> values obtained by post-multiplying them by the current transform of <i>port</i> , and then evaluates <i>body</i> . The <i>points</i> symbol must be bound to a list of alternating <i>x</i> and <i>y</i> values representing coordinate points in the <i>port</i> .	

with-transformed-rect*Macro*

Summary	Transforms the coordinates of a rectangle and binds them to four variables for the duration of the macro's body.	
Package	<code>graphics-ports</code>	
Signature	<code>with-transformed-rect (nx1 ny1 nx2 ny2 port x1 y1 x2 y2) &body body => result</code>	
Arguments	<i>nx1</i>	A variable.
	<i>ny1</i>	A variable.
	<i>nx2</i>	A variable.
	<i>ny2</i>	A variable.
	<i>port</i>	A graphics port.
	<i>x1</i>	A real number.
	<i>y1</i>	A real number.
	<i>x2</i>	A real number.
	<i>y2</i>	A real number.
	<i>body</i>	A body of Lisp code.
Values	<i>result</i>	The return value of the last form executed in <i>body</i> .
Description	During the evaluation of the <code>with-transformed-rect</code> macro <i>body</i> , the two points (<i>x1</i> , <i>y1</i>) and (<i>x2</i> , <i>y2</i>) are transformed by the port's current transform and the resulting values bound to the variables named by the <i>nx1 ny1 nx2 ny2</i> args.	

write-external-image*Function*

Summary	Writes external image data to a file.
---------	---------------------------------------

Package `graphics-ports`

Signature `write-external-image external-image file &key if-exists`

Arguments

- `external-image` An external-image.
- `file` A file.
- `if-exists` A keyword.

Description

The `write-external-image` function writes an external image to a file `file`. It writes the image data byte-for-byte without attempting any conversion of the image format.

`if-exists` is passed to `open` when opening `file`. The default value of `if-exists` is `:error`.

See also `externalize-image`

3

COLOR Reference Entries

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

apropos-color-alias-names

Function

Summary	Returns color aliases containing a given string.	
Package	<code>color</code>	
Signature	<code>apropos-color-alias-names <i>substring</i> => <i>list</i></code>	
Arguments	<i>substring</i>	A string.
Values	<i>list</i>	A list of symbols.
Description	Returns a list of symbols whose symbol-names contain <i>substring</i> and which are defined as aliases in the color-database defining color aliases. By convention these are in the keyword package.	

Example In this example, a color alias is defined for the color `indianred1`. `apropos-color-alias-names` only returns this alias, rather than both the alias and the original color, despite the similarity in the names.

```
COLOR 8 > (define-color-alias :myindianred1
           :indianred1)
(#S(COLOR-ALIAS COLOR :INDIANRED1))

COLOR 9 > (apropos-color-names "INDIANRED1")
(:INDIANRED1 :MYINDIANRED1)

COLOR 10 > (apropos-color-alias-names "INDIANRED1")
(:MYINDIANRED1)

COLOR 11 >
```

See also `apropos-color-names`
`apropos-color-spec-names`
`get-all-color-names`

apropos-color-names

Function

Summary	Returns colors and color aliases containing a given string.
Package	<code>color</code>
Signature	<code>apropos-color-names <i>substring</i> => <i>list</i></code>
Arguments	<i>substring</i> A string.
Values	<i>list</i> A list of symbols.
Description	Returns a list of symbols whose symbol-names contain <i>substring</i> and which are present in the color-database defining color aliases. By convention these are in the keyword package.

Example

```
COLOR-4> (color:apropos-color-names "RED")
(:ORANGERED3 :ORANGERED1 :INDIANRED3 :INDIANRED1
 :PALEVIOLETRED :RED :INDIANRED :INDIANRED2
 :INDIANRED4 :ORANGERED :MEDIUMVIOLETRED
 :VIOLETRED :ORANGERED2 :ORANGERED4 :RED1 :RED2 :RED3
 :RED4 :PALEVIOLETRED1 :PALEVIOLETRED2 :PALEVIOLETRED3
 :PALEVIOLETRED4 :VIOLETRED3 :VIOLETRED1 :VIOLETRED2
 :VIOLETRED4)
```

See also

```
apropos-color-alias-names
apropos-color-spec-names
get-all-color-names
```

apropos-color-spec-names

Function

Summary Returns colors containing a given string.

Package `color`

Signature `apropos-color-spec-names substring => list`

Arguments *substring* A string.

Values *list* A list of symbols.

Description Returns a list of symbols whose symbol-names contain *substring* and which are defined as original entries in the color-database defining color aliases. By convention these are in the keyword package.

Example

```
COLOR 14 > (define-color-alias :mygray100 :gray100)
(#S(COLOR-ALIAS COLOR :GRAY100))

COLOR 15 > (apropos-color-names "GRAY100")
(:MYGRAY100 :GRAY100)

COLOR 16 > (ap]ropos-color-spec-names "GRAY100")
(:GRAY100)

COLOR 17 >
```

See also `apropos-color-alias-names`
`apropos-color-names`
`get-all-color-names`

color-alpha

Function

Summary Returns the alpha component of a color specification.

Package `color`

Signature `color-alpha color-spec &optional default => alpha`

Arguments *color-spec* A color specification.
default A number between 0 and 1.

Values *alpha* The alpha component of *color-spec*.

Description *color-spec* is a color specification in any model.
`color-alpha` returns the alpha component of *color-spec*. If *color-spec* does not have an alpha component, then *default* is returned.
The default value of *default* is 1.0.

See also `make-hsv`
`make-rgb`
`make-gray`

color-<component>

Function

Summary Returns the associated component of a color specification.

Package `color`

Signature	<code>color-red <i>color-spec</i> => <i>color-component</i></code> <code>color-green <i>color-spec</i> => <i>color-component</i></code> <code>color-blue <i>color-spec</i> => <i>color-component</i></code> <code>color-hue <i>color-spec</i> => <i>color-component</i></code> <code>color-saturation <i>color-spec</i> => <i>color-component</i></code> <code>color-value <i>color-spec</i> => <i>color-component</i></code>
Arguments	<code><i>color-spec</i></code> A color specification.
Values	<code><i>color-component</i></code> A color component from the appropriate color model.
Description	If <code><i>color-spec</i></code> is not from the appropriate color model (<code>:rgb</code> in the case of <code>color-red</code> , <code>color-green</code> and <code>color-blue</code> , and <code>:hsv</code> in the case of <code>color-hue</code> , <code>color-saturation</code> and <code>color-value</code>) then the component is calculated.
Example	<pre>COLOR 31 > (color:make-rgb 1.0s0 0.0s0 0.0s0) #(:RGB 1.0S0 0.0S0 0.0S0) COLOR 32 > (color-red *) 1.0S0 COLOR 33 > (color-green **) 0.0S0 COLOR 34 > (color-value ***) 1.0S0 COLOR 35 ></pre>
See also	<code>make-hsv</code> <code>make-rgb</code> <code>make-gray</code> <code>color-model</code> <code>color-level</code>

color-database

Variable

Summary The current color-database.

Package	<code>color</code>
Description	This should contain definitions for all the colors used in the environment when you start it. Those colors are determinable from the file <code>config/colors.db</code> .
Example	To replace the current color database with a new one, do the following: <pre>(setf color:*color-database* (color:make-color-db))</pre>
See also	<code>delete-color-translation</code> <code>read-color-db</code> <code>load-color-database</code>

color-level*Function*

Summary	Returns the gray level of a color specification.
Package	<code>color</code>
Signature	<code>color-level</code> <i>color-spec</i> => <i>gray-level</i>
Arguments	<i>color-spec</i> A color specification.
Values	<i>gray-level</i> Color component from the <code>:gray</code> model.
Description	Return the gray level of <i>color-spec</i> . If <i>color-spec</i> is not from the <code>:GRAY</code> model, the component is calculated.
Example	<pre>COLOR 2 > (color:make-gray 0.66667s0) #(:GRAY 0.66667s0) COLOR 3 > (color-level *) 0.66667s0 COLOR 4 ></pre>

See also `make-hsv`
`make-rgb`
`make-gray`
`color-model`
`color-<component>`

color-model

Function

Summary Returns the color-model for a color-spec.

Package `color`

Signature `color-model color-spec => color-model`

Arguments `color-spec` A color specification.

Values `color-model` `:gray`, `:rgb`, OR `:hsv`.

Example `COLOR 29 > (color:make-gray 0.66667s0)`
`#(:GRAY 0.66667S0)`

`COLOR 30 > (color-model *)`
`:GRAY`

`COLOR 31 >`

See also `make-hsv`
`make-rgb`
`make-gray`
`color-<component>`
`color-level`

colors=

Function

Summary Tests to see if two colors are equal.

Package `color`

Signature	<code>colors= color1 color2 &optional tolerance => bool</code>	
Arguments	<code>color1</code>	A color specification.
	<code>color2</code>	A color specification.
	<code>tolerance</code>	A tolerance level within which <code>color1</code> and <code>color2</code> may vary. The default value is <code>0.001s0</code> .
Values	<code>bool</code>	<code>⊤</code> if the two colors are equal within the given tolerance, <code>nil</code> otherwise.
Description	Return <code>⊤</code> if the two colors are equal to the given tolerance.	
See also	<code>ensure-<command></code> <code>convert-color</code>	

convert-color*Function*

Summary	Return the representation of a color specification on a given graphics port.	
Package	<code>color</code>	
Signature	<code>convert-color port color &key errorp => color-rep</code>	
Arguments	<code>port</code>	A graphics port.
	<code>color</code>	A color specification.
	<code>errorp</code>	If <code>⊤</code> , check for errors. By default, this is <code>⊤</code> .
Values	<code>color-rep</code>	Representation of <code>color</code> on <code>port</code> .
Description	Return the representation of <code>color</code> on the given graphics port <code>port</code> . In CLX, this is the “pixel” value, which corresponds to an index into the default colormap. It is more efficient to use	

the result of `convert-color` in place of its argument in drawing function calls, but the penalty is the risk of erroneous colors being displayed should the colormap or the colormap entry be changed.

See also `colors=`
`ensure-<command>`
`unconvert-color`

define-color-alias

Function

Summary	Lets you define an alias for a color specification or alias.	
Package	<code>color</code>	
Signature	<code>define-color-alias name color &optional if-exists => alias</code>	
Arguments	<i>name</i>	The name of the new alias.
	<i>color</i>	A color specification for the new alias.
	<i>if-exists</i>	This can be one of the following: <code>:replace</code> — Replace any existing alias. <code>:error</code> — Raise an error if alias is already defined. <code>:ignore</code> — Ignore redefinition of an alias. By default, it is <code>:replace</code> .
Values	<i>alias</i>	The color alias.
Description	Define <i>name</i> to be a color alias for <i>color</i> , which may be another color alias or a color-spec.	
Example 1	<pre>COLOR 16 > (define-color-alias :mygray :darkslategray) (#S(COLOR-ALIAS COLOR :DARKSLATEGRAY))</pre>	

```
COLOR 17 > (define-color-alias :mygray :darkslategray
             :error)
```

```
Error: :MYGRAY names an existing alias for #(:RGB
0.1843133S0 0.309803S0 0.309803S0)
```

```
  1 (continue) Replace :MYGRAY with the alias
: DARKSLATEGRAY
  2 Continue, without redefining alias :MYGRAY
  3 Try a new name for the alias, instead of :MYGRAY
  4 (abort) Return to level 0.
  5 Return to top loop level 0.
  6 Destroy process.
```

```
Type :c followed by a number to proceed or type :? for
other options
```

```
COLOR 18 : 1 >
```

Example 2

```
COLOR 19 > (define-color-alias :lispworks-blue
             (make-rgb 0.70s0 0.90s0 0.99s0))
(#S(COLOR-ALIAS COLOR #(:RGB 0.699999S0 0.9S0 0.99S0)))
```

```
COLOR 20 >
```

See also

```
get-color-alias-translation
get-color-spec
```

define-color-models

Macro

Summary	Defines <i>all</i> the color models.
Package	color
Signature	<code>define-color-models <i>model-descriptors</i>=> <i>color-models</i></code>
Arguments	<i>model-descriptors</i> A list, each element being a model-descriptor.
Values	<i>color-models</i> The color models defined.
Description	A model descriptor has the syntax:

(model-name component-descr)*

A *component-descr* is a list:

(component-name lowest-value highest-value)

The default color models are defined by the following form:

```
(define-color-models ((:rgb (red 0.0 1.0)
                          (green 0.0 1.0)
                          (blue 0.0 1.0))
                     (:hsv (hue 0.0 5.99999)
                          (saturation 0.0 1.0)
                          (value 0.0 1.0))
                     (:gray (level 0.0 1.0))))
```

If you want to keep existing color models, add your new ones to this list: only one `define-color-models` form is recognized. The form should be compiled.

Example

To replace the HSV color model with a CMYK model, while retaining the other color models:

```
(define-color-models ((:rgb (red 0.0 1.0)
                          (green 0.0 1.0)
                          (blue 0.0 1.0))
                     (:cmymk (cyan 0.0 1.0)
                          (magenta 0.0 1.0)
                          (yellow 0.0 1.0)
                          (black 0.0 1.0))
                     (:gray (level 0.0 1.0))))
```

delete-color-translation

Function

Summary	Removes an entry from the color-database.
Package	<code>color</code>
Signature	<code>delete-color-translation <i>color-name</i> => <no values></code>
Arguments	<i>color-name</i> A defined color spec or alias.

Values	None.
Description	Both original entries and aliases can be removed.
See also	<code>load-color-database</code> <code>*color-database*</code> <code>read-color-db</code>

ensure-*<command>**Function*

Summary	Return a color specification for a given model. The model depends on the particular function called
Package	<code>color</code>
Signature	<code>ensure-rgb <i>color-spec</i> => <i>result</i></code> <code>ensure-hsv <i>color-spec</i> => <i>result</i></code> <code>ensure-gray <i>color-spec</i> => <i>result</i></code> <code>ensure-model-color <i>color-spec model</i> => <i>result</i></code> <code>ensure-color <i>color-spec match-color-spec</i> => <i>result</i></code>
Arguments	For all functions: <i>color-spec</i> A color specification. <i>match-color-spec</i> A color specification. <i>model</i> A color-model (<code>:rgb</code> , <code>:hsv</code> or <code>:gray</code>).
Values	<i>result</i> A color specification.
Description	These functions all return a color specification, given (at least) a color specification as argument. <code>ensure-rgb</code> , <code>ensure-hsv</code> and <code>ensure-gray</code> all return a color specification in the appropriate model. If <i>color-spec</i> is in the same model, it is just returned. Otherwise a new color specifi-

ation for that model is calculated. Thus, `ensure-rgb` returns a color specification in the RGB color model, whatever color model is used in *color-spec*.

If *color-spec* has an alpha component, then *result* has that same alpha component.

`ensure-model-color` is similar to the above three functions, except that a color-model *model* is explicitly passed as an argument to the function. The color-spec returned is in the color-model specified by *model*.

`ensure-color` returns a color specification for *color-spec*, in the color model specified by *match-color-spec*. Thus, color specifications may be converted from one model to another with having to explicitly state the color model.

Example

```
COLOR 36 > (ensure-hsv (make-rgb 0.70s0 0.90s0 0.99s0))
#(:HSV 4.31033S0 0.707069S0 0.99S0)
```

```
COLOR 37 > (ensure-gray (make-rgb 0.70s0 0.90s0
0.99s0))
#(:GRAY 0.863331S0)
```

```
COLOR 39 > (ensure-model-color (make-rgb 0.70s0 0.90s0
0.99s0) :hsv)
#(:HSV 4.31033S0 0.707069S0 0.99S0)
```

```
COLOR 43 > (ensure-color (make-hsv 0.70s0 0.90s0
0.99s0) (make-rgb 0.70s0 0.90s0 0.99s0))
#(:RGB 0.99S0 0.890999S0 0.92069924)
```

See also

```
convert-color
colors=
```

get-all-color-names

Function

Summary Returns a list of all color-names in the color database.

Package `color`

Signature	<code>get-all-color-names &optional sort => color-names</code>	
Arguments	<code>sort</code>	If <code>t</code> , sort list of color names alphanumerically. By default, this is <code>nil</code> .
Values	<code>color-names</code>	A list of all color names in the color database.
Description	Returns a list of all color-names in the color database. By convention these are symbols in the keyword package. The returned list is alphanumerically sorted on the symbol-names if the optional argument is non- <code>nil</code> .	
See also	<code>apropos-color-names</code> <code>apropos-color-spec-names</code> <code>apropos-color-alias-names</code>	

get-color-alias-translation*Function*

Summary	Return the ultimate color name associated with <i>color-alias</i> .	
Package	<code>color</code>	
Signature	<code>get-color-alias-translation color-alias => color-name</code>	
Arguments	<code>color-alias</code>	A defined color alias.
Values	<code>color-name</code>	The color name associated with <i>color-alias</i> .
Example	<pre>COLOR 23 > (color:define-color-alias :lispworks-blue (color:make-rgb 0.70s0 0.90s0 0.99s0)) (#S(COLOR-ALIAS COLOR #(:RGB 0.699999s0 0.9s0 0.99s0))) COLOR 24 > (color:define-color-alias :color-background :lispworks-blue) (#S(COLOR-ALIAS COLOR :LISPWORKS-BLUE))</pre>	

```

COLOR 25 > (color:define-color-alias
            :listener-background :color-background)
(#S(COLOR-ALIAS COLOR :COLOR-BACKGROUND))

COLOR 26 > (get-color-alias-translation
            :listener-background)
:LISPWORKS-BLUE

COLOR 27 > (color:get-color-alias-translation
            :color-background)
:LISPWORKS-BLUE

COLOR 28 >

```

See also `define-color-alias`
`get-color-spec`

get-color-spec

Function

Summary	Returns the color-spec for a color.	
Package	<code>color</code>	
Signature	<code>get-color-spec</code> <i>color</i> => <i>color-spec</i>	
Arguments	<i>color</i>	A defined color specification, color alias, or an original color name.
Values	<i>color-spec</i>	A color specification.
Description	Returns the color-spec for <i>color</i> , which can be a color-spec, a color-alias, or an original color name.	
Example	<pre> COLOR 28 > (color:define-color-alias :lispworks-blue (color:make-rgb 0.70s0 0.90s0 0.99s0)) (#S(COLOR-ALIAS COLOR #(:RGB 0.699999S0 0.9S0 0.99S0))) COLOR 29 > (color:define-color-alias :color-background :lispworks-blue) (#S(COLOR-ALIAS COLOR :LISPWORKS-BLUE)) </pre>	

```

COLOR 30 > (color:define-color-alias
             :listener-background :color-background)
(#S(COLOR-ALIAS COLOR :COLOR-BACKGROUND))

COLOR 31 > (get-color-spec :listener-background)
#(:RGB 0.699999S0 0.9S0 0.99S0)

COLOR 32 > (get-color-spec :color-background)
#(:RGB 0.699999S0 0.9S0 0.99S0)

COLOR 33 > (get-color-spec :lispworks-blue)
#(:RGB 0.699999S0 0.9S0 0.99S0)

COLOR 34 > (get-color-spec
            #(:RGB 0.70S0 0.90S0 0.99S0))
#(:RGB 0.699999S0 0.9S0 0.99S0)

COLOR 35 >

```

See also `define-color-alias`
`get-color-alias-translation`

load-color-database

Function

Summary	Loads a color database.	
Package	<code>color</code>	
Signature	<code>load-color-database <i>data</i> => <no values></code>	
Arguments	<i>data</i>	A description of a color database.
Values	None.	
Description	This loads the color database with color definitions contained in <i>data</i> , which should have been obtained via the functions <code>color:read-color-db</code> . The colors thus defined may not be replaced by color aliases.	

See also `*color-database*`
`delete-color-translation`
`read-color-db`

make-gray

Function

Summary Returns a color specification in the gray model.

Package `color`

Signature `make-gray level &optional alpha => color-spec`

Arguments *level* A color component used to define the gray level required.
alpha A number between 0 and 1, or `nil`.

Values *color-spec* A color specification.

Description Return a color-spec in the `:GRAY` model with component *level*.
Note that short-floats are used for the component; this results in the most efficient color conversion process. However, any floating point number type can be used.
alpha indicates the alpha value of the color. 0 means it is transparent, 1 means it is solid. If *alpha* is `nil` or not specified then the color does not have an alpha component and it is assumed to be solid.

Example

```
COLOR 25 > (color:make-gray 0.66667s0)
#(:GRAY 0.66667S0)
```

See also `make-hsv`
`make-rgb`
`color-model`

```
color-<component>
color-level
color-alpha
```

make-hsv*Function*

Summary	Returns a color specification in the hue-saturation-value model.	
Package	color	
Signature	make-hsv <i>hue saturation value</i> &optional <i>alpha</i> => <i>color-spec</i>	
Arguments	<i>hue</i>	A hue component.
	<i>saturation</i>	A saturation component.
	<i>value</i>	A value component.
	<i>alpha</i>	A number between 0 and 1, or <code>nil</code> .
Values	<i>color-spec</i>	A color specification.
Description	<p>Return a color-spec in the <code>:hsv</code> model with components <i>hue</i>, <i>saturation</i> and <i>value</i>.</p> <p>Note that short-floats are used for each component; this results in the most efficient color conversion process. However, any floating-point number type can be used.</p> <p><i>alpha</i> indicates the alpha value of the color. 0 means it is transparent, 1 means it is solid. If <i>alpha</i> is <code>nil</code> or not specified then the color does not have an alpha component and it is assumed to be solid.</p>	
Example	<pre>COLOR 27 > (color:make-hsv 1.2s0 0.5s0 0.9s0) #(:HSV 1.2S0 0.5S0 0.9S0)</pre>	

See also `make-rgb`
`make-gray`
`color-model`
`color-<component>`
`color-level`
`color-alpha`

make-rgb

Function

Summary Returns a color specification in the red-green-blue model.

Package `color`

Signature `make-rgb red green blue &optional alpha => color-spec`

Arguments

<i>red</i>	A red component.
<i>green</i>	A green component.
<i>blue</i>	A blue component.
<i>alpha</i>	A number between 0 and 1, or <code>nil</code> .

Values

<i>color-spec</i>	A color specification.
-------------------	------------------------

Description

Return a `color-spec` in the `:RGB` model with components *red*, *green* and *blue*.

Note that short floats are used for each component; this results in the most efficient color conversion process. However, any floating point number type can be used.

alpha indicates the alpha value of the color. 0 means it is transparent, 1 means it is solid. If *alpha* is `nil` or not specified then the color does not have an alpha component and it is assumed to be solid.

Example

The object returned by the following call defines the color red in the RGB model:

```
COLOR 25 > (color:make-rgb 1.0s0 0.0s0 0.0s0)
#(:RGB 1.0S0 0.0S0 0.0S0)
```

See also

```
make-hsv
make-gray
color-model
color-<component>
color-level
color-alpha
```

read-color-db*Function*

Summary Reads the color definitions contained in a file.

Package `color`

Signature `read-color-db &optional file => color-database`

Arguments *file* A filename or pathname containing the color definitions to be read. If *file* is not given, `read-color-db` uses the default color definitions file in the LispWorks library.

Values *color-database* A database definition.

Description This reads color definitions from the given *file* (a filename or pathname). The returned data structure can be passed to `color:load-color-database`. The format of the file is:

```
#(:RGB 1.0s0 0.980391s0 0.980391s0)    snow
#(:RGB 0.972548s0 0.972548s0 1.0s0)    GhostWhite
...
```

Each line contains a color definition which consists of a color-spec and a name. The names are converted to uppercase and interned in the keyword package. Whitespace in names is preserved.

See also `load-color-database`
`*color-database*`
`delete-color-translation`

unconvert-color

Function

Summary Returns a color specification for a color representation.

Package `color`

Signature `unconvert-color port color-rep => color`

Arguments *port* A graphics port.
color-rep A color representation on *port*.

Values *color* A color specification.

Description The function `unconvert-color` returns a color specification corresponding to the color representation *color-rep* on the Graphics Port *port*.

If *color-rep* is a color specification, a symbol or a color alias, then it is simply returned since the color system can interpret these directly.

Otherwise *color-rep* is assumed to be a color representation on *port*, like those returned by `convert-color` and `image-access-pixel`, and a corresponding RGB value is returned.

See also `convert-color`
`image-access-pixel`

Index

A

- abort-callback function 1
- abort-dialog function 2
- abort-exit-confirmer function 3
- :accelerator initarg 275
- Accelerators 212, 275
- accepts-focus-p generic function 4
- :accepts-focus-p initarg 50, 135
- accessor functions
 - application-interface-application-menu 43
 - application-interface-dock-menu 43
 - application-interface-message-callback 43
 - button-alternate-callback 385
 - button-armed-image 14
 - button-cancel-p 14
 - button-default-p 14
 - button-disabled-image 14
 - button-enabled 14
 - button-image 14
 - button-press-callback 385
 - button-selected 14
 - button-selected-disabled-image 14
 - button-selected-image 14
 - callbacks-action-callback 25
 - callbacks-callback-type 25
 - callbacks-extend-callback 25
 - callbacks-retract-callback 25
 - callbacks-selection-callback 25
 - capi-object-name 28
 - capi-object-plist 28
 - choice-initial-focus-item 32
 - choice-interaction 32
 - choice-selection 32
 - cocoa-view-pane-init-function 45
 - cocoa-view-pane-view-class 45
 - collection-items 50
 - collection-items-count-function 50
 - collection-items-get-function 50
 - collection-items-map-function 50
 - collection-print-function 50
 - collection-test-function 50
 - collector-pane-stream 55
 - display-pane-text 93
 - docking-layout-controller 98
 - docking-layout-divider-p 98
 - docking-layout-docking-test-function 98
 - docking-layout-items 98
 - docking-layout-orientation 98
 - document-frame-container 103
 - drawn-pinboard-object-display-callback 113
 - editor-pane-buffer 131
 - editor-pane-change-callback 124
 - editor-pane-enabled 124
 - editor-pane-fixed-fill 124
 - editor-pane-line-wrap-face 124
 - editor-pane-line-wrap-marker 124
 - editor-pane-text 124
 - editor-pane-wrap-style 124
 - element-interface 137
 - element-parent 137
 - filled 142, 394
 - filtering-layout-matches-text 149
 - filtering-layout-state 149

form-title-adjust 157
 form-title-gap 157
 form-vertical-adjust 157
 form-vertical-gap 157
 graph-edge-from 165
 graph-edge-to 165
 graph-node-height 166
 graph-node-in-edges 166
 graph-node-out-edges 166
 graph-node-width 166
 graph-node-x 166
 graph-node-y 166
 graph-object-element 167
 graph-object-object 167
 graph-pane-layout-function 168
 graph-pane-roots 168
 help-key 50, 137, 275, 333, 503
 image-height 594
 image-pinboard-object-image 185
 image-width 594
 interactive-pane-stream 190
 interactive-pane-top-level-function 190
 interface-activate-callback 195
 interface-confirm-destroy-function 195
 interface-create-callback 195
 interface-destroy-callback 195
 interface-geometry-change-callback 195
 interface-help-callback 195
 interface-iconify-callback 195, 201
 interface-menu-bar-items 195
 interface-message-area 195
 interface-override-cursor 195
 interface-pointer-documentation-enabled 195
 interface-title 195
 interface-tooltips-enabled 195
 interface-window-styles 195
 item-collection 220
 item-data 220
 item-print-function 220
 item-selected 220
 item-text 220
 labelled-line-text-foreground 223
 layout-description 225
 layout-ratios 57, 413
 layout-x-adjust 549
 layout-x-gap 178
 layout-x-ratios 178
 layout-y-adjust 549
 layout-y-gap 178
 layout-y-ratios 178
 list-panel-right-click-selection-behavior 228
 list-view-auto-reset-column-widths 236
 list-view-columns 236
 list-view-image-function 236
 list-view-state-image-function 236
 list-view-subitem-function 236
 list-view-subitem-print-functions 236
 list-view-view 236
 menu-image-function 268
 menu-items 268
 menu-object-enabled 280
 menu-popup-callback 279
 menu-title 493
 menu-title-function 493
 ole-control-component-pane 293
 option-pane-enabled 304
 option-pane-enabled-positions 304
 option-pane-image-function 304
 option-pane-popup-callback 304
 option-pane-separator-item 304
 option-pane-visible-items-count 304
 output-pane-create-callback 308
 output-pane-destroy-callback 308
 output-pane-display-callback 308
 output-pane-focus-callback 308
 output-pane-graphics-options 308
 output-pane-input-model 308
 output-pane-resize-callback 308
 output-pane-scroll-callback 308
 pane-layout 19, 195
 password-pane-overwrite-character 327
 pinboard-object-activep 333
 pinboard-object-graphics-args 333
 pinboard-object-pinboard 333
 popup-menu-button-menu 348
 popup-menu-button-menu-function 348
 range-callback 393
 range-end 393
 range-orientation 393
 range-slug-end 393
 range-slug-start 393
 range-start 393

- rich-text-pane-change-callback 403
- rich-text-pane-limit 403
- rich-text-pane-text 403
- screen-depth 415
- screen-height 415
- screen-height-in-millimeters 415
- screen-interfaces 102, 415
- screen-number 415
- screen-width 415
- screen-width-in-millimeters 415
- scroll-bar-line-size 421
- scroll-bar-page-size 421
- shell-pane-command 450
- simple-pane-background 455
- simple-pane-cursor 455
- simple-pane-drop-callback 455
- simple-pane-enabled 149, 455, 509
- simple-pane-font 455
- simple-pane-foreground 455
- simple-pane-horizontal-scroll 455
- simple-pane-scroll-callback 455
- simple-pane-vertical-scroll 455
- simple-pane-visible-border 455
- slider-show-value-p 465
- slider-start-point 465
- switchable-layout-combine-child-constraints 470
- switchable-layout-visible-child 470
- tab-layout-combine-child-constraints 472
- tab-layout-visible-child-function 472
- text-input-pane-before-change-callback 483
- text-input-pane-buttons-enabled 478
- text-input-pane-callback 478
- text-input-pane-caret-position 478
- text-input-pane-change-callback 478
- text-input-pane-completion-function 478
- text-input-pane-confirm-change-function 478
- text-input-pane-editing-callback 478
- text-input-pane-enabled 478
- text-input-pane-max-characters 478
- text-input-pane-navigation-callback 478
- text-input-pane-text 478
- text-input-range-callback 491
- text-input-range-callback-type 491
- text-input-range-end 491
- text-input-range-start 491
- text-input-range-value 491
- text-input-range-wraps-p 491
- titled-object-message 495
- titled-object-message-font 201, 495
- titled-object-title 495
- titled-object-title-font 495
- title-pane-text 492
- toolbar-button-dropdown-menu 503
- toolbar-button-dropdown-menu-function 503
- toolbar-button-dropdown-menu-kind 503
- toolbar-button-image 503
- toolbar-button-popup-interface 503
- toolbar-button-selected-image 503
- toolbar-flat-p 500
- toolbar-object-enabled-function 509
- top-level-interface-external-border 195
- top-level-interface-transparency 195
- tree-view-action-callback-expand-p 520
- tree-view-checkbox-change-callback 520
- tree-view-checkbox-child-function 520
- tree-view-checkbox-initial-status 520
- tree-view-checkbox-next-map 520
- tree-view-checkbox-parent-function 520
- tree-view-checkbox-status 520
- tree-view-children-function 520
- tree-view-expandp-function 520
- tree-view-has-root-line 520
- tree-view-image-function 520
- tree-view-leaf-node-p-function 520
- tree-view-retain-expanded-nodes 520
- tree-view-right-click-extended-match 520
- tree-view-roots 520
- tree-view-state-image-function 520
- :action-callback initarg 25
- :action-callback-expand-p initarg 518
- :activate-callback initarg 193

- activate-pane function 5
- :activep initarg 332, 392
- ActiveX 298
- :adjust initarg 57, 413
- :adjust item in :buttons initarg 482
- :after-input-callback initarg 124
- :alternate-callback initarg 385
- :alternative initarg 275
- analyze-external-image function 551
- append-items generic function 6
- Application menu 42
- application-interface-application-menu accessor function 43
- application-interface-dock-menu accessor function 43
- application-interface-message-callback accessor function 43
- :application-menu initarg 43
- apply-in-pane-process function 6
- apply-rotation function 552
- apply-scale function 552
- apply-translation function 553
- apropos-color-alias-names function 659
- apropos-color-names function 660
- apropos-color-spec-names function 661
- :armed-image initarg 14
- :armed-images initarg 18
- arrow-pinboard-object class 7
- attach-interface-for-callback function 9
- attach-simple-sink function 10
- attach-sink function 11
- augment-font-description function 553
- :auto-menus initarg 193
- :auto-reset-column-widths initarg 235, 286

B

- :background initarg 454
- beep-pane function 12
- :before-change-callback initarg 483
- :before-input-callback initarg 124
- :best-height initarg 193
- :best-width initarg 193
- :best-x initarg 193
- :best-y initarg 193
- :browse-file item in :buttons initarg 481
- :buffer-modes initarg 123

- :buffer-name initarg 55, 123
- built-in scrolling 163
- button class 13
- button-alternate-callback accessor function 385
- button-armed-image accessor function 14
- button-cancel-p accessor function 14
- :button-class initarg 18
- button-default-p accessor function 14
- button-disabled-image accessor function 14
- button-enabled accessor function 14
- :button-height initarg 500
- button-image accessor function 14
- button-panel class 18
- button-press-callback accessor function 385
- :buttons initarg 477
- button-selected accessor function 14
- button-selected-disabled-image accessor function 14
- button-selected-image accessor function 14
- :button-width initarg 500

C

- calculate-constraints generic function 23
- calculate-layout generic function 24
- :callback initarg 13, 15, 148, 279, 392, 421, 477, 490, 502
- :callback-data-function initarg 279
- :callback-object initarg 148
- callbacks 25
 - for button panels 18
 - for buttons 15
 - passing different variables 10
- callbacks class 24
- :callbacks initarg 18, 500, 508
- callbacks-action-callback accessor function 25
- callbacks-callback-type accessor function 25
- callbacks-extend-callback accessor function 25
- callbacks-retract-callback accessor function 25
- callbacks-selection-callback accessor function 25
- :callback-type initarg 25, 472, 477, 491
- call-editor generic function 27
- :cancel item in :buttons initarg 480

- cancel-button image identifier 483
- :cancel-button initarg 18
- :cancel-function item in :buttons initarg 481
- :cancel-p initarg 13
- CAPI process 87
- capi-object class 28
- capi-object-name accessor function 28
- capi-object-plist accessor function 28
- capi-object-property function 29
- :caret-position initarg 477
- :change-callback initarg 123, 148, 403, 477
- :change-callback-type initarg 477
- :character-format initarg 403
- :checkbox-change-callback initarg 519
- :checkbox-child-function initarg 519
- :checkbox-initial-status initarg 519
- :checkbox-next-map initarg 519
- :checkbox-parent-function initarg 519
- :checkbox-status initarg 519
- check-button class 30
- check-button-panel class 31
- %child% geometry slot 540
- :child initarg 463
- :children-function initarg 167, 517
- choice class 31
- choice-initial-focus-item accessor function 32
- choice-interaction accessor function 32
- choice-selected-item generic function 35
- choice-selected-item-p function 37
- choice-selected-items generic function 37
- choice-selection accessor function 32
- choice-update-item function 39
- class options
 - :coclass 82
 - :definition 75
 - :interfaces 82
 - :layouts 75
 - :menu-bar 75
 - :menus 75
 - :panes 75
 - :source-interfaces 82
- classes
 - arrow-pinboard-object 7
 - button 13
 - button-panel 18
 - callbacks 24
 - capi-object 28
 - check-button 30
 - check-button-panel 31
 - choice 31
 - cocoa-default-application-interface 42
 - cocoa-view-pane 45
 - collection 49
 - collector-pane 55
 - color-screen 56
 - column-layout 57
 - display-pane 92
 - docking-layout 97
 - document-container 102
 - document-frame 103
 - double-headed-arrow-pinboard-object 106
 - double-list-panel 107
 - drawn-pinboard-object 113
 - echo-area-pane 121
 - editor-pane 123
 - element 135
 - ellipse 142
 - expandable-item-pinboard-object 147
 - extended-selection-tree-view 147
 - external-image 578
 - filtering-layout 148
 - foreign-owned-interface 156
 - form-layout 157
 - graph-edge 165
 - graph-node 165
 - graph-object 166
 - graph-pane 167
 - grid-layout 177
 - image 594
 - image-list 183
 - image-pinboard-object 185
 - interactive-pane 190
 - interface 192
 - item 220
 - item-pinboard-object 222
 - labelled-arrow-pinboard-object 222
 - labelled-line-pinboard-object 223
 - layout 224
 - line-pinboard-object 226
 - listener-pane 240
 - list-panel 228
 - list-view 234
 - menu 267
 - menu-component 272
 - menu-item 274
 - menu-object 279

message-pane 283
 mono-screen 285
 multi-column-list-panel 286
 multi-line-text-input-pane 290
 ole-control-component 292
 ole-control-doc 294
 ole-control-frame 295
 ole-control-pane 298
 ole-control-pane-simple-sink 302
 option-pane 303
 output-pane 306
 password-pane 327
 pinboard-layout 329
 pinboard-object 331
 pixmap-port 619
 popup-menu-button 348
 progress-bar 361
 push-button 385
 push-button-panel 386
 radio-button 389
 radio-button-panel 390
 range-pane 392
 rectangle 393
 rich-text-pane 403
 right-angle-line-pinboard-object 411
 row-layout 412
 screen 414
 scroll-bar 421
 shell-pane 450
 simple-layout 452
 simple-network-pane 453
 simple-pane 453
 simple-pinboard-layout 463
 slider 464
 sorted-object 466
 switchable-layout 469
 tab-layout 471
 text-input-choice 475
 text-input-pane 476
 text-input-range 490
 titled-menu-object 493
 titled-object 494
 titled-pinboard-object 497
 title-pane 492
 toolbar 500
 toolbar-button 502
 toolbar-component 507
 toolbar-object 509
 tracking-pinboard-layout 514
 tree-view 517
 x-y-adjustable-layout 549
 clear-external-image-conversions
 function 554
 clear-graphics-port function 555
 clear-graphics-port-state function 555
 clear-rectangle function 556
 clipboard function 40
 clipboard-empty function 41
 clone generic function 42
 :close-callback initarg 299
 :coclass class option 82
 Cocoa Event Loop process 87
 cocoa-default-application-interface
 class 42
 cocoa-view-pane class 45
 cocoa-view-pane-init-function acces-
 sor function 45
 cocoa-view-pane-view function 47
 cocoa-view-pane-view-class accessor
 function 45
 collect-interfaces generic function 48
 collection class 49
 :collection initarg 220
 collection-find-next-string generic
 function 52
 collection-find-string generic func-
 tion 53
 collection-items accessor function 50
 collection-items-count-function
 accessor function 50
 collection-items-get-function acces-
 sor function 50
 collection-items-map-function acces-
 sor function 50
 collection-last-search generic func-
 tion 54
 collection-print-function accessor
 function 50
 collection-search generic function 54
 collection-test-function accessor
 function 50
 collector-pane class 55
 collector-pane-stream accessor function
 55
 color-<component> function 662
 color-database variable 663
 :color-function initarg 228
 color-level function 662, 664
 color-model function 665
 colors= function 665
 color-screen class 56
 :column initarg 177
 :column-function initarg 286

- column-layout class 57
- column-layout-divider 59
- :columns initarg 286
- :combine-child-constraints initarg 469, 472
- :command initarg 450
- Command key 309
- command table 307
- complete-button image identifier 483
- :completion item in :buttons initarg 480
- :completion-function initarg 477
- component-name function 60
- :component-name initarg 298, 302
- compress-external-image function 556
- compute-char-extents function 557
- comtab 307
- Confirm Before Exiting 61, 428
- :confirm-change-function initarg 477
- :confirm-destroy-function initarg 193
- confirmer-pane function 62
- confirm-quit function 60
- confirm-yes-or-no function 61
- contain function 63
- container 103
- container special slot 103
- continuation function, dialog
 - creating 537
 - using 90, 317, 344, 352, 364, 365, 368, 370, 372, 373, 375, 378, 380, 381, 383, 384
- :controller initarg 97
- convert-color function 666
- convert-external-image function 557
- convert-relative-position function 64
- convert-to-font-description function 558
- convert-to-screen function 65
- copy-external-image function 559
- copy-pixels function 559
- copy-transform function 560
- count-collection-items generic function 67
- :create-callback initarg 193, 292, 307
- create-pixmap-port function 561
- current-dialog-handle function 68
- current-document generic function 69
- current-pointer-position function 70
- current-printer function 71
- :cursor initarg 454

D

- :data initarg 220

- :default initarg 224
- :default-button initarg 18
- *default-editor-pane-line-wrap-marker* variable 71
- :default-image-set initarg 500, 508
- *default-image-translation-table* variable 562, 601
- default-library function 72
- :default-p initarg 13
- define-color-alias function 667
- define-color-models macro 668
- define-command macro 72
- define-font-alias function 562
- define-interface macro 74
- define-layout macro 80
- define-menu macro 83
- define-ole-control-component macro 81
- :definition class option 75
- delete-color-translation function 669
- :depth initarg 415
- :description initarg 224, 472
- destroy button
 - removal 200
- destroy generic function 83
- :destroy-callback initarg 193, 293, 307
- destroy-pixmap-port function 563
- detach-simple-sink function 84
- detach-sink function 85
- dialog continuation function
 - creating 537
 - using 90, 317, 344, 352, 364, 365, 368, 370, 372, 373, 375, 378, 380, 381, 383, 384
- dialogs
 - aborting 2
- :disabled-image initarg 13
- :disabled-images initarg 18
- display function 86
- :display-callback initarg 113, 307
- display-dialog function 88
- display-errors macro 91
- display-message function 91
- display-message-for-pane function 92
- display-message-on-screen function 92
- display-pane class 92
- display-pane-text accessor function 93
- display-popup-menu function 94
- display-replacable-dialog function 95
- :display-state initarg 194
- display-tooltip generic function 96
- dither-color-spec function 563

- :divider-p initarg 97
- :dividerp initarg 500
- Dock menu 42
- :docking-callback initarg 97
- docking-layout class 97
- docking-layout-controller accessor function 98
- docking-layout-divider-p accessor function 98
- docking-layout-docking-test-function accessor function 98
- docking-layout-items accessor function 98
- docking-layout-orientation accessor function 98
- docking-layout-pane-docked-p function 101
- docking-layout-pane-visible-p function 101
- :docking-test-function initarg 97
- :dock-menu initarg 43
- document-container class 102
- document-frame class 103
- document-frame-container accessor function 103
- double-headed-arrow-pinboard-object class 106
- :double-head-predicate initarg 106
- double-list-panel class 107
- Drag and drop
 - coordinates 118, 119
 - dragging 109
 - dropping 455
 - effect 116
 - formats 120, 431
 - object 117
- drag-pane-object function 109
- draw-arc function 564
- draw-arcs function 565
- draw-character function 566
- draw-circle function 566
- draw-ellipse function 567
- draw-image function 568
- draw-line function 569
- draw-lines function 570
- draw-metafile function 111
- draw-metafile-to-image function 111
- drawn-pinboard-object class 113
- drawn-pinboard-object-display-callback accessor function 113
- draw-pinboard-object generic function 114

- draw-pinboard-object-highlighted generic function 115
- draw-pinboard-object-unhighlighted generic function 115
- draw-point function 571
- draw-points function 571
- draw-polygon function 572
- draw-polygons function 573
- draw-rectangle function 574
- draw-rectangles function 575
- draw-string function 576
- :drop-callback initarg 455
- :dropdown-menu initarg 503
- :dropdown-menu-function initarg 503
- :dropdown-menu-kind initarg 503
- drop-object-allows-drop-effect-p function 116
- drop-object-drop-effect function 116
- drop-object-get-object function 117
- drop-object-pane-x function 118
- drop-object-pane-y function 119
- drop-object-provides-format function 120

E

- :echo-area initarg 124
- *echo-area-cursor-inactive-style* variable 120
- echo-area-pane class 121
- :edge-pinboard-class initarg 168
- :editing-callback initarg 477
- *editor-cursor-active-style* variable 122
- *editor-cursor-color* variable 121
- *editor-cursor-drag-style* variable 122
- *editor-cursor-inactive-style* variable 123
- editor-pane class 123
- editor-pane-blink-rate generic function 131
- editor-pane-buffer accessor function 131
- editor-pane-change-callback accessor function 124
- editor-pane-enabled accessor function 124
- editor-pane-fixed-fill accessor function 124
- editor-pane-line-wrap-face accessor function 124
- editor-pane-line-wrap-marker accessor

- function 124
- editor-pane-native-blink-rate function 132
- editor-pane-selected-text function 133
- editor-pane-selected-text-p function 133
- editor-pane-stream function 134
- editor-pane-text accessor function 124
- editor-pane-wrap-style accessor function 124
- editor-window generic function 134
- element class 135
- element-container function 140
- element-interface accessor function 137
- element-interface-for-callback function 141
- element-parent accessor function 137
- element-screen function 141
- ellipse class 142
- :enabled initarg 13, 123, 148, 279, 303, 454, 477, 509
- :enabled-function initarg 279, 509
- :enabled-positions initarg 304
- :enabled-slot initarg 279
- :enable-pointer-documentation initarg 194
- :enable-tooltips initarg 194
- :end-x initarg 226
- :end-y initarg 226
- ensure-<command> function 670
- ensure-area-visible generic function 142
- ensure-gdiplus function 576
- ensure-interface-screen function 143
- Escape key 342
- event handler
 - key strokes 308
 - mouse click 308
 - mouse gestures 308
 - mouse move 308
- execute-with-interface function 143
- execute-with-interface-if-alive function 144
- exit-confirmer function 145
- exit-dialog function 146
- expandable-item-pinboard-object class 147
- :expandp-function initarg 518
- :extend-callback initarg 25
- extended-selection-tree-view class 147
- :external-border initarg 194
- external-image class 578

- external-image-color-table function 578, 579
- externalize-image function 580
- :external-max-height initarg 136, 333
- :external-max-width initarg 136, 332
- :external-min-height initarg 136, 332
- :external-min-width initarg 136, 332

F

- :filename initarg 403
- filled accessor function 142, 394
- :filled initarg 142, 394
- filtering-layout class 148
- filtering-layout-matches-text accessor function 149
- filtering-layout-match-object-and-exclude-p function 152
- filtering-layout-state accessor function 149
- find-best-font function 581
- find-graph-edge generic function 152
- find-graph-node generic function 153
- finding panes
 - interfaces 75
- find-interface generic function 154
- find-matching-fonts function 581
- find-pane 75
- find-string-in-collection generic function 155
- :fit-size-to-children initarg 329
- :fixed-fill initarg 124
- :flatp initarg 500
- focus
 - keyboard input on Cocoa 201
 - mouse events on Cocoa 201
 - moving to a new pane 5
 - setting to a pane 321, 434
- :focus-callback initarg 307
- :font initarg 454
- font-description function 582
- font-description-attributes function 583
- font-description-attribute-value function 583
- font-fixed-width-p function 584
- force-screen-update function 155
- force-update-all-screens function 156
- :foreground initarg 454
- foreign-owned-interface class 156
- form-layout class 157
- form-title-adjust accessor function 157
- form-title-gap accessor function 157

- form-vertical-adjust accessor function 157
- form-vertical-gap accessor function 157
- frame 495
- free-image function 584
- free-image-access function 585
- free-metafile function 158
- free-sound function 158
- :from initarg 165
- functions
 - abort-callback 1
 - abort-dialog 2
 - abort-exit-confirmer 3
 - activate-pane 5
 - analyze-external-image 551
 - apply-in-pane-process 6
 - apply-rotation 552
 - apply-scale 552
 - apply-translation 553
 - apropos-color-alias-names 659
 - apropos-color-names 660
 - apropos-color-spec-names 661
 - attach-interface-for-callback 9
 - attach-simple-sink 10
 - attach-sink 11
 - augment-font-description 553
 - beep-pane 12
 - capi-object-property 29
 - choice-selected-item-p 37
 - choice-update-item 39
 - clear-external-image-conversions 554
 - clear-graphics-port 555
 - clear-graphics-port-state 555
 - clear-rectangle 556
 - clipboard 40
 - clipboard-empty 41
 - cocoa-view-pane-view 47
 - color-<component> 662
 - color-level 662, 664
 - color-model 665
 - colors= 665
 - component-name 60
 - compress-external-image 556
 - compute-char-extents 557
 - confirmer-pane 62
 - confirm-quit 60
 - confirm-yes-or-no 61
 - contain 63
 - convert-color 666
 - convert-external-image 557
 - convert-relative-position 64
 - convert-to-font-description 558
 - convert-to-screen 65
 - copy-external-image 559
 - copy-pixels 559
 - copy-transform 560
 - create-pixmap-port 561
 - current-dialog-handle 68
 - current-pointer-position 70
 - current-printer 71
 - default-library 72
 - define-color-alias 667
 - define-font-alias 562
 - delete-color-translation 669
 - destroy-pixmap-port 563
 - detach-simple-sink 84
 - detach-sink 85
 - display 86
 - display-dialog 88
 - display-message 91
 - display-message-for-pane 92
 - display-message-on-screen 92
 - display-popup-menu 94
 - display-replacable-dialog 95
 - dither-color-spec 563
 - docking-layout-pane-docked-p 101
 - docking-layout-pane-visible-p 101
 - drag-pane-object 109
 - draw-arc 564
 - draw-arcs 565
 - draw-character 566
 - draw-circle 566
 - draw-ellipse 567
 - draw-image 568
 - draw-line 569
 - draw-lines 570
 - draw-metafile 111
 - draw-metafile-to-image 111
 - draw-point 571
 - draw-points 571
 - draw-polygon 572
 - draw-polygons 573
 - draw-rectangle 574
 - draw-rectangles 575
 - draw-string 576
 - drop-object-allows-drop-effect-p 116
 - drop-object-drop-effect 116
 - drop-object-get-object 117
 - drop-object-pane-x 118
 - drop-object-pane-y 119
 - drop-object-provides-format 120

editor-pane-native-blink-rate 132
 editor-pane-selected-text 133
 editor-pane-selected-text-p 133
 editor-pane-stream 134
 element-container 140
 element-interface-for-callback 141
 element-screen 141
 ensure-<command> 670
 ensure-gdiplus 576
 ensure-interface-screen 143
 execute-with-interface 143
 execute-with-interface-if-alive
 144
 exit-confirmer 145
 exit-dialog 146
 external-image-color-table 578, 579
 externalize-image 580
 filtering-layout-match-object-
 and-exclude-p 152
 find-best-font 581
 find-matching-fonts 581
 font-description 582
 font-description-attributes 583
 font-description-attribute-value
 583
 font-fixed-width-p 584
 force-screen-update 155
 force-update-all-screens 156
 free-image 584
 free-image-access 585
 free-metafile 158
 free-sound 158
 get-all-color-names 671
 get-bounds 585
 get-character-extent 586
 get-char-ascent 587
 get-char-descent 587
 get-char-width 588
 get-color-alias-translation 672
 get-color-spec 673
 get-constraints 159
 get-enclosing-rectangle 588
 get-font-ascent 589
 get-font-average-width 589
 get-font-descent 590
 get-font-height 590
 get-font-width 591
 get-graphics-state 591
 get-origin 592
 get-page-area 161
 get-printer-metrics 162
 get-scroll-position 163
 get-string-extent 593
 get-transform-scale 593
 graphics-port-transform 594
 graphics-state-background 611
 graphics-state-dash 611
 graphics-state-dashed 611
 graphics-state-fill-style 611
 graphics-state-font 611
 graphics-state-foreground 611
 graphics-state-line-end-style 611
 graphics-state-line-joint-style
 611
 graphics-state-mask 611
 graphics-state-mask-x 611
 graphics-state-mask-y 611
 graphics-state-operation 611
 graphics-state-pattern 611
 graphics-state-scale-thickness 611
 graphics-state-stipple 611
 graphics-state-thickness 611
 graphics-state-transform 611
 graph-pane-edges 174
 graph-pane-nodes 175
 graph-pane-object-at-position 175
 hide-interface 182
 image-access-pixel 595
 image-access-pixels-from-bgra 596
 image-access-pixels-to-bgra 597
 image-access-transfer-from-image
 598
 image-access-transfer-to-image 599
 image-freed-p 600
 image-loader 600
 image-set 186
 image-translation 601
 initialize-dithers 601
 inset-rectangle 602
 inside-rectangle 603
 installed-libraries 189
 install-postscript-printer 187
 interface-display-title 208
 interface-iconified-p 211
 interface-visible-p 216
 invalidate-pane-constraints 218
 invalidate-rectangle 603
 invert-transform 604
 invoke-command 218
 invoke-untranslated-command 219
 line-pinboard-object-coordinates
 227
 list-all-font-names 604
 listener-pane-insert-value 240

load-color-database 674
 load-cursor 241
 load-icon-image 605
 load-image 607
 load-sound 243
 lower-interface 246
 make-dither 609
 make-docking-layout-controller
 247
 make-font-description 610
 make-foreign-owned-interface 248
 make-general-image-set 249
 make-graphics-state 611
 make-gray 675
 make-hsv 676
 make-icon-resource-image-set 250
 make-image 612
 make-image-from-port 614
 make-image-locator 251
 make-menu-for-pane 252
 make-resource-image-set 255
 make-rgb 677
 make-scaled-general-image-set
 256
 make-scaled-image-set 257
 make-sorting-description 258
 make-sub-image 615
 make-transform 616
 map-typeout 266
 merge-font-descriptions 616
 modify-editor-pane-buffer 284
 offset-rectangle 617
 ole-control-add-verbs 291
 ole-control-close-object 292
 ole-control-i-dispatch 296
 ole-control-insert-object 297
 ole-control-ole-object 298
 ole-control-pane-frame 301
 ole-control-user-component 302
 ordered-rectangle-union 618
 page-setup-dialog 316
 pane-close-display 320
 pane-has-focus-p 321
 pixblt 619
 play-sound 328
 popup-confirmer 340
 port-height 620
 port-string-height 620
 port-string-width 621
 port-width 621
 postmultiply-transforms 622
 premultiply-transforms 622
 print-dialog 351
 print-editor-buffer 352
 printer-configuration-dialog 355
 printer-metrics-device-height 356
 printer-metrics-device-width 356
 printer-metrics-dpi-x 357
 printer-metrics-dpi-y 357
 printer-metrics-height 357
 printer-metrics-left-margin 357
 printer-metrics-max-height 357
 printer-metrics-max-width 357
 printer-metrics-min-left-margin
 357
 printer-metrics-min-top-margin 357
 printer-metrics-paper-height 357
 printer-metrics-paper-width 357
 printer-metrics-top-margin 357
 printer-metrics-width 357
 printer-port-handle 358
 printer-port-supports-p 359
 print-file 353
 print-rich-text-pane 354
 print-text 355
 process-pending-messages 361
 prompt-for-color 362
 prompt-for-confirmation 363
 prompt-for-directory 364
 prompt-for-file 366
 prompt-for-files 369
 prompt-for-font 370
 prompt-for-form 371
 prompt-for-forms 373
 prompt-for-integer 374
 prompt-for-items-from-list 375
 prompt-for-number 376
 prompt-for-string 377
 prompt-for-symbol 379
 prompt-for-value 381
 prompt-with-list 382
 prompt-with-message 384
 quit-interface 388
 raise-interface 391
 read-and-convert-external-image
 623
 read-color-db 678
 read-external-image 624
 read-sound-file 393
 rectangle-union 628
 redisplay-menu-bar 395
 redraw-pinboard-layout 396
 redraw-pinboard-object 396
 register-image-load-function 630

- register-image-translation 631
- remove-capi-object-property 397
- replace-dialog 399
- reset-image-translation-table 632
- reuse-interfaces-p 402
- rich-text-pane-character-format 405
- rich-text-pane-operation 406
- rich-text-pane-paragraph-format 410
- rich-text-version 411
- screen-active-interface 416
- screen-active-p 417
- screen-internal-geometry 418
- screen-logical-resolution 417
- screens 418
- selection 423
- selection-empty 424
- separation 633
- set-application-interface 425
- set-clipboard 426
- set-confirm-quit-flag 428
- set-default-editor-pane-blink-rate 428
- set-default-image-load-function 633
- set-default-interface-prefix-suffix 429
- set-drop-object-supported-formats 431
- set-geometric-hint 432
- set-graphics-port-coordinates 634
- set-graphics-state 635
- set-hint-table 432
- set-object-automatic-resize 435
- set-pane-focus 434
- set-printer-metrics 445
- set-printer-options 446
- set-rich-text-pane-character-format 439
- set-rich-text-pane-paragraph-format 442
- set-selection 444
- set-text-input-pane-selection 447
- set-top-level-interface-geometry 447
- show-interface 451
- show-pane 452
- simple-pane-handle 460
- simple-print-port 464
- sort-object-items-by 466
- start-gc-monitor 468

- stop-gc-monitor 469
- tab-layout-panes 474
- tab-layout-visible-child 475
- text-input-copy 486
- text-input-cut 486
- text-input-delete 487
- text-input-pane-complete-text 485
- text-input-pane-selected-text 488
- text-input-pane-selection 489
- text-input-pane-selection-p 489
- text-input-paste 487
- top-level-interface-display-state 510
- top-level-interface-geometry 511
- transform-area 636
- transform-distance 636
- transform-distances 637
- transform-is-rotated 637
- transform-point 638
- transform-points 638
- transform-rect 639
- tree-view-ensure-visible 526
- tree-view-item-checkbox-status 527
- tree-view-item-children-checkbox-status 527
- unconvert-color 679
- undefine-font-alias 640
- uninstall-postscript-printer 530
- unit-transform-p 641
- unmap-typeout 531
- untransform-distance 642
- untransform-distances 643
- untransform-point 643
- untransform-points 644
- update-all-interface-titles 531
- update-pinboard-object 532
- update-screen-interface-titles 533
- update-toolbar 534
- validate-rectangle 645
- wrap-text 547
- wrap-text-for-pane 547
- write-external-image 656

G

- :gap initarg 57, 413
- generic functions
 - accepts-focus-p 4
 - append-items 6
 - calculate-constraints 23
 - calculate-layout 24
 - call-editor 27
 - choice-selected-item 35

choice-selected-items 37
 clone 42
 collect-interfaces 48
 collection-find-next-string 52
 collection-find-string 53
 collection-last-search 54
 collection-search 54
 count-collection-items 67
 current-document 69
 destroy 83
 display-tooltip 96
 draw-pinboard-object 114
 draw-pinboard-object-high-
 lighted 115
 draw-pinboard-object-unhigh-
 lighted 115
 editor-pane-blink-rate 131
 editor-window 134
 ensure-area-visible 142
 find-graph-edge 152
 find-graph-node 153
 find-interface 154
 find-string-in-collection 155
 get-collection-item 159
 get-horizontal-scroll-parame-
 ters 160
 get-vertical-scroll-parameters
 164
 graph-node-children 166
 graph-pane-add-graph-node 171
 graph-pane-delete-object 172
 graph-pane-delete-objects 172
 graph-pane-delete-selected-
 objects 173
 graph-pane-direction 173
 graph-pane-select-graph-nodes
 176
 graph-pane-update-moved-objects
 176
 hide-pane 182
 highlight-pinboard-object 183
 interactive-pane-execute-com-
 mand 191
 interface-display 207
 interface-editor-pane 209
 interface-extend-title 209
 interface-geometry 210
 interface-keys-style 211
 interface-match-p 214
 interface-menu-groups 214
 interface-reuse-p 215
 interpret-description 217
 itemp 221
 list-panel-enabled 233
 locate-interface 244
 make-container 246
 make-image-access 613
 make-pane-popup-menu 253
 manipulate-pinboard 260
 map-collection-items 263
 map-pane-children 263
 map-pane-descendant-children 266
 merge-menu-bars 282
 move-line 285
 over-pinboard-object-p 316
 pane-adjusted-offset 317
 pane-adjusted-position 318
 pane-got-focus 321
 pane-initial-focus 322
 pane-popup-menu-items 323
 pane-string 325
 parse-layout-descriptor 326
 pinboard-object-at-position 335
 pinboard-object-graphics-arg 336
 pinboard-object-overlap-p 337
 pinboard-pane-position 338
 pinboard-pane-size 339
 print-capi-button 349
 print-collection-item 350
 redisplay-collection-item 394
 redisplay-interface 394
 reinitialize-interface 397
 remove-items 398
 replace-items 400
 report-active-component-failure
 401
 scroll 419
 search-for-item 423
 set-button-panel-enabled-items 426
 set-horizontal-scroll-parameters
 433
 set-scroll-position 420
 set-scroll-range 434, 449
 set-vertical-scroll-parameters 448
 simple-pane-visible-height 461
 simple-pane-visible-size 461
 simple-pane-visible-width 462
 sorted-object-sort-by 467
 switchable-layout-switchable-
 children 471
 top-level-interface 509
 top-level-interface-geometry-key
 512
 top-level-interface-p 514

- top-level-interface-save-geometry-p 514
- tree-view-expanded-p 526
- tree-view-update-an-item 528
- tree-view-update-item 529
- unhighlight-pinboard-object 530
- update-interface-title 532
- geometry slots
 - %child% 540
 - %height% 541
 - %max-height% 541
 - %max-width% 541
 - %min-height% 541
 - %min-width% 541
 - %object% 540
 - %ratio% 541
 - %scroll-height% 541
 - %scroll-horizontal-page-size% 541
 - %scroll-horizontal-slug-size% 541
 - %scroll-horizontal-step-size% 541
 - %scroll-start-x% 541
 - %scroll-start-y% 541
 - %scroll-vertical-page-size% 542
 - %scroll-vertical-slug-size% 542
 - %scroll-vertical-step-size% 542
 - %scroll-width% 541
 - %scroll-x% 542
 - %scroll-y% 542
 - %width% 541
 - %x% 540
 - %y% 540
- :geometry-change-callback initarg 193
- get pane
 - interface 75
- get-all-color-names function 671
- get-bounds function 585
- get-character-extent function 586
- get-char-ascent function 587
- get-char-descent function 587
- get-char-width function 588
- get-collection-item generic function 159
- get-color-alias-translation function 672
- get-color-spec function 673
- get-constraints function 159
- get-enclosing-rectangle function 588
- get-font-ascent function 589
- get-font-average-width function 589
- get-font-descent function 590
- get-font-height function 590
- get-font-width function 591
- get-graphics-state function 591
- get-horizontal-scroll-parameters generic function 160
- get-origin function 592
- get-page-area function 161
- get-pane 75
- get-printer-metrics function 162
- get-scroll-position function 163
- get-string-extent function 593
- get-transform-scale function 593
- get-vertical-scroll-parameters generic function 164
- graph-edge class 165
- graph-edge-from accessor function 165
- graph-edge-to accessor function 165
- :graphics-args initarg 332
- :graphics-options initarg 308
- graphics-port-transform function 594
- graphics-state-background function 611
- graphics-state-dash function 611
- graphics-state-dashed function 611
- graphics-state-fill-style function 611
- graphics-state-font function 611
- graphics-state-foreground function 611
- graphics-state-line-end-style function 611
- graphics-state-line-joint-style function 611
- graphics-state-mask function 611
- graphics-state-mask-x function 611
- graphics-state-mask-y function 611
- graphics-state-operation function 611
- graphics-state-pattern function 611
- graphics-state-scale-thickness function 611
- graphics-state-stipple function 611
- graphics-state-thickness function 611
- graphics-state-transform function 611
- graph-node class 165
- graph-node-children generic function 166
- graph-node-height accessor function 166
- graph-node-in-edges accessor function 166
- graph-node-out-edges accessor function 166
- graph-node-width accessor function 166
- graph-node-x accessor function 166
- graph-node-y accessor function 166
- graph-object class 166
- graph-object-element accessor function 167

- graph-object-object accessor function 167
- graph-pane class 167
- graph-pane-add-graph-node generic function 171
- graph-pane-delete-object generic function 172
- graph-pane-delete-objects generic function 172
- graph-pane-delete-selected-objects generic function 173
- graph-pane-direction generic function 173
- graph-pane-edges function 174
- graph-pane-layout-function accessor function 168
- graph-pane-nodes function 175
- graph-pane-object-at-position function 175
- graph-pane-roots accessor function 168
- graph-pane-select-graph-nodes generic function 176
- graph-pane-update-moved-objects generic function 176
- grid-layout class 177
- groupbox 495

H

- :has-root-line initarg 519
- :has-title-column-p initarg 177
- :head initarg 7
- :head-breadth initarg 7
- :head-direction initarg 7
- :header-args initarg 286
- :head-graphics-args initarg 7
- :head-length initarg 7
- %height% geometry slot 541
- :height initarg 415
- help
 - context help 201
 - help-callback 197
- :help item in :buttons initarg 481
- :help-callback initarg 194
- help-key accessor function 50, 137, 275, 333, 503
- :help-key initarg 50, 135, 275, 332, 502
- :help-keys initarg 18
- hide-interface function 182
- hide-pane generic function 182
- highlight-pinboard-object generic function 183

- :highlight-style initarg 329
- :horizontal-scroll initarg 422, 454
- HWND 68, 460

I

- :iconify-callback initarg 193
- image class 594
- image identifiers
 - cancel-button 483
 - complete-button 483
 - ok-button 483
- :image initarg 13, 185, 502
- image-access-pixel function 595
- image-access-pixels-from-bgra function 596
- image-access-pixels-to-bgra function 597
- image-access-transfer-from-image function 598
- image-access-transfer-to-image function 599
- image-freed-p function 600
- :image-function initarg 234, 268, 304, 517
- image-height accessor function 594
- :image-height initarg 184, 500, 518
- image-list class 183
- :image-lists initarg 235, 517
- image-loader function 600
- image-pinboard-object class 185
- image-pinboard-object-image accessor function 185
- :images initarg 18, 500, 507
- image-set function 186
- :image-sets initarg 184
- image-translation function 601
- image-width accessor function 594
- :image-width initarg 184, 500, 518
- :init-function initarg 45
- :initial-focus initarg 194, 224
- :initial-focus-item initarg 32
- initialize-dithers function 601
- input focus 4
- :input-model initarg 307
- :insert-callback initarg 299
- inset-rectangle function 602
- inside-rectangle function 603
- installed-libraries function 189
- install-postscript-printer function 187
- :interaction initarg 13, 32
- interaction styles 15

- interactions
 - for choice 33
- interactive-pane class 190
- interactive-pane-execute-command
 - generic function 191
- interactive-pane-stream accessor function 190
- interactive-pane-top-level-function
 - accessor function 190
- interactive-stream 190
- interactive-stream-stream 190
- interactive-stream-top-level-function 190
- interface class 192
 - :interface initarg 135
- interface-activate-callback accessor function 195
- interface-confirm-destroy-function
 - accessor function 195
- interface-create-callback accessor function 195
- interface-destroy-callback accessor function 195
- interface-display generic function 207
- interface-display-title function 208
- interface-editor-pane generic function 209
- interface-extend-title generic function 209
- interface-geometry generic function 210
- interface-geometry-change-callback
 - accessor function 195
- interface-help-callback accessor function 195
- interface-iconified-p function 211
- interface-iconify-callback accessor function 195, 201
- interface-keys-style generic function 211
- interface-match-p generic function 214
- interface-menu-bar-items accessor function 195
- interface-menu-groups generic function 214
- interface-message-area accessor function 195
- interface-override-cursor accessor function 195
- interface-pointer-documentation-enabled
 - accessor function 195
- interface-reuse-p generic function 215
- :interfaces class option 82
- :interfaces initarg 415

- interface-title accessor function 195
- interface-tooltips-enabled accessor function 195
- interface-visible-p function 216
- interface-window-styles accessor function 195
- internal scrolling 311
 - :internal-border initarg 454
 - :internal-max-height initarg 136, 333
 - :internal-max-width initarg 136, 333
 - :internal-min-height initarg 136, 333
 - :internal-min-width initarg 136, 333
- interpret-description generic function 217
- invalidate-pane-constraints function 218
- invalidate-rectangle function 603
- invert-transform function 604
- invoke-command function 218
- invoke-untranslated-command function 219
- item class 220
- item-collection accessor function 220
- item-data accessor function 220
- item-generic function 221
- item-pinboard-object class 222
- item-print-function accessor function 220
 - :item-print-functions initarg 286
- :items initarg 49, 97, 268, 272, 472
- :items-count-function initarg 49
- item-selected accessor function 220
- :items-function initarg 268, 272
- :items-get-function initarg 49
- :items-map-function initarg 50
- item-text accessor function 220

K

- :keep-selection-p initarg 32
- key press event handler 308
- :key-function initarg 472
- key-press events 308

L

- labelled-arrow-pinboard-object class 222
- labelled-line-pinboard-object class 223
- labelled-line-text-foreground
 - accessor function 223
- :large-image-height initarg 235

- :large-image-width initarg 235
- layout class 224
- :layout initarg 193
- :layout-args initarg 18
- :layout-class initarg 18
- layout-description accessor function 225
- *layout-divider-default-size* 59, 414
- :layout-function initarg 167
- layout-ratios accessor function 57, 413
- :layouts class option 75
- layout-x-adjust accessor function 549
- :layout-x-adjust initarg 168
- layout-x-gap accessor function 178
- layout-x-ratios accessor function 178
- layout-y-adjust accessor function 549
- :layout-y-adjust initarg 168
- layout-y-gap accessor function 178
- layout-y-ratios accessor function 178
- :leaf-node-p-function initarg 518
- line-pinboard-object class 226
- line-pinboard-object-coordinates function 227
- :line-size initarg 421
- :line-wrap-face initarg 124
- :line-wrap-marker initarg 124
- LispWorks as ActiveX control 81, 292
- list-all-font-names function 604
- listener-pane class 240
- listener-pane-insert-value function 240
- list-panel class 228
- list-panel-enabled generic function 233
- list-panel-right-click-selection-behavior accessor function 228
- list-view class 234
- list-view-auto-reset-column-widths accessor function 236
- list-view-columns accessor function 236
- list-view-image-function accessor function 236
- list-view-state-image-function accessor function 236
- list-view-subitem-function accessor function 236
- list-view-subitem-print-functions accessor function 236
- list-view-view accessor function 236
- load-color-database function 674
- load-cursor function 241

- load-icon-image function 605
- load-image function 607
- load-sound function 243
- locate-interface generic function 244
- lookup pane
 - interface 75
- lookup-pane 75
- lower-interface function 246

M

- Mac OS X Dock 42
- macros
 - define-color-models 668
 - define-command 72
 - define-interface 74
 - define-layout 80
 - define-menu 83
 - define-ole-control-component 81
 - display-errors 91
 - rectangle-bind 625
 - rectangle-bottom 626
 - rectangle-height 626
 - rectangle-left 627
 - rectangle-right 627
 - rectangle-top 628
 - rectangle-width 629
 - rect-bind 630
 - undefine-menu 529
 - union-rectangle 640
 - unless-empty-rect-bind 642
 - with-atomic-redisplay 534
 - with-busy-interface 535
 - with-dialog-results 536
 - with-dither 646
 - with-document-pages 538
 - with-external-metafile 538
 - with-geometry 540
 - with-graphics-mask 646
 - with-graphics-rotation 647
 - with-graphics-scale 648
 - with-graphics-state 649
 - with-graphics-transform 650
 - with-graphics-translation 651
 - with-internal-metafile 542
 - with-inverse-graphics 651
 - with-output-to-printer 543
 - without-relative-drawing 652
 - with-page 544
 - with-page-transform 545
 - with-pixmap-graphics-port 652
 - with-print-job 545
 - with-random-typeout 546

- with-transformed-area 653
- with-transformed-point 654
- with-transformed-points 655
- with-transformed-rect 656
- make-container generic function 246
- make-dither function 609
- make-docking-layout-controller function 247
- make-font-description function 610
- make-foreign-owned-interface function 248
- make-general-image-set function 249
- make-graphics-state function 611
- make-gray function 675
- make-hsv function 676
- make-icon-resource-image-set function 250
- make-image function 612
- make-image-access generic function 613
- make-image-from-port function 614
- make-image-locator function 251
- make-menu-for-pane function 252
- make-pane-popup-menu generic function 253
- make-resource-image-set function 255
- make-rgb function 677
- make-scaled-general-image-set function 256
- make-scaled-image-set function 257
- make-sorting-description function 258
- make-sub-image function 615
- make-transform function 616
- manipulate-pinboard generic function 260
- map-collection-items generic function 263
- map-pane-children generic function 263
- map-pane-descendant-children generic function 266
- map-typeout function 266
- :matches-title initarg 149
- :max-characters initarg 477
- %max-height% geometry slot 541
- *maximum-moving-objects-to-track-edges* variable 267
- %max-width% geometry slot 541
- MDI 65, 69, 103
- menu class 267
- :menu initarg 348
- :menu-bar class option 75
- :menu-bar-items initarg 193
- menu-component class 272
- :menu-function initarg 348
- menu-image-function accessor function 268
- menu-item class 274
- menu-items accessor function 268
- menu-object class 279
- menu-object-enabled accessor function 280
- menu-popup-callback accessor function 279
- :menus class option 75
- menu-title accessor function 493
- menu-title-function accessor function 493
- merge-font-descriptions function 616
- merge-menu-bars generic function 282
- :message initarg 495
- :message-area initarg 194
- :message-callback initarg 43
- :message-gap initarg 495
- message-pane class 283
- %min-height% geometry slot 541
- %min-width% geometry slot 541
- :mnemonic initarg 14, 19, 268, 275
- :mnemonic-escape initarg 14, 19, 268, 275
- :mnemonic-text initarg 14, 19
- :mnemonic-title initarg 19, 268, 275, 495
- modal dialogs 90, 344, 536
- modify-editor-pane-buffer function 284
- mono-screen class 285
- mouse clicks 308
- mouse coordinates 70
- mouse events 308
- mouse position 70
- move-line generic function 285
- multi-column-list-panel class 286
- multi-line-text-input-pane class 290
- Multiple Document Interface 65, 69, 103
- multiple-selection interaction style 15

N

- :name initarg 28
- :navigation-callback initarg 477
- :node-pane-function initarg 168
- :node-pinboard-class initarg 168
- no-selection interaction style 15
- :number initarg 415

O

- %object% geometry slot 540
- offset-rectangle function 617

- :ok item in :buttons initarg 480
- ok-button image identifier 483
- OLE control 81, 292
- OLE embedding 81, 292
- ole-control-add-verbs function 291
- ole-control-close-object function 292
- ole-control-component class 292
- ole-control-component-pane accessor function 293
- ole-control-doc class 294
- ole-control-frame class 295
- ole-control-i-dispatch function 296
- ole-control-insert-object function 297
- ole-control-ole-object function 298
- ole-control-pane class 298
- ole-control-pane-frame function 301
- ole-control-pane-simple-sink class 302
- ole-control-user-component function 302
- option-pane class 303
- option-pane-enabled accessor function 304
- option-pane-enabled-positions accessor function 304
- option-pane-image-function accessor function 304
- option-pane-popup-callback accessor function 304
- option-pane-separator-item accessor function 304
- option-pane-visible-items-count accessor function 304
- ordered-rectangle-union function 618
- ordinary scrolling 311
- :orientation initarg 97, 177, 393
- :orientation item in :buttons initarg 482
- output-pane class 306
- output-pane-create-callback accessor function 308
- output-pane-destroy-callback accessor function 308
- output-pane-display-callback accessor function 308
- output-pane-focus-callback accessor function 308
- output-pane-graphics-options accessor function 308
- output-pane-input-model accessor

- function 308
- output-pane-resize-callback accessor function 308
- output-pane-scroll-callback accessor function 308
- over-pinboard-object-p generic function 316
- :override-cursor initarg 194

P

- page-setup-dialog function 316
- :page-size initarg 421
- pane-adjusted-offset generic function 317
- pane-adjusted-position generic function 318
- :pane-can-scroll initarg 307
- pane-close-display function 320
- :pane-function initarg 292
- pane-got-focus generic function 321
- pane-has-focus-p function 321
- pane-initial-focus generic function 322
- pane-layout accessor function 19, 195
- :pane-menu initarg 455
- pane-popup-menu-items generic function 323
- :panes class option 75
- pane-string generic function 325
- :paragraph-format initarg 403
- :parent initarg 135
- parse-layout-descriptor generic function 326
- password-pane class 327
- password-pane-overwrite-character accessor function 327
- :pinboard initarg 332
- pinboard-layout class 329
- pinboard-object class 331
- pinboard-object-active-p accessor function 333
- pinboard-object-at-position generic function 335
- pinboard-object-graphics-arg generic function 336
- pinboard-object-graphics-args accessor function 333
- pinboard-object-overlap-p generic function 337
- pinboard-object-pinboard accessor function 333
- pinboard-pane-position generic function 338

- pinboard-pane-size generic function 339
- pixblt function 619
- pixmap-port class 619
- play-sound function 328
- :plist initarg 28
- :popup-callback initarg 279, 304, 476
- popup-confirmer function 340
- :popup-interface initarg 503
- popup-menu-button class 348
- popup-menu-button-menu accessor function 348
- popup-menu-button-menu-function accessor function 348
- port-height function 620
- port-string-height function 620
- port-string-width function 621
- port-width function 621
- :position item in :buttons initarg 482
- postmultiply-transforms function 622
- *ppd-directory* variable 358
- premultiply-transforms function 622
- :press-callback initarg 385
- print-capi-button generic function 349
- print-collection-item generic function 350
- print-dialog function 351
- print-editor-buffer function 352
- printer-configuration-dialog function 355
- printer-metrics structure type 356
- printer-metrics-device-height function 356
- printer-metrics-device-width function 356
- printer-metrics-dpi-x function 357
- printer-metrics-dpi-y function 357
- printer-metrics-height function 357
- printer-metrics-left-margin function 357
- printer-metrics-max-height function 357
- printer-metrics-max-width function 357
- printer-metrics-min-left-margin function 357
- printer-metrics-min-top-margin function 357
- printer-metrics-paper-height function 357
- printer-metrics-paper-width function 357
- printer-metrics-top-margin function 357

- printer-metrics-width function 357
- printer-port-handle function 358
- printer-port-supports-p function 359
- *printer-search-path* variable 360
- print-file function 353
- :print-function initarg 49, 220, 472
- print-rich-text-pane function 354
- print-text function 355
- process
 - CAPI 87
 - Cocoa Event Loop 87
- process-pending-messages function 361
- progress-bar class 361
- prompt-for-color function 362
- prompt-for-confirmation function 363
- prompt-for-directory function 364
- prompt-for-file function 366
- prompt-for-files function 369
- prompt-for-font function 370
- prompt-for-form function 371
- prompt-for-forms function 373
- prompt-for-integer function 374
- prompt-for-items-from-list function 375
- prompt-for-number function 376
- prompt-for-string function 377
- prompt-for-symbol function 379
- prompt-for-value function 381
- prompt-with-list function 382
- prompt-with-message function 384
- :protected-callback initarg 403
- push-button class 385
- push-button-panel class 386

Q

- quit-interface function 388

R

- radio-button class 389
- radio-button-panel class 390
- raise-interface function 391
- range-callback accessor function 393
- range-end accessor function 393
- range-orientation accessor function 393
- range-pane class 392
- range-slug-end accessor function 393
- range-slug-start accessor function 393
- range-start accessor function 393
- %ratio% geometry slot 541
- :ratios initarg 57, 413
- read-and-convert-external-image

- function 623
- read-color-db function 678
- read-external-image function 624
- read-sound-file function 393
- rectangle class 393
- rectangle-bind macro 625
- rectangle-bottom macro 626
- rectangle-height macro 626
- rectangle-left macro 627
- rectangle-right macro 627
- rectangle-top macro 628
- rectangle-union function 628
- rectangle-width macro 629
- rect-bind macro 630
- redisplay-collection-item generic function 394
- redisplay-interface generic function 394
- redisplay-menu-bar function 395
- redraw-pinboard-layout function 396
- redraw-pinboard-object function 396
- register-image-load-function function 630
- register-image-translation function 631
- reinitialize-interface generic function 397
- :remapped initarg 503
- remove-capi-object-property function 397
- remove-items generic function 398
- replace-dialog function 399
- replace-items generic function 400
- report-active-component-failure generic function 401
- reset-image-translation-table function 632
- resizable
 - dialogs 201
 - elements 139
 - windows 196
- :resize-callback initarg 307
- resizing 139, 196, 201
- :retain-expanded-nodes initarg 518
- :retract-callback initarg 15, 25
- Return key 342
- reuse-interfaces-p function 402
- rich-text-pane class 403
- rich-text-pane-change-callback accessor function 403
- rich-text-pane-character-format function 405

- rich-text-pane-limit accessor function 403
- rich-text-pane-operation function 406
- rich-text-pane-paragraph-format function 410
- rich-text-pane-text accessor function 403
- rich-text-version function 411
- right-angle-line-pinboard-object class 411
- :right-click-extended-match initarg 518
- :right-click-selection-behavior initarg 228
- :roots initarg 167, 517
- row-layout class 412
- row-layout-divider 414
- :rows initarg 178

S

- :save-name initarg 299
- screen
 - usable region of 418
- screen class 414
- screen-active-interface function 416
- screen-active-p function 417
- screen-depth accessor function 415
- screen-height accessor function 415
- screen-height-in-millimeters accessor function 415
- screen-interfaces accessor function 102, 415
- screen-internal-geometry function 418
- screen-logical-resolution function 417
- screen-number accessor function 415
- screens function 418
- screen-width accessor function 415
- screen-width-in-millimeters accessor function 415
- scroll generic function 419
- scroll-bar class 421
- scroll-bar-line-size accessor function 421
- scroll-bar-page-size accessor function 421
- :scroll-callback initarg 307
- %scroll-height% geometry slot 541
- %scroll-horizontal-page-size% geometry slot 541
- %scroll-horizontal-slug-size% geometry slot 541
- %scroll-horizontal-step-size% geometry slot 541

- try slot 541
- scrolling
 - built-in 163
 - internal 311
 - ordinary 311
- %scroll-start-x% geometry slot 541
- %scroll-start-y% geometry slot 541
- %scroll-vertical-page-size% geometry slot 542
- %scroll-vertical-slug-size% geometry slot 542
- %scroll-vertical-step-size% geometry slot 542
- %scroll-width% geometry slot 541
- %scroll-x% geometry slot 542
- %scroll-y% geometry slot 542
- search-for-item generic function 423
- :selected initag 13, 220
- :selected-disabled-image initag 14
- :selected-disabled-images initag 18
- :selected-function initag 275
- :selected-image initag 13, 502
- :selected-images initag 18
- :selected-item initag 32, 525
- :selected-item-function initag 273
- :selected-items initag 32
- :selected-items-function initag 273
- selection function 423
- :selection initag 32
- :selection-callback initag 15, 25, 472
- selection-empty function 424
- :selection-function initag 272
- separation function 633
- :separator-item initag 304
- set-application-interface function 425
- set-button-panel-enabled-items generic function 426
- set-clipboard function 426
- set-confirm-quit-flag function 428
- set-default-editor-pane-blink-rate function 428
- set-default-image-load-function function 633
- set-default-interface-prefix-suffix function 429
- set-drop-object-supported-formats function 431
- set-geometric-hint function 432
- set-graphics-port-coordinates function 634
- set-graphics-state function 635
- set-hint-table function 432
- set-horizontal-scroll-parameters generic function 433
- set-object-automatic-resize function 435
- set-pane-focus function 434
- set-printer-metrics function 445
- set-printer-options function 446
- set-rich-text-pane-character-format function 439
- set-rich-text-pane-paragraph-format function 442
- set-scroll-position generic function 420
- set-scroll-range generic function 434, 449
- set-selection function 444
- set-text-input-pane-selection function 447
- set-top-level-interface-geometry function 447
- :setup-callback-argument initag 279
- set-vertical-scroll-parameters generic function 448
- shell-pane class 450
- shell-pane-command accessor function 450
- show-interface function 451
- show-pane function 452
- :show-value-p initag 465, 467
- simple-layout class 452
- simple-network-pane class 453
- simple-pane class 453
- simple-pane-background accessor function 455
- simple-pane-cursor accessor function 455
- simple-pane-drop-callback accessor function 455
- simple-pane-enabled accessor function 149, 455, 509
- simple-pane-font accessor function 455
- simple-pane-foreground accessor function 455
- simple-pane-handle function 460
- simple-pane-horizontal-scroll accessor function 455
- simple-pane-scroll-callback accessor function 455
- simple-pane-vertical-scroll accessor function 455
- simple-pane-visible-border accessor function 455

- simple-pane-visible-height generic function 461
- simple-pane-visible-size generic function 461
- simple-pane-visible-width generic function 462
- simple-pinboard-layout class 463
- simple-print-port function 464
- single-selection interaction style 15
- :sinks initarg 299
- slider class 464
- slider-show-value-p accessor function 465
- slider-start-point accessor function 465
- :slug-end initarg 392
- :slug-start initarg 392
- :small-image-height initarg 235
- :small-image-width initarg 235
- sorted-object class 466
- sorted-object-sort-by generic function 467
- sort-object-items-by function 466
- :source-interfaces class option 82
- special slots
 - container 103
 - windows-menu 103
- :start initarg 392, 490
- start-gc-monitor function 468
- :start-point initarg 465
- :start-x initarg 226
- :start-y initarg 226
- :state-image-function initarg 235, 517
- :state-image-height initarg 236, 518
- :state-image-width initarg 236, 518
- stop-gc-monitor function 469
- :stream initarg 55
- streams 55
- :stretch-text-p initarg 500
- structure types
 - printer-metrics 356
- :subitem-function initarg 234
- :subitem-print-functions initarg 234
- switchable-layout class 469
- switchable-layout-combine-child-constraints accessor function 470
- switchable-layout-switchable-children generic function 471
- switchable-layout-visible-child accessor function 470

T

- tab-layout class 471
- tab-layout-combine-child-constraints accessor function 472
- tab-layout-panes function 474
- tab-layout-visible-child function 475
- tab-layout-visible-child-function accessor function 472
- tabstops 4
 - :test-function initarg 49
 - :text initarg 93, 123, 148, 220, 403, 477, 490, 492
 - :text-foreground initarg 223
- text-input-choice class 475
- text-input-pane class 476
- text-input-pane-before-change-callback accessor function 483
- text-input-pane-buttons-enabled accessor function 478
- text-input-pane-callback accessor function 478
- text-input-pane-caret-position accessor function 478
- text-input-pane-change-callback accessor function 478
- text-input-pane-complete-text function 485
- text-input-pane-completion-function accessor function 478
- text-input-pane-confirm-change-function accessor function 478
- text-input-pane-copy function 486
- text-input-pane-cut function 486
- text-input-pane-delete function 487
- text-input-pane-editing-callback accessor function 478
- text-input-pane-enabled accessor function 478
- text-input-pane-max-characters accessor function 478
- text-input-pane-navigation-callback accessor function 478
- text-input-pane-paste function 487
- text-input-pane-selected-text function 488
- text-input-pane-selection function 489
- text-input-pane-selection-p function 489
- text-input-pane-text accessor function 478
- text-input-range class 490
- text-input-range-callback accessor

- function 491
- text-input-range-callback-type accessor function 491
- text-input-range-end accessor function 491
- text-input-range-start accessor function 491
- text-input-range-value accessor function 491
- text-input-range-wraps-p accessor function 491
- :text-limit initarg 403
- title bar
 - removal 200
- :title initarg 193, 493, 494
- :title-adjust initarg 157, 494
- :title-args initarg 494
- titled-menu-object class 493
- titled-object class 494
- titled-object-message accessor function 495
- titled-object-message-font accessor function 201, 495
- titled-object-title accessor function 495
- titled-object-title-font accessor function 495
- titled-pane 497
- titled-pane-message 497
- titled-pane-title 497
- titled-pinboard-object class 497
- :title-font initarg 494
- :title-function initarg 493
- :title-gap initarg 157, 494
- title-pane class 492
- title-pane-text accessor function 492
- :title-position initarg 494
- :to initarg 165
- toolbar class 500
- toolbar-button class 502
- toolbar-button-dropdown-menu accessor function 503
- toolbar-button-dropdown-menu-function accessor function 503
- toolbar-button-dropdown-menu-kind accessor function 503
- toolbar-button-image accessor function 503
- toolbar-button-popup-interface accessor function 503
- toolbar-button-selected-image accessor function 503
- toolbar-component class 507
- toolbar-flat-p accessor function 500
- toolbar-object class 509
- toolbar-object-enabled-function accessor function 509
- :tooltip initarg 502
- :tooltips initarg 500, 508
- :top-level-function initarg 190
- :top-level-hook initarg 194
- top-level-interface generic function 509
- top-level-interface-display-state function 510
- top-level-interface-external-border accessor function 195
- top-level-interface-geometry function 511
- top-level-interface-geometry-key generic function 512
- top-level-interface-p generic function 514
- top-level-interface-save-geometry-p generic function 514
- top-level-interface-transparency accessor function 195
- tracking-pinboard-layout class 514
- transform type 635
- transform-area function 636
- transform-distance function 636
- transform-distances function 637
- transform-is-rotated function 637
- transform-point function 638
- transform-points function 638
- transform-rect function 639
- :transparency initarg 194
- tree-view class 517
- tree-view-action-callback-expand-p accessor function 520
- tree-view-checkbox-change-callback accessor function 520
- tree-view-checkbox-child-function accessor function 520
- tree-view-checkbox-initial-status accessor function 520
- tree-view-checkbox-next-map accessor function 520
- tree-view-checkbox-parent-function accessor function 520
- tree-view-checkbox-status accessor function 520
- tree-view-children-function accessor function 520

- tree-view-ensure-visible function 526
- tree-view-expanded-p generic function 526
- tree-view-expandp-function accessor function 520
- tree-view-has-root-line accessor function 520
- tree-view-image-function accessor function 520
- tree-view-item-checkbox-status function 527
- tree-view-item-children-checkbox-status function 527
- tree-view-leaf-node-p-function accessor function 520
- tree-view-retain-expanded-nodes accessor function 520
- tree-view-right-click-extended-match accessor function 520
- tree-view-roots accessor function 520
- tree-view-state-image-function accessor function 520
- tree-view-update-an-item generic function 528
- tree-view-update-item generic function 529
- :type initarg 412
- types
 - transform 635

U

- unconvert-color function 679
- undefine-font-alias function 640
- undefine-menu macro 529
- unhighlight-pinboard-object generic function 530
- :uniform-size-p initarg 57, 413
- uninstall-postscript-printer function 530
- union-rectangle macro 640
- *unit-transform* variable 641
- unit-transform-p function 641
- unless-empty-rect-bind macro 642
- unmap-timeout function 531
- untransform-distance function 642
- untransform-distances function 643
- untransform-point function 643
- untransform-points function 644
- update-all-interface-titles function 531
- update-interface-title generic func-

- tion 532
- update-pinboard-object function 532
- *update-screen-interfaces-hooks* variable 533
- update-screen-interface-titles function 533
- update-toolbar function 534
- :use-images initarg 518
- :use-large-images initarg 235
- :user-component initarg 298
- :use-small-images initarg 235
- :use-state-images initarg 235, 518

V

- validate-rectangle function 645
- variables
 - *color-database* 663
 - *default-editor-pane-line-wrap-marker* 71
 - *default-image-translation-table* 562, 601
 - *echo-area-cursor-inactive-style* 120
 - *editor-cursor-active-style* 122
 - *editor-cursor-color* 121
 - *editor-cursor-drag-style* 122
 - *editor-cursor-inactive-style* 123
 - *maximum-moving-objects-to-track-edges* 267
 - *ppd-directory* 358
 - *printer-search-path* 360
 - *unit-transform* 641
 - *update-screen-interfaces-hooks* 533
- :vertical-adjustment initarg 157
- :vertical-gap initarg 157
- :vertical-scroll initarg 422, 454
- :view initarg 234, 235
- :view-class initarg 45
- :visible-border initarg 454
- :visible-child initarg 469
- :visible-child-function initarg 472
- :visible-items-count initarg 304, 476
- :visible-max-height initarg 136, 333
- :visible-max-width initarg 136, 333
- :visible-min-height initarg 136, 333
- :visible-min-width initarg 136, 333

W

- %width% geometry slot 541
- :width initarg 415

- windoid 201
- Window handle 68, 460
- window title
 - removal 200
- window-modal dialogs 90, 344, 536
- windows-menu 104
- windows-menu special slot 103
- :window-styles initarg 194
- with-atomic-redisplay macro 534
- with-busy-interface macro 535
- with-dialog-results macro 536
- with-dither macro 646
- with-document-pages macro 538
- with-external-metafile macro 538
- with-geometry macro 540
- with-graphics-mask macro 646
- with-graphics-rotation macro 647
- with-graphics-scale macro 648
- with-graphics-state macro 649
- with-graphics-transform macro 650
- with-graphics-translation macro 651
- with-internal-metafile macro 542
- with-inverse-graphics macro 651
- with-output-to-printer macro 543
- without-relative-drawing macro 652
- with-page macro 544
- with-page-transform macro 545
- with-pixmap-graphics-port macro 652
- with-print-job macro 545
- with-random-typeout macro 546
- with-transformed-area macro 653
- with-transformed-point macro 654
- with-transformed-points macro 655
- with-transformed-rect macro 656
- :wraps-p initarg 490
- :wrap-style initarg 124
- wrap-text function 547
- wrap-text-for-pane function 547
- write-external-image function 656

X

- %x% geometry slot 540
- :x initarg 136, 332
- X window ID 68, 460
- X Window System
 - display 65
 - fallback resources 65
- :x-adjust initarg 549
- :x-gap initarg 178, 453
- :x-ratios initarg 178
- :x-uniform-size-p initarg 178
- x-y-adjustable-layout class 549

Y

- %y% geometry slot 540
- :y initarg 136, 332
- :y-adjust initarg 549
- :y-gap initarg 178
- :y-ratios initarg 178
- :y-uniform-size-p initarg 178

Z

- Z-order
 - of interfaces 49
 - of pinboard-objects 226, 330

